6 Preorder Relations*

Stefan D. Bruda

Department of Computer Science
Bishop's University
Lennoxville, Quebec J1M 1Z7, Canada
Email: bruda@cs.ubishops.ca

6.1 Introduction

The usefulness of formalisms for the description and the analysis of reactive systems is closely related to the underlying notion of *behavioral equivalence*. Such an equivalence should formally identify behaviors that are informally indistinguishable from each other, and at the same time distinguish between behaviors that are informally different.

One way of determining behavioral equivalences is by observing the systems we are interesting in, experimenting on them, and drawing conclusions about the behavior of such systems based on what we see. We refer to this activity as *testing*. We then consider a set of relevant observers (or tests) that interact with our systems; the tests are carried out by human or by machine, in many different ways (i.e., by using various means of interaction with the system being tested).

In this context, we may be interested in finding out whether two systems are equivalent; for indeed two equivalent (sub)systems can then be replaced with each other without affecting the overall functionality, and we may also want to compare the specification of a system with its implementation to determine whether we actually implemented what we wanted to implement. We could then create an equivalence relation between systems, as follows: two systems are equivalent (with respect to the given tests) if they pass exactly the same set of tests. Such an equivalence can be further broken down into **preorder relations** on systems, i.e., relations that are reflexive and transitive (though not necessarily symmetric).

Preorders are in general easier to deal with, and one can reconstruct an equivalence relation by studying the preorder that generates it. Preorders are also more convenient—indeed, more meaningful—than equivalences in comparing specifications and their implementation: If two systems are found to be in a preorder relation with each other, then one is the implementation of the other, in the sense that the implementation is able to perform the same actions upon its computational environment as the other system (by contrast with equivalences the implementation may be now able to perform more actions, but this is immaterial as far as the capacity to implement is concerned). Preorders can thus be practically interpreted as *implementation relations*.

Recall from the first paragraph that we are interested in a formal approach to systems and their preorders. We are thus not interested how this system is built, whether by system we mean a reactive program or a protocol, they are all representable from a

* This work was supported by the Natural Sciences and Engineering Research Council of Canada, and by the Fond québécois de recherche sur la nature et les technologies.

behavioral point of view by a common model. We shall refer to the behavior of a system as a *process*, and we start this chapter by offering a formal definition for the notion of process.

Depending on the degree of interaction with processes that we consider allowable, many preorder relations can be defined, and many have been indeed defined. In this chapter we survey the most prominent preorder relations over processes that have been developed over time. We leave the task of actually using these preorders to subsequent chapters.

Preorders are not created equal. Different preorders are given by varying the ability of our tests to examine the processes we are interested in. For example we may restrict our tests and only allow them to observe the processes, but we may also allow our tests to interact with the process being observed in some other ways. By determining the abilities of the tests we establish a *testing scenario*, under the form of a set of tests. By varying the testing scenario—i. e., the capabilities of tests to extract information about the process being tested—we end up with different preorders. We start with a generic testing scenario, and then we vary it and get a whole bunch of preorders in return.

It is evident that one testing scenario could be able to extract more information about processes (and thus to differentiate more between them). It is however not necessarily true that more differentiation between processes is better, simply because for some particular application a higher degree of differentiation may be useless. It is also possible that one testing scenario may be harder to implement[1] than another. In our discussion about various testing scenarios and their associated preorders we shall always keep in mind these practical considerations, and compare the preorders in terms of how much differentiation they make between processes, but also in terms of the practical realization of the associated testing scenario. In other words, we keep wondering how difficult is to convince the process being tested to provide the information or perform the actions required by the testing scenario we have in mind. For instance, it is arguably harder to block possible future action of the process under test (as we need to do in the testing scenario inducing the refusal preorder and presented in Section 6.6 on page 152) than to merely observe the process and write down the actions that have been performed (as is the case with the testing scenario inducing trace preorders presented in Section 6.3 on page 142). The increase in differentiation power of refusal preorder over trace preorder comes thus at a cost which may or may not be acceptable in practice.

One reason for which practical considerations are of interest is that preorders are a key element in conformance testing [Tre94]. In such a framework we are given a formal specification and a possible implementation. The implementation is treated as a black box (perhaps somebody else wrote a poorly commented piece of code) exhibiting some external behavior. The goal is then to determine by means of testing whether the implementation implements correctly the specification. Such a goal induces naturally an implementation relation, or a preorder. Informally, the practical use of a preorder relation $\sqsubseteq$ consists then in the algorithmic problem of determining whether $s \sqsubseteq i$ for

[1] Implementing a testing scenario means implementing the means of interaction between a process and a test within the scenario. Implementing a preorder then means implementing an algorithm that takes two processes and determines whether they are in the given preorder relation or not by applying tests from the associated testing scenario.

two processes i (the implementation) and s (the specification) by means of applying on the two processes tests taken from the testing scenario associated with $\sqsubseteq$. If the relation holds then i implements (or conforms to) s (according to the respective testing scenario). The formal introduction of conformance testing is left to the end of this chapter, namely to Section 6.9 on page 159 to which we direct the interested reader for details. For now we get busy with defining preorders and analyzing their properties.

Where we go from here We present in the next section the necessary preliminaries related to process representation and testing (including a first preorder to compare things with). Sections 6.3 to 6.8 are then the main matter of this chapter; we survey here the most prominent preorders and we compare them with each other. We also include a presentation of conformance testing in Section 6.9.

6.1.1 Notations and Conventions

It is often the case that our definitions of various sets (and specifically inductive definitions) should feature a final item containing a statement along the line that "nothing else than the above constructions belong to the set being defined." We consider that the presence of such an item is understood and we shall not repeat it over and over. "Iff" stands for "if and only if." We denote the empty string, and only the empty string by ε.

We present a number of concepts throughout this chapter based on one particular paper [vG01] without citing it all the time, in order to avoid tiresome repetitions.

Many figures show processes that are compared throughout the paper using various preorders. We show parenthetically in the captions of such figures the most relevant relations established between the depicted processes. Parts of these parenthetical remarks do not make sense when the figures are first encountered, but they will reveal themselves as the reader progresses through the chapter.

6.2 Process Representation and Testing

Many formal descriptions for processes have been developed in the past, most notably under the form of process algebraic languages such as CCS [Mil80] and LOTOS [BB87]. The underlying semantics of all these descriptions can be described by **labeled transition systems**. We will use in what follows the labeled transition system as our semantical model (feeling free to borrow concepts from other formalisms whenever they simplify the presentation).

Our model is a slight variation of the model presented in Appendix 22 in that we need a notion of divergence for processes, and we introduce the concept of derived transition system; in addition, we enrich the terminology in order to blend the semantic model into the bigger picture on an intuitive level. For these reasons we also offer here a short presentation of labeled transition systems [vG01, Abr87]. Our presentation should be considered a complement to, rather than a replacement for Appendix 22.

6.2.1 Processes, States, and Labeled Transition Systems

Processes are capable of performing *actions* from a given, countable set Act. By action we mean any activity that is a conceptual entity at a given, arbitrary level of abstraction; we do not differentiate between, say input actions and output actions. Different activities that are indistinguishable on the chosen level of abstraction are considered occurrences of the same action.

What action is taken by a process depends on the *state* of the process. We denote the countable set of states by $\mathbf{Q}$. A process goes from a state to another by performing an action. The behavior of the process is thus given by the *transition relation* $\rightarrow \subseteq \mathbf{Q} \times \mathsf{Act} \times \mathbf{Q}$.

Sometimes a process may go from a state to another by performing an internal action, independent of the environment. We denote such an action by τ, where $\tau \notin \mathsf{Act}$.

The existence of partially defined states stem from (and facilitate) the semantic of sequential computations (where Ω is often used to denote a partial program whose behavior is totally undefined). The existence of such states is also useful for reactive programs. They are thus introduced by a *divergence predicate* $\uparrow$ ranging over $\mathbf{Q}$ and used henceforth in postfix notation; a state p for which $p \uparrow$ holds is a "partial state," in the sense that its properties are undefined; we say that such a state diverges (is divergent, etc.). The opposite property (that a state converges) is denoted by the postfix operator $\downarrow$.

Note that divergence (and thus convergence) is a property that is inherent to the state; in particular, it does not have any relation whatsoever with the actions that may be performed from the given state. Consider for example state x from Figure 6.4 on page 145 (where states are depicted by nodes, and the relation $\rightarrow$ is represented by arrows between nodes, labeled with actions). It just happens that x features no outgoing actions, but this does not make it divergent (though it may be divergent depending on the definition of the predicate $\uparrow$ for the respective labeled transition system). Divergent states stand intuitively for some form of error condition in the state itself, and encountering a divergent state during testing is a sure sign of failure for that test.

To summarize all of the above, we offer the following definition:

Definition 6.1. A **labeled transition system with divergence** (simply labeled transition system henceforth in this chapter) is a tuple $(\mathbf{Q}, \mathsf{Act} \cup \{\tau\}, \rightarrow, \uparrow)$, where $\mathbf{Q}$ is a countable set of states, Act is a countable set of (atomic) actions, $\rightarrow$ is the transition relation, $\rightarrow \subseteq \mathbf{Q} \times (\mathsf{Act} \cup \{\tau\}) \times \mathbf{Q}$, and $\uparrow$ is the divergence predicate. By τ we denote an internal action, $\tau \notin \mathsf{Act}$.

For some state $p \in \mathbf{Q}$ we write $p \downarrow$ iff $\neg\ (p \uparrow)$. Whenever $(q, a, p) \in \rightarrow$ we write $p \xrightarrow{a} q$ (to be read "p offers a and after executing a becomes q"). We further extend this notation to the reflexive and transitive closure of $\rightarrow$ as follows: $p \xrightarrow{\varepsilon} p$ for any $p \in \mathbf{Q}$; and $p \xrightarrow{\sigma} q$, with $\sigma \in \mathbf{Q}^*$, iff $\sigma = \sigma_1\sigma_2$ and there exists $q' \in \mathbf{Q}$ such that $p \xrightarrow{\sigma_1} q' \xrightarrow{\sigma_2} q$. □

We use the notation $p \xrightarrow{\sigma}$ as a shorthand for "there exists $q \in \mathbf{Q}$ such that $p \xrightarrow{\sigma} q$," and the notation $\not\rightarrow$ as the negation of $\rightarrow$ ($p \xrightarrow{a} \not\rightarrow q$ iff it is not the case that $p \xrightarrow{a} q$, etc.).

Assume now that we are given a labeled transition system. The internal action τ is unobservable. In order to formalize this unobservability, we define an associated derived transition system in which we hide all the internal actions; the transition relation $\Longrightarrow$ of such a system ignores the actions τ performed by the system. Formally, we have:

Definition 6.2. Given a transition system $B = (\mathbf{Q}, \mathsf{Act} \cup \{\tau\}, \rightarrow, \uparrow_B)$, its **derived transition system** is a tuple $D = (\mathbf{Q}, \mathsf{Act} \cup \{\varepsilon\}, \Longrightarrow, \uparrow)$, where $\Longrightarrow \subseteq \mathbf{Q} \times (\mathsf{Act} \cup \{\varepsilon\}) \times \mathbf{Q}$ and is defined by the following relations:

$$p \overset{a}{\Longrightarrow} q \text{ iff } p \xrightarrow{\tau^* a} q$$
$$p \overset{\varepsilon}{\Longrightarrow} q \text{ iff } p \xrightarrow{\tau^*} q$$

The divergence predicate is defined as follows: $p \uparrow$ iff there exists q such that $q \uparrow_B$ and $p \overset{\varepsilon}{\Longrightarrow} q$, or there exists a sequence $(p_i)_{i \geq 0}$, such that $p_0 = p$ and for any $i > 0$ it holds that $p_i \xrightarrow{\tau} p_{i+1}$. □

In passing, note that we deviate slightly in Definition 6.2 from the usual definition of $\Longrightarrow$ ($p \overset{a}{\Longrightarrow} q$ iff $p \xrightarrow{\tau^* a \tau^*} q$, see Appendix 22), as this allows for a clearer presentation.

Also note that a state can diverge in two ways in a derived transition system: it can either perform a number of internal actions and end up in a state that diverges in the associated labeled transition system, or evolve perpetually into new states by performing internal actions. Therefore this definition does not make distinction between *deadlock* (first case) and *livelock* (second variant). We shall discuss in subsequent sections whether such a lack of distinction is a good or a bad thing, and we shall distinguish between these variants using the original labeled transition system (since the derived system is unable to make the distinction).

It is worth emphasizing once more (this time using an example) that the definition of divergence in a derived transition system is different from the correspondent definition in a labeled transition system. Indeed, consider state y from Figure 6.6 on page 147 (again, states are depicted by nodes, and the relation $\rightarrow$ is represented by arrows between nodes, labeled with actions). It may be the case that y is a nice, convergent state in the respective labeled transition system (i.e., $y \downarrow_B$). Still, it is obviously the case that $y \uparrow$ in the derived transition system (we refer to this as "y may diverge" instead of "y diverges," given that y may decide at some time to perform action b and get out of the loop of internal actions).

Again, we shall use in what follows natural extensions of the relation $\Longrightarrow$ such as $p \overset{a}{\Longrightarrow}$ and $\Longrightarrow$. We also use by abuse of notation the same operator for the reflexive and transitive closure of $\Longrightarrow$ (in the same way as we did for $\rightarrow$).

A transition system gives a description of the actions that can be performed by a process depending on the state that process is in. A process does in addition start from an *initial state*. In other words, a process is fully described by a transition system and an initial state. In most cases we find it convenient to fix a global transition system for all the processes under consideration. In this setting, a process is then uniquely defined by its initial state. We shall then blur the distinction between a process and a state, often referring to "the process $p \in \mathbf{Q}$."

Finally, a process can be represented as a tree in a natural way: Tree nodes represent states. The root node is the initial state. The edges of the tree will be labeled by actions,

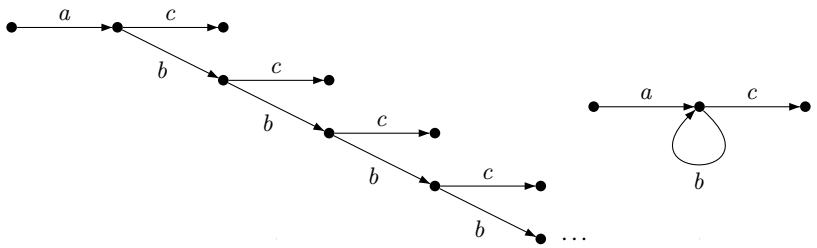

Fig. 6.1. Representation of infi nite process trees: an infi nite tree (a), and its graph representation (b).

and there exists an edge between nodes p and q labeled with a iff it holds that $p \xrightarrow{a} q$ in the given transition system (or that $p \overset{a}{\Longrightarrow} q$ if we talk about a derived transition system). We shall not make use of this representation except when we want to represent a process (or part thereof) graphically for illustration purposes. Sometimes we find convenient to "abbreviate" tree representation by drawing a graph rather than a tree when we want to represent infinite trees with states whose behavior repeats over and over (in which case we join those states in a loop). The reader should keep in mind that this is just a convenient representation, and that in fact she is in front of a finite representation of an infinite tree. As an example, Figure 6.1 shows such a graph together with a portion of the unfolded tree represented by the graph.

Two important properties of transition systems are **image-finiteness** and **sort-finiteness**. A transition system is image-finite if for any $a \in \mathsf{Act}$, $p \in \mathbf{Q}$ the set $\{q \in \mathbf{Q} \mid p \xrightarrow{a} q\}$ is finite, and is sort-finite if for any $p \in \mathbf{Q}$ the set $\{a \in \mathsf{Act} \mid \exists \sigma \in \mathsf{Act}^*, \exists\, q \in \mathbf{Q}$ such that $p \xrightarrow{\sigma} q \xrightarrow{a}\}$ is finite. This definition also applies to derived transition systems.

In all of the subsequent sections we shall assume a transition system $(\mathbf{Q}, \mathsf{Act} \cup \{\tau\}, \rightarrow, \restriction_B)$ with its associated derived transition system $(\mathbf{Q}, \mathsf{Act} \cup \{\tau\}, \Longrightarrow, \restriction)$, applicable to all the processes under scrutiny; thus a process shall be identified only by its initial state.

6.2.2 Processes and Observations

As should be evident from the need of defining derived transition systems, we can determine the characteristics of a system by performing observations on it. Some observations may reveal the whole internal behavior of the system being inspected, some may be more restricted.

In general, we may think of a set of processes and a set of relevant observers (or *tests*). Observers may be thought of as agents performing observations. Observers can be viewed themselves as processes, running in parallel with the process being observed and synchronizing with it over visible actions. We can thus represent the observers as labeled transition systems, just as we represent processes; we prefer however to use a different, "denotational" syntax for observers in our presentation.

Assume now that we have a predefined set $\mathcal{O}$ of observers. The effect of observers performing tests is formalized by considering that for every observer o and process p there exists a set of runs $\textsc{Runs}(o, p)$. If we have $r \in \textsc{Runs}(o, p)$ then the result of o testing p may be the run r.

We take the outcomes of particular runs of a test as being success or failure [Abr87, dNH84] (though we shall differentiate between two kinds of failure later). We then represent outcomes as elements in the two-point lattice

$$\mathbb{O} \stackrel{\text{def}}{=} \begin{array}{c} \top \\ | \\ \bot \end{array}$$

The notion of failure incorporates divergence, so for some observer o and some process p, the elements of $\mathbb{O}$ have the following meaning:

- the outcome of o testing p is $\top$ if there exists $r \in \textsc{Runs}(o, p)$ such that r is successful;
- the outcome of o testing p is $\bot$ if there exists $r \in \textsc{Runs}(o, p)$ such that either r is unsuccessful, or r contains a state q such that $q \uparrow$ and q is not preceded by a successful state.

Note that for the time being we do not differentiate between runs with a deadlock (i.e., in which a computation terminates without reaching a successful state) and runs that diverge; the outcome is $\bot$ in both cases.

Processes may be nondeterministic, so there may be different runs of a given test on a process, with different outcomes. In effect, the (overall) outcome of an observer testing a process is a *set*, and therefore we are led to use powerdomains of $\mathbb{O}$. In fact, we have three possible powerdomains:

$$\mathbb{P}_{\text{may}} \stackrel{\text{def}}{=} \begin{array}{c} \{\top\} = \{\bot, \top\} \\ | \\ \{\bot\} \end{array} \qquad \mathbb{P}_{\text{conv}} \stackrel{\text{def}}{=} \begin{array}{c} \{\top\} \\ | \\ \{\bot, \top\} \\ | \\ \{\bot\} \end{array} \qquad \mathbb{P}_{\text{must}} \stackrel{\text{def}}{=} \begin{array}{c} \{\top\} \\ | \\ \{\bot\} = \{\bot, \top\} \end{array}$$

The names of the three powerdomains are not chosen haphazardly. By considering $\mathbb{P}_{\text{may}}$ as possible outcomes we identify processes that *may* pass a test in order to be considered successful. Similarly, $\mathbb{P}_{\text{must}}$ identifies tests that *must* be successful, and by using $\mathbb{P}_{\text{conv}}$ we combine the may and must properties. The partial order relations induced by the lattices $\mathbb{P}_{\text{may}}$, $\mathbb{P}_{\text{must}}$, and $\mathbb{P}_{\text{conv}}$ shall be denoted by $\subseteq_{\text{may}}$, $\subseteq_{\text{must}}$, and $\subseteq_{\text{conv}}$, respectively.

We also need to introduce the notion of **refusal**. A process refuses an action if the respective action is not applicable in the current state of the process, and there is no internal transition to change the state (so that we are sure that the action will not be applicable unless some other visible action is taken first).

Definition 6.3. Process $p \in \mathbf{Q}$ refuses action $a \in \mathsf{Act}$, written p **ref** a, iff $p \downarrow_B$, $p \stackrel{\tau}{\not\longrightarrow}$, and $p \stackrel{a}{\not\longrightarrow}$. □

We thus described the notions of test and test outcomes. We also introduce at this point a syntax for tests. In fact tests are as we mentioned just processes that interact with the process under test, so we can represent tests in the same way as we represent processes. Still, we find convenient to use a "denotational" representation for tests since we shall refer quite often to such objects. We do this by defining a set $\mathcal{O}$ of test expressions.

While we are at it, we also define the "semantics" of tests, i.e., the way tests are allowed to interact with the processes being tested. Such a semantics for tests is defined using a function obs : $\mathcal{O} \times \mathbf{Q} \to \mathcal{P}$, where $\mathcal{P} \in \{\mathbb{P}_{\text{may}}, \mathbb{P}_{\text{conv}}, \mathbb{P}_{\text{must}}\}$ such that obs(o, p) is the set of all the possible outcomes.

To concretize the concepts of syntax and semantics, we introduce now our first *testing scenario* (i.e., set of test expressions and their semantics), of *observable testing equivalence*[2][Abr87]. This is a rather comprehensive testing model, which we will mostly restrict in order to introduce other models—indeed, we shall restrict this scenario in all but one of our subsequent presentations. A concrete model for tests also allows us to introduce our first preorder.

For the remainder of this section, we fix a transition system $(\mathbf{Q}, \mathsf{Act} \cup \{\tau\}, \to, \uparrow_B)$ together with its derived transition system $(\mathbf{Q}, \mathsf{Act} \cup \{\varepsilon\}, \Longrightarrow, \uparrow)$.

Definition 6.4. The set $\mathcal{O}$ of test expressions inducing the observable testing equivalence contains exactly all of the following constructs, with o, o_1, and o_2 ranging over $\mathcal{O}$:

$$
\begin{aligned}
o \stackrel{\text{def}}{=}\ & \textsc{Succ} && (6.1)\\
\mid\ & \textsc{Fail} && (6.2)\\
\mid\ & ao \qquad \text{for } a \in \mathsf{Act} && (6.3)\\
\mid\ & \widetilde{a}o \qquad \text{for } a \in \mathsf{Act} && (6.4)\\
\mid\ & \varepsilon o && (6.5)\\
\mid\ & o_1 \wedge o_2 && (6.6)\\
\mid\ & o_1 \vee o_2 && (6.7)\\
\mid\ & \forall\, o && (6.8)\\
\mid\ & \exists\, o && (6.9)
\end{aligned}
$$

□

Intuitively, Expressions (6.1) and (6.2) state that a test can succeed or fail by reaching two designated states Succ and Fail, respectively. A test may check whether an action can be taken when into a given state, or whether an action is not possible at all; these are expressed by (6.3) and (6.4). We can combine tests by means of boolean operators using expressions of form (6.6) and (6.7). By introducing tests of form (6.5) we allow a process to "stabilize" itself through internal actions. Finally, we have universal and existential quantifiers for tests given by (6.8) and (6.9). Nondeterminism is introduced in the tests themselves by the Expressions (6.7) and (6.9), the latter being a generalization of the former.

[2] Just *testing equivalence* originally [Abr87]; we introduce the new, awkward terminology because the original name clashes with the names of preorders introduced subsequently.

$\wedge$	$\bot$	$\top$
$\bot$	$\bot$	$\bot$
$\top$	$\bot$	$\top$

$\wedge$	$\{\bot\}$	$\{\bot,\top\}$	$\{\top\}$
$\{\bot\}$	$\{\bot\}$	$\{\bot\}$	$\{\bot\}$
$\{\bot,\top\}$	$\{\bot\}$	$\{\bot,\top\}$	$\{\bot,\top\}$
$\{\top\}$	$\{\bot\}$	$\{\bot,\top\}$	$\{\top\}$

$\forall$	
$\{\bot\}$	$\{\bot\}$
$\{\bot,\top\}$	$\{\bot\}$
$\{\top\}$	$\{\top\}$

$\vee$	$\bot$	$\top$
$\bot$	$\bot$	$\top$
$\top$	$\top$	$\top$

$\vee$	$\{\bot\}$	$\{\bot,\top\}$	$\{\top\}$
$\{\bot\}$	$\{\bot\}$	$\{\bot,\top\}$	$\{\top\}$
$\{\bot,\top\}$	$\{\bot,\top\}$	$\{\bot,\top\}$	$\{\top\}$
$\{\top\}$	$\{\top\}$	$\{\top\}$	$\{\top\}$

$\exists$	
$\{\bot\}$	$\{\bot\}$
$\{\bot,\top\}$	$\{\top\}$
$\{\top\}$	$\{\top\}$

Fig. 6.2. Semantics of logical operators on test outcomes.

Definition 6.5. With the semantics of logical operators as defined in Figure 6.2, the function obs inducing the observable testing equivalence, $\mathrm{obs} : O \times \mathbf{Q} \to \mathbb{P}_{\mathrm{conv}}$, is defined as follows:

$$
\begin{aligned}
\mathrm{obs}(\textsc{Succ}, p) &= \{\top\} \\
\mathrm{obs}(\textsc{Fail}, p) &= \{\bot\} \\
\mathrm{obs}(a o, p) &= \bigcup \{\mathrm{obs}(o, p') \mid p \stackrel{a}{\Longrightarrow} p'\} \cup \{\bot \mid p \uparrow\} \cup \{\bot \mid p \stackrel{\varepsilon}{\Longrightarrow} p', p' \textbf{ ref } a\} \\
\mathrm{obs}(\widetilde{a}\, o, p) &= \bigcup \{\mathrm{obs}(o, p') \mid p \stackrel{a}{\Longrightarrow} p'\} \cup \{\bot \mid p \uparrow\} \cup \{\top \mid p \stackrel{\varepsilon}{\Longrightarrow} p', p' \textbf{ ref } a\} \\
\mathrm{obs}(\varepsilon o, p) &= \bigcup \{\mathrm{obs}(o, p') \mid p \stackrel{\varepsilon}{\Longrightarrow} p'\} \cup \{\bot \mid p \uparrow\} \\
\mathrm{obs}(o_1 \wedge o_2, p) &= \mathrm{obs}(o_1, p) \wedge \mathrm{obs}(o_2, p) \\
\mathrm{obs}(o_1 \vee o_2, p) &= \mathrm{obs}(o_1, p) \vee \mathrm{obs}(o_2, p) \\
\mathrm{obs}(\forall\, o, p) &= \forall\, \mathrm{obs}(o, p) \\
\mathrm{obs}(\exists\, o, p) &= \exists\, \mathrm{obs}(o, p)
\end{aligned}
$$

□

The function from Definition 6.5 follows the syntax of test expressions faithfully, so most cases should need no further explanation. We note that tests of form (6.3) are allowed to continue only if the action a is available to, and is performed by the process under test; if the respective action is not available, the test fails. In contrast, when a test of form (6.4) is applied to some process, we record a success whenever the process refuses the action (the primary purpose of such a test), but then we go ahead and allow the action to be performed anyway, to see what happens next (i.e., we remove the block on the action; maybe in addition to the noted success we get a failure later). As we shall see in Section 6.7 such a behavior of allowing the action to be performed after a refusal is of great help in identifying crooked coffee machines (and also in differentiating between processes that would otherwise appear equivalent).

As a final thought, we note again that tests can be in fact expressed in the same syntax as the one used for processes. A test then moves forward synchronized with the process under investigation, in the sense that the visible action performed by the process should always be the same as the action performed by the test. This synchronized run is typically denoted by the operator |, and the result is itself a process. We thus obtain an operational formulation of tests, which is used as well [Abr87, Phi87] and is quite

intuitive. Since we find the previous version more convenient for this presentation, we do not insist on it and direct instead the reader elsewhere [Abr87] for details.

6.2.3 Equivalence and Preorder Relations

The semantics of tests presented in the previous section associates a set of outcomes for each pair test–process. By comparing these outcomes (i.e., the set of possible observations one can make while interacting with two processes, or the observable behavior of the processes) we can define the *observable testing preorder*[3] $\sqsubseteq$. Given the preorder one can easily define the *observable testing equivalence* $\simeq$.

Definition 6.6. The **observable testing preorder** is a relation $\sqsubseteq \subseteq \mathbf{Q} \times \mathbf{Q}$, where $p \sqsubseteq q$ iff $\text{obs}(o, p) \subseteq \text{obs}(o, q)$ for any test $o \in \mathcal{O}$. The observable testing equivalence is a relation $\simeq \subseteq \mathbf{Q} \times \mathbf{Q}$, with $p \simeq q$ iff $p \sqsubseteq q$ and $q \sqsubseteq p$. □

If we restrict the definition of $\mathcal{O}$ (and thus the definition of the function obs), we obtain a different preorder, and thus a different equivalence. In other words, if we change the set of possible tests that can be applied to processes (the **testing scenario**), then we obtain a different classification of processes.

We will present in what follows various preorder relations under various testing scenarios. These preorders correspond to sets of changes imposed on $\mathcal{O}$ and obs, and we shall keep comparing various scenarios with the testing scenario presented in Section 6.2.2. As it turns out, the changes we impose on $\mathcal{O}$ are in all but one case restrictions (i.e., simplification of the possible tests).

We will in most cases present an equivalent modal characterization corresponding to these restrictions. Such a modal characterization (containing a set of testing formulae and a satisfaction operator) will in essence model exactly the same thing, but we are able to offer some results that are best shown using the modal characterization rather than other techniques.

When we say that a preorder $\sqsubseteq_\alpha$ makes more distinction than another preorder $\sqsubseteq_\beta$ we mean that there exist processes that are distinguishable under $\sqsubseteq_\alpha$ but not under $\sqsubseteq_\beta$. This does not imply that $\sqsubseteq_\alpha$ and $\sqsubseteq_\beta$ are comparable, i.e., it could be possible that $\sqsubseteq_\alpha$ makes more distinction than $\sqsubseteq_\beta$ *and* that $\sqsubseteq_\beta$ makes more distinction than $\sqsubseteq_\alpha$. Whenever $\sqsubseteq_\alpha$ makes more distinction than $\sqsubseteq_\beta$ but not the other way around we say that $\sqsubseteq_\alpha$ is *coarser* than $\sqsubseteq_\beta$, or that $\sqsubseteq_\beta$ is *finer* than $\sqsubseteq_\alpha$.

6.3 Trace Preorders

We thus begin our discussion on preorder and equivalence relations with what we believe to be the simplest assumption: we compare two processes by their *trace*, i.e., by the sequence of actions they perform. In this section we follow roughly [vG01, dN87].

We consider that the divergence predicate $\uparrow_B$ of the underlying transition system is empty (no process diverges). The need for such a strong assumption will become clear later, when we discover that trace preorders do not cope well with divergence.

[3] Recall that this was originally named testing preorder [Abr87], but we introduce the new name because of name clashes that developed over time.

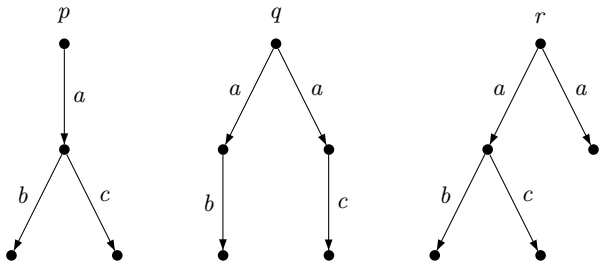

Fig. 6.3. Three sample processes ($p \simeq_{CT} q \simeq_{CT} r$; $q \not\sqsubseteq_B p$).

The trace preorder is based on the following *testing scenario*: We view a process as a black box that contains only one interface to the real world. This interface is a window displaying at any given moment the action that is currently carried out by the process. The process chooses its execution path autonomously, according to the given transition system. As soon as no action is carried out, the display becomes empty. The observer records a sequence of actions (a trace), or a sequence of actions followed by an empty window (a complete trace). Internal moves are ignored (indeed, by their definition they are not observable). We regard two processes as equivalent if we observe the same complete trace using our construction for both processes.

Specifically, $\sigma \in \mathsf{Act}^*$ is a **trace** of a process p iff there exists a process q such that $p \stackrel{\sigma}{\Longrightarrow} q$. A **complete trace** $\sigma \in \mathsf{Act}^*$ is a trace such that $p \stackrel{\sigma}{\Longrightarrow} q$ and $q \not\Longrightarrow$.

The set $\mathcal{L}_{CT}$ of **complete trace formulae** is inductively defined as follows:

- $\top \in \mathcal{L}_{CT}$ ($\top$ marks the end of a trace);
- $0 \in \mathcal{L}_{CT}$ (0 marks the end of a complete trace);
- if $\psi \in \mathcal{L}_{CT}$ and $a \in \mathsf{Act}$ then $a\psi \in \mathcal{L}_{CT}$.

A modal characterization for trace formulae is given by the **satisfaction operator** $\models \subseteq \mathbf{Q} \times \mathcal{L}_{CT}$ inductively defined by:

- $p \models \top$ for all $p \in \mathbf{Q}$;
- $p \models 0$ if $p \not\Longrightarrow$;
- $p \models a\psi$ if $p \stackrel{a}{\Longrightarrow} q$ and $q \models \psi$ for some $q \in \mathbf{Q}$.

We can now define the **complete trace preorder** $\sqsubseteq_{CT}$ and implicitly the complete trace equivalence $\simeq_{CT}$:

Definition 6.7. $p \sqsubseteq_{CT} q$ iff $p \models \psi$ implies $q \models \psi$ for any $\psi \in \mathcal{L}_{CT}$. □

The complete trace preorder induces the equivalence used in the theory of automata and languages. Indeed, consider the processes as language generators and then the trace preorder is given by the inclusion of the language of complete traces generated by one process into the language of complete traces generated by the other process. Take for instance the processes shown in Figure 6.3. We notice that $p \simeq_{CT} q$ since they both generate the language $\{\top, a\top, ab0, ac0\}$, and that $q \sqsubseteq_{CT} r$ (since r generates the larger language $\{\top, a\top, ab0, ac0, a0\}$).

We note in passing that an even weaker (in the sense of making less distinction) preorder relation can be defined [vG01] by eliminating the distinction between traces and complete traces (by putting $\top$ whenever we put 0). Under such a preorder (called **trace preorder**), the three processes in Figure 6.3 are all equivalent, generating the language $\{\top, a\top, ab\top, ac\top\}$. (We note however that the complete trace preorder is quite limited so we do not find necessary to further elaborate on an even weaker preorder.)

For one thing, trace preorder (complete or not) does not deal very well with diverging processes. Indeed, we need quite some patience in order to determine whether a state diverges or not; no matter how long we wait for the action to change in our display window, we cannot be sure that we have a diverging process or that we did not reach the end of an otherwise finite sequence of internal moves. We also have the problem of infinite traces. This is easily fixed in the same language theoretic spirit that does not preclude an automaton to generate infinite words, but then we should arm ourselves with the same immense amount of patience. Trace preorders imply the necessity of infinite observations, which are obviously impractical.

Despite all these inconveniences, trace preorders are the most elementary preorders, and perhaps the most intuitive (that's why we chose to start our presentation with them). In addition, such preorders seem to capture the finest differences in behavior one would probably like to distinguish (namely, the difference between observable sequences of actions). Surprisingly, it turns out that other preorders make an even greater distinction. Such a preorder is the subject of the next section.

6.4 Observation Preorders and Bisimulation

As opposed to the complete trace preorder that *seems* to capture the finest observable differences in behavior, the **observation preorder** [Mil80, HM80], the subject of this section, *is* the finest behavioral preorder one would want to impose; i.e., it incorporates all distinctions that could reasonably be made by external observation. The additional discriminating power is the ability to take into account not only the sequences of actions, but also some of the intermediate states the system goes through while performing the respective sequence of actions. Indeed, differences between intermediate states can be exploited to produce different behaviors.

It has also been argued that observation equivalence makes too fine a distinction, even between behaviors that cannot be really differentiated by an observer. Such an argument turns out to be pertinent, but we shall postpone such a discussion until we introduce other preorder relations and have thus something to compare.

The **observation preorder** $\sqsubseteq_B$ is defined using a family of preorder relations $\sqsubseteq_n$, $n \geq 0$ [Abr87]:

(1) it is always the case that $p \sqsubseteq_0 q$;
(2) $p \sqsubseteq_{n+1} q$ iff, for any $a \in \mathsf{Act}$ it holds that
   - for any p' such that $p \stackrel{a}{\Longrightarrow} p'$ there exists q' such that $q \stackrel{a}{\Longrightarrow} q'$ and $p' \sqsubseteq_n q'$, and
   - if $p \downarrow$ then (*i*) $q \downarrow$ and (*ii*) for any q' such that $q \stackrel{a}{\Longrightarrow} q'$ there exists p' such that $p \stackrel{a}{\Longrightarrow} p'$ and $p' \sqsubseteq_n q'$;

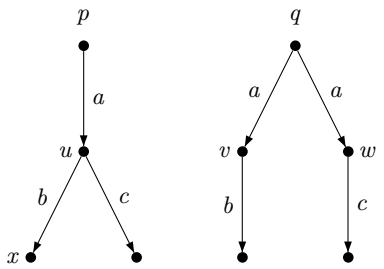

Fig. 6.4. Processes not equivalent under observation preorder ($p \not\simeq_B q$; $p \simeq_{CT} q$; $p \simeq_R q$).

(3) $p \sqsubseteq_B q$ iff for any $n \geq 0$ it holds that $p \sqsubseteq_n q$.

The equivalence $\simeq_B$ induced by $\sqsubseteq_B$ ($p \simeq_B q$ iff $p \sqsubseteq_B q$ and $q \sqsubseteq_B p$) is called *observation equivalence*.

The observation equivalence is often called **(weak) bisimulation** equivalence, hence the B subscript (the other logical–and often used–subscript O having the inconvenience of being easily confused with a zero).

It is clear that the observation preorder makes more distinction than trace preorders. Consider the processes p and q from Figure 6.3, shown again in Figure 6.4 this time with names for some of the extra states. It is immediate that $v \sqsubseteq_1 u$, and that $w \sqsubseteq_1 u$. It follows that $q \sqsubseteq_2 p$. However, it is *not* the case that $u \sqsubseteq_1 v$, and thus $q \not\sqsubseteq_2 p$. We have a strict implementation relation between q and p. Recall however that these two processes are equivalent under trace preorders.

Observation preorder corresponds to a *testing scenario* identical with the general scenario presented in Definitions 6.4 and 6.5 (in Section 6.2.2). As is the case with trace preorder we can *inspect* the sequence of *actions* performed by the process under scrutiny. This is given by expressions of form (6.1), (6.2), and (6.3).

As a side note, we mentioned at the beginning of this section that observation preorder makes more distinction than trace preorder. The expressions we allow up to this point are enough to show this: Then the tests only have the form $a_1 a_2 \ldots a_n$Succ or $a_1 a_2 \ldots a_n$Fail for some $n \geq 0$. This way we can actually distinguish between processes such as p and q from Figure 6.4. Indeed, we notice that

$$\text{obs}(ab\text{Succ}, p) = \{\top\}$$

whereas

$$\text{obs}(ab\text{Succ}, q) = \{\top, \bot\}$$

(we can start on the ac branch of q, which will produce $\bot$). In other words, we are able to distinguish between distinct paths in the run of a process, not only between different sequences of actions.

We close the side remark and go on with the description of the testing scenario for observation preorder. The addition of expressions of form (6.4) introduces the concept of *refusals*, which allow one to obtain information about the failure of the process to

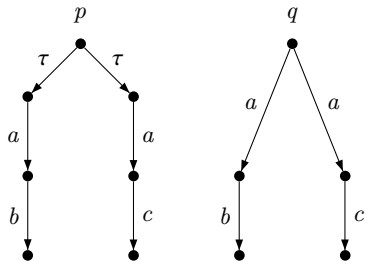

Fig. 6.5. More processes not equivalent under observation preorder ($p \not\simeq_B q$; $p \simeq_{CT} q$; $p \simeq_{\text{must}} q$; $p \simeq_R q$).

perform some action (as opposed to its ability to perform something). The expressions of form (6.6) and (6.7) allows us to *copy* the process being tested at any time during its execution, and to further test the copies by performing separate tests. *Global testing* is possible given expressions of form (6.8) and (6.9). This is a generalization of the two copy operations, in the sense that information is gathered independently for each possible test, and the results are then combined together. Finally, *nondeterminism* is introduced in the tests themselves by Expression (6.5). Such a nondeterminism is however controlled by the process being tested; indeed, if the process is convergent then we will eventually perform test o from an εo construction. By this mechanism we allow the process to "stabilize" before doing more testing on it.

Proposition 6.8. *With the set O of tests as defined in the above testing scenario, $p \sqsubseteq_B q$ iff* $\text{obs}(o, p) \subseteq \text{obs}(o, q)$ *for any test $o \in O$.*

In other words, observation preorder and observable testing preorder are the same, i.e., observation equivalence corresponds exactly to indistinguishability under testing.

A modal characterization of observation equivalence can be given in terms of the set $\mathcal{L}_{HM}$ of *Hennessy-Milner formulae*:

- $\top, \bot \in \mathcal{L}_{HM}$;
- if $\phi, \psi \in \mathcal{L}_{HM}$ then $\phi \wedge \psi, \phi \vee \psi, [a]\psi, \langle a\rangle\phi \in \mathcal{L}_{HM}$ for some $a \in \mathsf{Act}$.

The satisfaction operator $\models \in \mathbf{Q} \times \mathcal{L}_{HM}$ is defined in the following manner:

- $p \models \top$ is true;
- $p \models \bot$ is false;
- $p \models \phi \wedge \psi$ iff $p \models \phi$ and $p \models \psi$;
- $p \models \phi \vee \psi$ iff $p \models \phi$ or $p \models \psi$;
- $p \models [a]\phi$ iff $p \downarrow$ and for any p' such that $p \stackrel{a}{\Longrightarrow} p'$ it holds that $p' \models \phi$;
- $p \models \langle a\rangle\phi$ iff there exists p' such that $p \stackrel{a}{\Longrightarrow} p'$ and $p' \models \phi$.

The following is then the modal characterization of the observation equivalence [Abr87]:

Proposition 6.9. *In an underlying sort-finite derived transition system, $p \sqsubseteq_B q$ iff $p \models \psi$ implies $q \models \psi$ for any $\psi \in \mathcal{L}_{HM}$.*

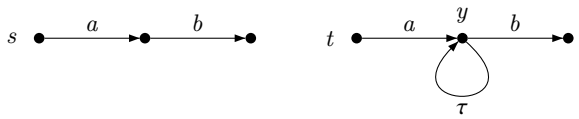

Fig. 6.6. Processes equivalent under observation preorder ($s \simeq_B t$; $s \simeq_R t$; $s \not\simeq_{\text{must}} t$; $s \simeq_{\text{fmust}} t$).

The translation between expressions in $\mathcal{L}_{HM}$ and tests is performed by the function $(\cdot)^* : \mathcal{L}_{HM} \to \mathcal{O}$ defined as follows [Abr87]:

$$\begin{array}{ll} (\top)^* = \textsc{Succ} & (\bot)^* = \textsc{Fail} \\ (\psi \wedge \phi)^* = (\psi)^* \wedge (\phi)^* & (\psi \vee \phi)^* = (\psi)^* \vee (\phi)^* \\ ([a]\psi)^* = \forall\, \widetilde{a}(\psi)^* & (\langle a \rangle \psi)^* = \exists\, a(\psi)^* \\ ([\varepsilon]\psi)^* = \forall\, \varepsilon(\psi)^* & (\langle \varepsilon \rangle \psi)^* = \exists\, \varepsilon(\psi)^* \end{array} \tag{6.10}$$

Essentially all the testing techniques from the general testing scenario are combined together in a rather comprehensive set of testing techniques to create observation preorder. The comprehensiveness of the testing scenario itself is a problem. While it has an elegant proof theory (which is not presented here, the interested reader is directed elsewhere [Abr87]), observation preorder induces a too complex testing scenario. We have constructed indeed a very strong notion of observability; most evidently, according to Expressions (6.8) and (6.9) we assume the ability to enumerate all possible operating environments, so as to guarantee that all the nondeterministic branches of the process are inspected. The number of such branches is potentially infinite. It is not believed that global testing is really acceptable from a practical point of view. Preorder relations that will be presented in what follows place restrictions in what we can observe, and thus have a greater practical potential.

It is also the case that observation preorder makes too much of a distinction between processes. One example of distinction not made in trace preorder has been given in Figure 6.4. One can argue that such a distinction may make sense in some cases, but such an argument is more difficult in the case of processes shown in Figure 6.5, which are slight modifications of the processes from Figure 6.4. Under (any) trace preorder the two processes p and q are equivalent, and we argue that this makes sense; for indeed by the very definition of internal moves they are not manifest to the outside world, and besides internal moves the two processes behave similarly. However, it is not the case that $q \simeq_B p$. Indeed, notice that q **ref** b, whereas it is not the case that p **ref** b (since p can move ahead by means of internal actions, and thus the refusal does not take place according to Definition 6.3). Then the test $\widetilde{b}\textsc{Succ}$ introduces a $\top$ outcome in q but not in p according to Definition 6.5; the non-equivalence follows. This certainly looks like nitpicking; we shall introduce below preorders that are not that sensitive to internal moves.

We observe on the other hand that the processes s and t from Figure 6.6 are equivalent under observation preorder. We saw observation preorder giving too much weight to internal moves; now we see the same preorder ignoring this kind of moves altogether. The reason for this is that the internal move never changes the state, so no matter how many times we go through it we end where we left from. Still, the τ-loop is not without

significance in practice since such a loop may produce divergence (if the process keeps iterating through it). However, it can also be argued that the τ-loop is executed an arbitrary but finite number of times and so the process executes b eventually (under some notion of fairness). We shall actually argue back and forth about these two processes as we go along with the description of other preorder relations, so you do not have to make up your mind just yet.

6.5 Testing Preorders

Testing preorders [dNH84] are coarser than observation preorder. Essentially, testing preorders differentiate between processes based on differences in deadlock behavior. We may differentiate by the ability to respond positively to a test, or the ability to respond negatively to a test, or both. In practical cases this is often sufficient.

Recall the concept of outcome of a test presented in Section 6.2.2. For a test o and a process p the result of applying o to p is the set of runs $\textsc{Runs}(o, p)$ with outcomes from the set $\{\bot, \top\}$. Also recall the lattices $\mathbb{P}_{\text{may}}$, $\mathbb{P}_{\text{must}}$, and $\mathbb{P}_{\text{conv}}$ over the powerset of $\{\bot, \top\}$, together with the corresponding partial order relations.

We then have the following *testing scenario* for testing preorders: We run a test in parallel to the process being tested, such that they perform the same actions. If the test reaches a success state, then the test succeeds; if on the other hand the process reaches a deadlock state (i.e., a state with no way out), or if the process diverges before the test has reached a success state, the test fails. Sometimes we are interested in running the same test repeatedly and collect all of the possible outcomes; we need this when we want to make sure that a test succeeds no matter what.

Formally, we change in what follows (simplify in fact) the semantics of Expression (6.3) from Definition 6.4 on page 140 to

$$\text{obs}(ao, p) = \bigcup \{\text{obs}(o, p') \mid p \stackrel{a}{\Longrightarrow} p'\} \cup \{\bot \mid p \uparrow\} \cup \{\bot \mid p \Longrightarrow\!\!\!\!\!\!\not\quad\} \tag{6.11}$$

Then we look at two alternative ways to restrict the set of tests O:

(1) Let O_{may} be defined only by expressions of form (6.1), (6.3), and (6.5). We do not need any test that signifies failure; instead, failure under test happens whenever we reach a deadlock, according to Expression (6.11). Indeed, we are not allowed to combine different testing outcomes at all (there are no boolean operators such as $\wedge$, $\vee$ on outcomes), so a test that fails does not differentiate between anything (it fails no matter what); therefore these tests are excluded as useless. According to the same Expression (6.11) we do not differentiate between deadlock and divergence—both constitute failure under test.

Incidentally, the inability to combine test outcomes makes sense in practice; for indeed recall our criticism with respect to the "global testing" allowed in the observation preorder and that we considered impractical. As it turns out it may also be a too strong restriction, so we end up introducing it again in our next set of tests.

(2) We are now interested in all the possible outcomes of a test. First, let O_{must} be defined only by expressions of form (6.1), (6.2), (6.3), and (6.5). This time we do like to combine tests, but only by taking the union of the outcomes without

combining them in any smarter way. This is the place where we deviate from (i.e., enhance) our generic testing scenario, and we add the following expression to our initial set of tests O:

$$o = o_1 + o_2 \tag{6.12}$$

with the semantics

$$\text{obs}(o_1 + o_2, p) = \text{obs}(o_1, p) \cup \text{obs}(o_2, p)$$

(3) A combination between these two testing scenarios is certainly possible, so put $O = O_{\text{may}} \cup O_{\text{must}}$.

In order to complete the test scenario, we define the following relations between processes and tests:

Definition 6.10. Process p may satisfy test o, written p **may** o iff $\top \in \text{obs}(o, p)$. Process p must satisfy test o, written p **must** o iff $\{\top\} = \text{obs}(o, p)$. □

The two relations introduced in Definition 6.10 correspond to the lattices $\mathbb{P}_{\text{may}}$ and $\mathbb{P}_{\text{must}}$, respectively. When we use the **may** relation we are happy with our process if it does not fail every time; if we have a successful run of the test, then the test overall is considered successful. Relation **must** on the other hand considers failure catastrophic; here we accept no failure, all the runs of the test have to be successful for a test to be considered a success. An intuitive comparison with the area of sequential programs is that the **may** relation corresponds to partial correctness, and the **must** relation to total correctness. We have one lattice left, namely $\mathbb{P}_{\text{conv}}$; this obviously corresponds to the conjunction of the two relations.

Based on this testing scenario, and according to our discussion on the relations **may** and **must** we can now introduce three testing preorders[4] $\sqsubseteq_{\text{may}}, \sqsubseteq_{\text{must}}, \sqsubseteq_{\text{conv}} \subseteq \mathbf{Q} \times \mathbf{Q}$:

(1) $p \sqsubseteq_{\text{may}} q$ if for any $o \in O_{\text{may}}$, p **may** o implies that q **may** o.
(2) $p \sqsubseteq_{\text{must}} q$ if for any $o \in O_{\text{must}}$, p **must** o implies that q **must** o.
(3) $p \sqsubseteq_{\text{conv}} q$ if $p \sqsubseteq_{\text{may}} q$ and $p \sqsubseteq_{\text{must}} q$.

The equivalence relations corresponding to the three preorders are denoted by $\simeq_{\text{may}}$, $\simeq_{\text{must}}$, and $\simeq_{\text{conv}}$, respectively. We shall use $\sqsubseteq_T$ (for "testing preorder") instead of $\sqsubseteq_{\text{conv}}$ in subsequent sections.

Note that the relation $\sqsubseteq_{\text{conv}}$ is implicitly defined in terms of observers from the set $O = O_{\text{may}} \cup O_{\text{must}}$. Also note that actually we do not need three sets of observers, since all the three preorders make sense under O. The reason for introducing these three distinct sets is solely for the benefit of having different testing scenarios for the three testing preorders (that are also tight, i.e., they contain the smallest set of observers possible), according to our ways of presenting things (in which the testing scenario defines the preorder).

The most discerning relation is of course $\sqsubseteq_{\text{conv}}$. It is also the case that in order to see whether two processes are in the relation $\sqsubseteq_{\text{conv}}$ we have to check both the other

[4] These preorders were given numerical names originally [dNH84]. We choose here to give names similar to the lattices they come from in order to help the intuition.

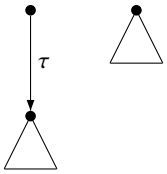

Fig. 6.7. Processes equivalent under $\simeq_{\text{may}}$.

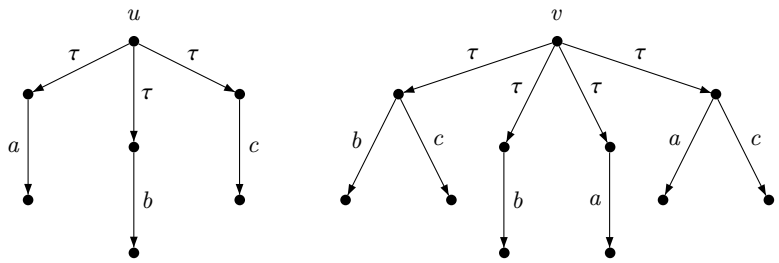

Fig. 6.8. Two processes not equivalent under testing preorder ($u \not\simeq_{\text{must}} v$; $u \simeq_{CT} v$).

relations, so our subsequent discussion will deal mostly the other two preorders (since the properties of $\sqsubseteq_{\text{conv}}$ will follow immediately).

One may wonder what we get out of testing preorders in terms of practical considerations. First, as opposed to trace preorders, we no longer need to record the whole trace of a process; instead we only distinguish between success and failure of tests. It is also the case that we do not need to combine all the outcomes of test runs as in observation preorder. We still have a notion of "global testing," but the combination of the outcomes is either forbidden (in $\sqsubseteq_{\text{may}}$) or simplified. In all, we arguably get a preorder that is more practical. We also note that, by contrast to trace preorders *we can have finite tests (or observers) even if the processes themselves consist in infinite runs*. Indeed, in trace preorders a test succeeds only when the end of the trace is reached, whereas we can now stop our test whenever we are satisfied with the behavior observed so far (at which time we simply insert a SUCC or FAIL in our test).

In terms of discerning power, recall first the example shown in Figure 6.5 on page 146, where the two processes p and q are not equivalent under observation preorder. We argued that this is not necessarily a meaningful distinction. According to this argument testing preorders are better, since they do not differentiate between these two processes. Indeed, p and q always perform an action a followed by either an action b or an action c, depending on which branch of the process tree is taken (recall that the distinction between p and q under observation preorder was made in terms of nitpicking refusals, that are no longer present in testing preorders). We thus revert to the "good" properties of trace preorders.

Recall now our argument that the processes from Figure 6.6 on page 147 should be considered the same. We also argued the other way around, but for now we stick

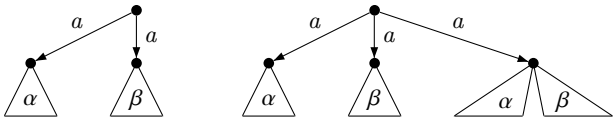

Fig. 6.9. Processes equivalent under any testing preorder.

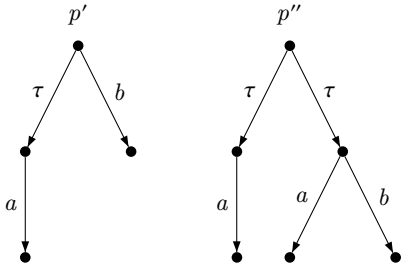

Fig. 6.10. More processes equivalent under testing preorders ($p' \simeq_T p''$; $p' \not\simeq_R p''$).

with the first argument because we also have $s \simeq_{\text{may}} t$. Indeed, it is always the case that processes such as the ones depicted in Figure 6.7 are equivalent under $\simeq_{\text{may}}$, and the equivalence of s and t follows. In other words, we keep the "good" properties of observation preorder.

In general, $\simeq_{\text{may}}$ ignores the tree structure of processes, which shows that this preorder is a very weak relation. This is not the case with $\simeq_{\text{must}}$. It is now the time to argue that the two processes depicted in Figure 6.6 should be considered different. They are so under $\simeq_{\text{must}}$, for indeed one branch of t diverges while no divergent computations are present in s. A suitable test such as abSucc will exploit this property under the **must** operator. In general, the presence of divergence in the form of an infinite path of internal moves will ruin a test under $\simeq_{\text{must}}$. Whether this is desired or not depends on one's interpretation of such an infinite path of internal moves.

Continuing with examples for $\simeq_{\text{must}}$, consider the processes shown in Figure 6.8. No matter what internal move is chosen by v, it can always perform either a or b. It follows that v **must** (aSucc + bSucc). On the other hand, at its point of choosing which way to go, u has the choice of performing c. It thus follow that u **may** (aSucc + bSucc), but it is not the case that u **must** (aSucc + bSucc). In general, it is easy to see that $u \simeq_{\text{may}} v$, but that $u \not\simeq_{\text{must}} v$. Incidentally, these processes are equivalent in trace preorders.

We should emphasize that, though $\simeq_{\text{must}}$ takes into consideration the tree structure of the process under scrutiny, it does so in a more limited way. This was shown in our discussion based on Figure 6.5. More generally, the processes depicted in Figure 6.9 are equivalent under any testing preorder.

Finally, an example that will come in handy when we compare testing preorders with refusal preorders (that is the subject of the next section) is given by the two processes shown in Figure 6.10, which are equivalent under $\sqsubseteq_{\text{conv}}$.

All of the examples presented here allow us to conclude the following: The preorder

$\sqsubseteq_{\text{may}}$ is a very weak relation, but has the advantage of needing no global testing. The other testing preorders do make use of global testing, but in a restricted way compared with observation preorder. The distinctions they make are not as rich as in the case of observation preorder, but they are nonetheless quite rich. On the principle that the most distinction we can make between processes the better we are, one now wonders whether we can do better in distinctions without the complexity of observation preorder.

Since $\sqsubseteq_{\text{conv}}$ is clearly the testing preorder that makes the most distinctions, we shall henceforth understand this preorder when we refer simply to testing preorder. Recall that we also decided to denote it by $\sqsubseteq_T$ in subsequent sections (with $\simeq_T$ as the name of the induced equivalence).

6.6 Refusal Testing

The only reasonable way in which one can obtain information about a process is by communicating with it by means of actions. This is precisely what we modeled in all this chapter. For example, we just inspect the actions performed by a process in trace preorders; we then take it one step further in the testing preorder, where we request sequences of actions that depend on the information gained about the process as the test progresses. In our generic testing scenario presented in Section 6.2.2 we go even further by adding to tests the ability of *refusing* actions. This is an interesting feature, that looks powerful and arguably practically feasible. Recall on the other hand that we definitely did not see observation preorder (the only preorder involving the concept of refusals) as practical, at least not as practical as testing preorders.

So on one hand we have refusals, that look promising (and practical enough), and on the other hand we have testing preorders, that look practical. We now combine them. While we are at it, we also differentiate between failure by deadlock (no outgoing actions) and divergence. We thus obtain the **refusal preorders** [Phi87].

Refusal preorders rely on the following *testing scenario*: We start from the scenario of complete trace semantics, i.e., we view a process as a black box with a window that displays the current action and becomes empty when a deadlock occurs. We now equip our box with one switch for each possible action $a \in \mathsf{Act}$. By flipping the switch for some action a to "on" we *block* a; the process continues to choose its execution path autonomously, but it may only start by executing actions that are not blocked by our manipulation of switches. The configuration of switches can be changed at any time during the execution of the process.

Formally, we restrict our set of tests O introduced in Definition 6.4 on page 140 by allowing only expressions of form (6.1)–(6.5), and a restricted variant of (6.12 on page 149) as follows:

$$o = ao_1 + \widetilde{a}o_2 \tag{6.13}$$

The semantics of this kind of expressions is immediately obtained by the semantics of Expressions (6.12) and (6.4) (since we are starting here from the scenario of the testing preorder, the semantics of tests of form (6.4) is given by Expression (6.11)). This is our "switch" that we flip to blocks a (and then we follow with o_2) or not.

We also differentiate between deadlock and divergence. We did not make such a differentiation in the development of previous preorders, because we could not do this readily (and in those cases when we could, we would simply express this in terms of the divergence predicate). However, now that we talk about refusals we will need to distinguish between tests that fail because of divergent processes, and tests that fail because all the actions are blocked. We find it convenient to do this explicitly, so we enrich our set of test outcomes to $\{\top, 0, \bot\}$, with $\bot$ now signifying only divergence, while 0 stands for deadlock. In order to do this, we alter the semantics of expressions of form (6.2), (6.3), and (6.4) to

$$\begin{aligned}
\text{obs}(\textsc{Fail}, p) &= \{0\} \\
\text{obs}(ao, p) &= \bigcup\{\text{obs}(o, p') \mid p \xrightarrow{a} p'\} \cup \{\bot \mid p \uparrow\} \cup \{0 \mid p \xrightarrow{a}\!\!\!\!/\ \} \\
\text{obs}(\widetilde{a}o, p) &= \bigcup\{\text{obs}(o, p) \mid p \xrightarrow{a}\!\!\!\!/\ , p \xrightarrow{\tau}\!\!\!\!/\ \} \cup \{\bot \mid p \uparrow\} \cup \{0 \mid p \xrightarrow{a} \text{ or } p \xrightarrow{\tau}\ \}
\end{aligned}$$

Note that in the general testing scenario we count a failure whenever we learn about a refusal. In this scenario, a refusal generates a failure only when no other action can be performed. Also note that this scenario imposes further restrictions on the applicable tests by restricting the semantics of the allowable test expressions. As a further restriction, we have the convention that test expressions of form (6.5) shall be applied with the highest priority of all the expressions (i.e., internal actions are performed before anything else, such that the system is allowed to fully stabilize itself before further testing is attempted—this is also the reason for replacing relation $\Longrightarrow$ with the stronger $\rightarrow$ in the semantics of the tests ao and $\widetilde{a}o$).

It should be mentioned that the original presentation of refusal testing [Phi87] allows initially to refuse sets of actions, not only individual actions. In this setting we can flip sets of switches as opposed to one switch at a time as we allow by the above definition of O. However, it is shown later in the same paper [Phi87] that refusing sets of actions is not necessary, hence our construction. Now that the purpose of our test scenario is clear, we shall further restrict the scenario. Apparently this restriction is less expressive, but the discussion we mentioned above [Phi87] shows that—against intuition—we do not lose anything; although the language is smaller, it is equally expressive. In the same spirit as for testing preorders, we restrict our set of tests in two ways, and then we introduce a new version of the operators **may** and **must**.

(1) Let the set O_1 contain exactly all the expressions of form (6.1) and a restricted version of form (6.13) where either $o_1 = \textsc{Fail}$ or $o_2 = \textsc{Fail}$.
Let then p **may** o iff $\top \in \text{obs}(p, o)$.
(2) Let the set O_2 contain exactly all the expressions of form (6.2) and a restricted version of form (6.13) where either $o_1 = \textsc{Succ}$ or $o_2 = \textsc{Succ}$.
Let then p **must** o iff $\{\top\} = \text{obs}(p, o)$.
(3) As usual, put $O = O_1 \cup O_2$.

In other words, at any given time we either block an action and succeed or fail (as the case may be), or we follow the action we would have blocked otherwise and move forward; no other test involving blocked actions is possible. One may wonder about the cause of the disappearance of form (6.5). Well, this expression was not that "real"

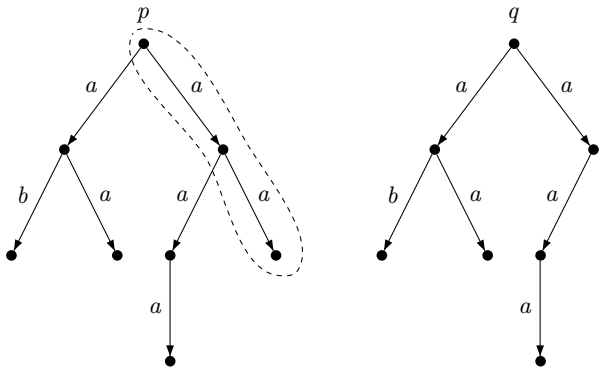

Fig. 6.11. Processes not equivalent under refusal preorder ($p \not\simeq_R q$; $p \simeq_T q$).

to begin with (we never wrote ε down in our test expressions, we provided it instead to allow the process to "stabilize" itself), and we can now replace the expression εo by $e\textsc{Fail} + \widetilde{e}o$, where e is a new action we invent outside Act (thus knowing that the process will never perform it).

With these helper operators and sets of tests we now define the **refusal preorder** $\sqsubseteq_R$ as follows: $p \sqsubseteq_R q$ iff (a) p **may** o implies q **may** o for any $o \in O_1$, and (b) p **must** o implies q **must** o for any $o \in O_2$. The induced refusal equivalence $\simeq_R$ is defined in the usual way.

The alert reader has noticed that the refusal preorder is by far the most restricted preorder we have seen. Let us now take a look at its power of discrimination. Since it has been shown that the generic refusal testing scenario (that we started with) and our restricted variant are in fact equally expressive, we shall feel free to use either of them as it suits our needs.

We now compare refusal preorder with the testing preorder. First, it is immediate that processes depicted in Figures 6.4 on page 145, 6.5 on page 146, and 6.9 on page 151 continue to be equivalent under refusal preorders.

On the other hand, consider the processes shown in Figure 6.10 on page 151 which are equivalent under testing preorder. We then notice that under refusal preorder we have $\text{obs}(b\textsc{Succ}, p') = \{0\}$, for indeed the internal action is performed first to stabilize the process, and after this no b action is possible. However, it is immediate that $\text{obs}(b\textsc{Succ}, p'') = \{\top, 0\}$. We do not even use refusals here, the two processes become non-equivalent because our convention that test expressions of form (6.5) shall always be performed first.

Even in the absence of such a convention we have a more precise preorder. Consider for instance the processes from Figure 6.11. They are immediately equivalent under testing preorder, but not so under refusal preorder. Indeed, it holds that $\text{obs}(a\widetilde{b}a\widetilde{a}\textsc{Succ}, p) = \{\top, 0\}$ and $\text{obs}(a\widetilde{b}a\widetilde{a}\textsc{Succ}, q) = \{0\}$ (the path circled in the figure is the only successful path under this test).

It is then apparent that refusal preorder makes more distinction than the testing preorder. We shall tackle the reverse comparison by giving a precise comparison of refusal

preorder with the observation preorder. Such a comparison is possible by developing a modal characterization for the refusal preorder. As it turns out, this characterization can also be given in terms of a subset of $\mathcal{L}_{HM}$ (which is the set of formulae corresponding to observation preorder). This subset (denote it by $\mathcal{L}_R$) is the domain of the following partial function $(\cdot)^* : \mathcal{L}_{HM} \to O$ translating between expressions in $\mathcal{L}_{HM}$ and tests and given by:

$$\begin{array}{ll} (\top)^* = \textsc{Succ} & (\bot)^* = \textsc{Fail} \\ ([a]\psi)^* = a(\psi)^* & ([a]\psi)^* = \underline{\widetilde{a}}(\psi)^* \\ (\langle\varepsilon\rangle([a]\bot \wedge [\varepsilon]\psi))^* = \widetilde{a}(\psi)^* & ([\varepsilon](\langle a\rangle\top \vee \langle\varepsilon\rangle\psi))^* = \underline{a}(\psi)^* \end{array} \tag{6.14}$$

For succinctness we abbreviated $ao + \widetilde{a}\textsc{Fail}$ by ao, $a\textsc{Fail} + \widetilde{a}o$ by $\widetilde{a}o$, $ao + \widetilde{a}\textsc{Succ}$ by $\underline{a}o$, and $a\textsc{Succ} + \widetilde{a}o$ by $\underline{\widetilde{a}}o$. We have [Phi87]:

Proposition 6.11. *For any process $p \in \mathbf{Q}$ and for any expression $\psi \in \mathcal{L}_R$, it holds that $p \models \psi$ iff p **may** $(\psi)^*$, and that $p \models \psi$ iff p **must** $(\psi)^*$. It then follows that $p \sqsubseteq_R q$ iff $p \models \psi$ implies $q \models \psi$ for any expression $\psi \in \mathcal{L}_R$.*

It then follows that:

Theorem 6.12. *For any two processes p and q, $p \sqsubseteq_B q$ implies $p \sqsubseteq_R q$, but not the other way around.*

Proof. The implication is immediate from Proposition 6.11 given that $\mathcal{L}_R$ is a strict subset of $\mathcal{L}_{HM}$. That observation preorder is strictly finer than refusal preorder is shown by the example depicted in Figure 6.5 on page 146. □

So we find that refusal preorder is coarser than observation preorder. This also allows us to compare refusal and testing preorders. Indeed, recall that the infinite processes shown in Figure 6.6 on page 147 are equivalent under observation preorder (and then according to Proposition 6.12 under refusal preorder). We have shown in the previous section that these processes are not equivalent under testing preorder. Given that on the other hand refusal preorder distinguishes between processes indistinguishable in testing preorder, we have

Corollary 6.13. *The preorders $\sqsubseteq_T$ and $\sqsubseteq_R$ are not comparable.*

We note here an apparent contradiction with results given elsewhere [Phi87] that the two preorders are comparable. This contradiction turns out to be caused by the unfortunate (and incorrect) terminology used in [Phi87].

In practical terms, refusal preorder is clearly more appealing than observation preorder. Arguably, it is also more appealing than testing preorder, because of the simplicity of tests; indeed, we eliminated all nondeterminism from the tests in O_1 and O_2 (and thus in O). The only possible practical downside (of refusal preorder compared with testing preorder) is that we need the ability to block actions.

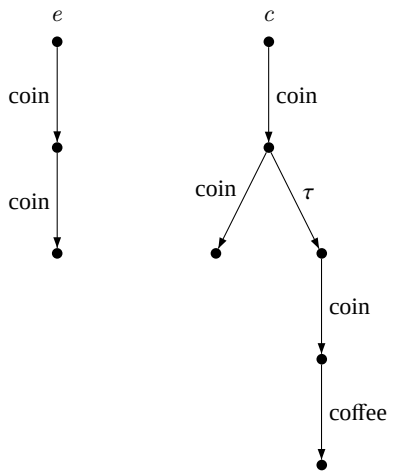

Fig. 6.12. Two vending machines ($e \sqsubseteq_R c$; $e \not\sqsubseteq_{FT} c$).

6.7 Failure Trace Testing

In refusal testing, whenever we observe a process that cannot continue because we blocked all of its possible actions we have a failed test. This seems a reasonable testing strategy, but we end up with surprising preorder relations because of it. Consider for example the rather instructive example [Lan90] of the two vending machines c and e depicted in Figure 6.12. Machine c may give us coffee if we insert two coins, while machine e eats up our money, period. In terms of refusal preorder, it is immediate that c passes strictly more tests than e, so $e \sqsubseteq_R c$. In other words, e is an implementation of c! Clearly, this contradicts most people's idea of a working coffee machine.

Such a strange concept of correct implementation is corrected by the **failure trace preorder** [Lan90]. This preorder is based on the following *testing scenario*: We have the same black box we did in the testing scenario for refusal preorder. The only difference is in our actions; when we observe the deadlock (by the empty window) we record such an occurrence (as a failure) and then we are allowed to flip switches off to allow the process to continue.

Formally, we allow exactly the same test expressions for the set O as we did initially in the previous section, but we revert the semantics of expressions of form (6.4) to its original form (continuing to make the distinction between failure as deadlock versus failure as divergence), i.e.,

$$\text{obs}(\widetilde{a}\,o, p) = \bigcup \{\text{obs}(o, p') \mid p \stackrel{a}{\Longrightarrow} p'\} \cup \{\bot \mid p \uparrow\} \cup \{0 \mid p \stackrel{\varepsilon}{\Longrightarrow} p', p' \textbf{ ref } a\}$$

We then define the operators **may** and **must** exactly as we did in the previous section, i.e., p **may** o iff $\top \in \text{obs}(p, o)$, and p **must** o iff $\{\top\} = \text{obs}(p, o)$. Finally, the **failure trace preorder** $\sqsubseteq_{FT}$ is defined as $p \sqsubseteq_{FT} q$ iff for all $o \in O$ it holds that p **may** o implies q **may** o and p **must** o implies q **must** o. As usual, the failure trace preorder induces the failure trace equivalence $\simeq_{FT}$.

Let us go back to our vending machines from Figure 6.12, and consider the test

o = coin $\widetilde{\text{coin}}$ coin coffee Succ

As opposed to refusal testing, we now have obs(o, e) = {0} (the action "coffee" is not available for the test), whereas obs(o, c) = {$\top$, 0} (we record a failure when we block action "coin" and then we move on to obtain a successful test on the right side branch). We thus notice that c **may** o but that it is not the case that e **may** o; a machine that does not give us coffee does not pass this test. Our two vending machines become thus incomparable (and justly so).

Failure trace preorder thus makes more distinction than refusal preorder. It is also easy to see that refusal preorder does not distinguish between processes that are not distinguishable under failure trace. Indeed, it is enough to place a Fail test after each action that is blocked in the tests and those tests become tests for the refusal preorder.

It is immediate to see that observation preorder is strictly finer than failure trace preorder. Indeed, we introduced on top of refusal order a semantics that is otherwise included in the semantics of observation preorder. So we have:

Proposition 6.14. *For any two processes p and q, $p \sqsubseteq_B q$ implies $p \sqsubseteq_{FT} q$ (but not the other way around), and $p \sqsubseteq_{FT} q$ implies $p \sqsubseteq_R q$ (but not the other way around).*

Using the failure trace preorder we can make distinctions that cannot be made using refusal preorder. However, this increase does not necessarily come for free. Indeed, the tests in the sets $\mathcal{O}_1$ and $\mathcal{O}_2$ described in the previous sections are *sequential*, in the sense that unions always occur between a test whose result that is immediately available (Succ or Fail) and some other, possibly longer test. In testing preorders as well as in failure trace preorder we need to copy the process while it runs; indeed, we may need to combine the outcomes of two (or more) different runs of the process, which means that we need to run two copies of the process to obtain these outcomes independently from each other. Because of the sequential tests used by refusal preorder copying is no longer necessary (but it becomes necessary once more in failure trace preorder). This being said, the definition of the **must** operator from refusal preorder implies that processes need to be copied anyway (since we have to apply many tests on them), so the failure trace testing scenario is not that bad after all.

6.8 Fair Testing

Recall the processes depicted in Figure 6.6 on page 147 and our back and forth argument that they should be considered equivalent (or not). When we considered them under the testing preorder, s and t were not equivalent, whereas they are so under the other preorders. Testing preorder, with its habit that the presence of divergence may ruin a test, will differentiate between these two processes as opposed to all the other preorders we have seen so far. As we mentioned, whether such a behavior is a good or bad thing depends on one's opinion about divergences.

For those who prefer to ignore divergences as long as there is a hope that the process will continue with its visible operation, i.e., for those who prefer to consider the processes shown in Figure 6.6 equivalent, **fair testing** is available [BRV95].

We have the same *testing scenario* for fair testing as the one used in Section 6.5, except that the operator **must** is enhanced, such that it chooses a visible action whenever such an action is available. With the same set O of observers as the one used to define the testing preorder, the new operator **fmust** is defined as follows:

> p **fmust** o iff for any $\sigma \in \mathsf{Act}^*$ and $o' \in O$ with $o = \sigma o'$, it holds that: $\mathrm{obs}(o', p') = \mathrm{obs}(o, p)$ for some $p' \in \mathbf{Q}$, $p \stackrel{\sigma}{\Longrightarrow} p'$, implies that there exists $a \in \mathsf{Act} \cup \{\textsc{Succ}\}$ such that $o' = ao''$, $o'' \in O \cup \{\varepsilon\}$.

The preorder $\sqsubseteq_{\text{fmust}}$, as well as the equivalence $\simeq_{\text{fmust}}$ induced by the operator **fmust** are defined in the usual manner.

The operator **fmust** is the "fair" variant of the operator **must** of testing preorder lineage. It ignores the divergences as long as there is a visible action (a in the above definition) accessible to the observer. The following characterization of $\sqsubseteq_{\text{fmust}}$ in terms of other preorders is easily obtained from the results presented elsewhere [BRV95]:

Proposition 6.15. *For any two processes p and q, $p \sqsubseteq_R q$ implies $p \sqsubseteq_{\text{fmust}} q$ (but not the other way around), and $p \sqsubseteq_{\text{must}} q$ implies $p \sqsubseteq_{\text{fmust}} q$ (but not the other way around).*

The modification of the testing preorder introduced by the preorder $\sqsubseteq_{\text{fmust}}$ brings us back into the generic testing scenario. In the following we go even further and tackle a problem that we did not encounter up to this point, but that is common to many preorders. This problem refers to the process of *hiding* a set of actions.

Given a transition system $B = (\mathbf{Q}, \mathsf{Act} \cup \{\tau\}, \rightarrow, \uparrow_B)$ and some set $A \subseteq \mathsf{Act}$, the result of *hiding* A is a transition system $B/A = (\mathbf{Q}, \mathsf{Act} \setminus A \cup \{\tau\}, \rightarrow_h, \uparrow_B)$, where $\rightarrow_h$ is identical to $\rightarrow$ except that all the transitions of form $p \stackrel{a}{\longrightarrow} q$ for some $a \in A$ are replaced by $p \stackrel{\tau}{\longrightarrow}_h q$.

Under a suitable transition system B, consider now the processes depicted in Figure 6.13, and the equivalent processes in $B/\{a\}$; the processes become non-equivalent under $\sqsubseteq_R$. Similar examples can be found for the other preorders presented in this chapter. These preorders are not pre-congruence relations under hiding.

A preorder based on the testing preorder and that is pre-congruent can also be introduced [BRV95]. Call such a preorder **should-testing**. The *testing scenario* is again the same as the one presented in Section 6.5, with the exception that the operators **must** and **may** are replaced by the operator **should** defined as follows (again, we have the same set O of observers as the one used to define the testing preorder):

> p **should** o iff for any $\sigma \in \mathsf{Act}^*$ and $o' \in O$ with $o = \sigma o'$, it holds that: $\mathrm{obs}(o, p) = \mathrm{obs}(o', p')$ for some $p' \in \mathbf{Q}$, $p \stackrel{\sigma}{\Longrightarrow} p'$, implies that there exists $\sigma' \in \mathsf{Act}^*$ such that $o' = \sigma' \textsc{Succ}$ and $\top \in \mathrm{obs}(o', p')$.

The preorder and the equivalence induced by the **should** operator are denoted by $\sqsubseteq_{\text{should}}$ and $\simeq_{\text{should}}$, respectively.

The idea of should-testing is that in a successful test there is always a reachable successful state, so if the choices are made fairly that state will eventually be reached. Fair testing states that a system passing the test may not deadlock unless success has

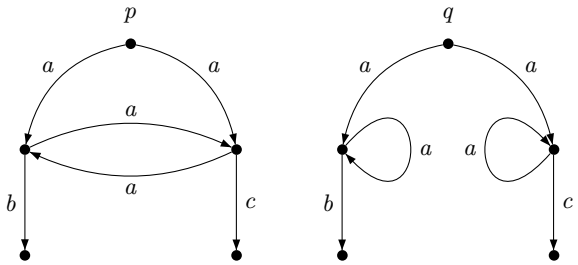

Fig. 6.13. Processes different after hiding $\{a\}$.

been reported before; should-testing requires a stronger condition in that a successful state must be reached from every state in the system.

It is immediate that $\sqsubseteq_{\text{should}}$ is coarser than $\sqsubseteq_{\text{fmust}}$ (since the success condition is stronger). This relationship is even stronger for processes with only finite visible runs:

Proposition 6.16. *For any two processes p and q, $p \sqsubseteq_{\text{should}} q$ implies $p \sqsubseteq_{\text{fmust}} q$ (but not the other way around); for any two processes p and q for which all the visible runs are finite $p \sqsubseteq_{\text{should}} q$ iff $p \sqsubseteq_{\text{fmust}} q$.*

In addition $\sqsubseteq_{\text{should}}$ is a pre-congruence under hiding—as well as under prefixing and synchronization [BRV95]; in fact we have:

Proposition 6.17. *The relation $\sqsubseteq_{\text{should}}$ is the largest relation contained in $\sqsubseteq_{\text{fmust}}$ that is a pre-congruence under synchronization and hiding.*

6.9 Conformance Testing, or Preorders at Work

This section is different from the previous ones, because it does not introduce new testing scenarios and new preorders. Instead, it puts the existing scenarios in a formalization of the concept of conformance testing [Tre94]. The description of such an environment in which preorders are put to good use is indeed a nice wrap up of our presentation.

We mentioned at least two times that preorders can be interpreted as implementation relations. In this sections we elaborate on this idea. We thus present here the *application* of everything we talked about before.

Conformance testing consists in testing the implementation of a system against that system's specification. Formally, we are given a formal specification language $\mathcal{L}_{FDT}$ (such as CCS [Mil80] or even labeled transition systems), and we have to determine for some specification $s \in \mathcal{L}_{FDT}$ what are the implementations that conform to s (i.e., are a correct implementation of s). Of course, implementations are physical objects, so we analyze their properties by means of formal models of such implementations, that are also members of $\mathcal{L}_{FDT}$. We assume that any concrete implementation can be modeled in $\mathcal{L}_{FDT}$.

There usually are more than one correct implementation of some specification, so we actually work with a set CONFORM$_s$ of implementations conforming to a specification s. This set can be defined using either a *behavior* (or model-based) *specification*, or a *requirement* (or logical) *specification*.

In the behavior specification approach the set CONFORM$_s$ is defined by means of an *implementation relation* **imp**, such that i **imp** s iff i conforms to s:

$$\text{CONFORM}_s = \{i \in \mathcal{L}_{FDT} \mid i \textbf{ imp } s\}.$$

In the requirement specification approach we define the set CONFORM$_s$ by giving all the properties that should hold for all of its elements. Such properties, or *requirements* are specified in a formal language $\mathcal{L}_{RQ}$, and if an implementation i has property r we say that i satisfies r and we write i **sat** r. A conforming implementation will have to satisfy all the properties from a set $R \subseteq \mathcal{L}_{RQ}$, so we have:

$$\text{CONFORM}_s = \{i \in \mathcal{L}_{FDT} \mid \text{ for all } r \in R, i \textbf{ sat } r\}.$$

If a suitable specification language has been chosen, we can define a *specification relation* **spec** $\subseteq \mathcal{L}_{FDT} \times \mathcal{L}_{RQ}$ which expresses the requirements that are implicitly specified by a behavior specification. Our definition for CONFORM$_s$ then becomes:

$$\text{CONFORM}_s = \{i \in \mathcal{L}_{FDT} \mid \text{ for all } r \in \mathcal{L}_{RQ}, s \textbf{ spec } r \text{ implies } i \textbf{ sat } r\}.$$

Both these approaches to the definition of CONFORM$_s$ are valid and they can be used independently from each other. They are both useful too: if we want to check an implementation against a specification the behavioral specification is appropriate; if on the other hand we want to determine conformance by testing the implementation, it is typically more convenient to derive requirements from the specification and then test them.

Of course, the two descriptions of CONFORM$_s$ should be *compatible* to each other, i.e., they should define the same set. We then have the following restriction on the relations **imp**, **sat**, and **spec**:

$$\text{for all } i \in \mathcal{L}_{FDT}, i \textbf{ imp } s \text{ iff (for all } r \in \mathcal{L}_{RQ}, s \textbf{ spec } r \text{ implies } i \textbf{ sat } r).$$

We note that the formal specification s is in itself not enough to allow for conformance testing. We need instead either a pair s and **imp**, or the combination of s, $\mathcal{L}_{RQ}$, **sat**, and **spec**.

Consider now our definition of processes, tests, and preorders, and pick one particular preorder $\sqsubseteq_\alpha$. We clearly have a specification language $\mathcal{L}_{FDT}$ given by the set of processes and the underlying transition system. We then model s using our language and we obtain a specification. Then the relation **imp** is precisely given by the preorder $\sqsubseteq_\alpha$. The preorder gives us the tools for conformance testing using the behavior specification. If we provide a modal characterization for the preorder we can do testing using requirement specification too. Indeed, the set $\mathcal{L}_{RQ}$ is the set of formulae that constitute the modal characterization, the relation **sat** is our satisfaction predicate $\models$, and the function $(\cdot)^*$ defines the relation **spec**.

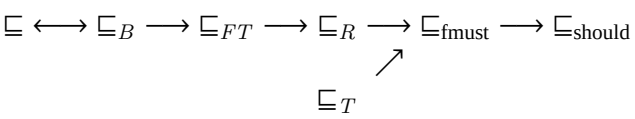

Fig. 6.14. Relations between preorders. The arrows $\sqsubseteq_\alpha \longrightarrow \sqsubseteq_\beta$ stand for "$p \sqsubseteq_\alpha q$ implies $p \sqsubseteq_\beta q$, but not the other way around."

It turns out that our theory of preorders has an immediate application in conformance testing. Indeed, all we did in this section was to translate the notation used elsewhere [Tre94] into the notation that we used in this chapter, and presto, we have a framework for formal conformance testing.

However, our framework is not fully practical because of the number of tests one needs to apply in order to check for conformance, which is often countably infinite. Elegant proof systems are not enough from a practical point of view, we also need to test implementations in a reasonable amount of time. We come back to our discussion on practical considerations. The observation preorder for instance, with its strong notion of observability, is unlikely in our opinion to create a realistic framework for conformance testing.

In any case, testing and test case generation in particular are also the subject of subsequent chapters, so our discussion about applications ends here.

6.10 Summary

We now conclude our presentation of preorder relations. We have surveyed quite a number of preorders, so before going any further a summary is in order. We have talked throughout this chapter about the following preorders:

$\sqsubseteq$	the observational testing preorder, as a general framework presented in Section 6.2.3
$\sqsubseteq_{CT}$	the complete trace preorder, presented in Section 6.3;
$\sqsubseteq_B$	observation preorder, the subject of Section 6.4;
$\sqsubseteq_T$	(aka $\sqsubseteq_{\text{conv}}$, together with $\sqsubseteq_{\text{may}}$ and $\sqsubseteq_{\text{must}}$), surveyed in Section 6.5;
$\sqsubseteq_R$	refusal preorder, presented in Section 6.6;
$\sqsubseteq_{FT}$	failure trace preorder, in Section 6.7;
$\sqsubseteq_{\text{fmust}}$	fair testing preorder, the subject of Section 6.8;
$\sqsubseteq_{\text{should}}$	should-testing preorder, a variant of $\sqsubseteq_{\text{fmust}}$, also a pre-congruence.

In addition, we have defined a generic testing scenario and the associated observable testing preorder $\sqsubseteq$. There exist preorders we did not consider specifically, such as Darondeau's preorder, because they were shown to coincide with preorders presented here [dN87]. We introduced trace preorders only because we had to start with something (and we decided to start with something simple), and because sometimes they make for useful comparison tools. However, trace preorders are awkward to work with, so we do not give too much thought to them henceforth.

One of the comparison criteria between preorders is their power of discrimination. In this respect, the observation preorder has been shown to coincide with the generic

preorder $\sqsubseteq$. The remaining preorders are strictly less discriminating and arrange themselves in a nice hierarchy. The only exception is the testing preorder, which is not comparable with the observation, failure trace, and refusal preorders. This is one reason for the introduction of $\sqsubseteq_{\text{fmust}}$, which has its place in the hierarchy allright. This comparison has been shown throughout the chapter by examples and propositions, and is summarized in Figure 6.14.

The relation $\sqsubseteq_{\text{fmust}}$ was also introduced because of fairness considerations (hence the name fair testing preorder). Specifically, the testing preorder deals unfairly with divergence, in the sense that divergence is reported as failure. In contrast, the fair interpretation of divergence implies that the tests succeed in presence of divergences as long as the system has a chance to eventually perform a visible action despite divergences. Since $\sqsubseteq_{\text{fmust}}$ is not a pre-congruence relation, the variant $\sqsubseteq_{\text{should}}$ (which is the largest pre-congruence included in $\sqsubseteq_{\text{fmust}}$) has also been defined.

Of course, the presence of fairness, or the greater power of discrimination are not an a priori good thing; it all depends on the desired properties one is interested in. The unfair interpretations of divergence in particular are useful in differentiating between livelock and deadlock, i.e., in detecting whether the system under test features busy-waiting loops and other such behaviors that are not deadlocked but are nonetheless unproductive (and undetectable under the fair testing scenario).

In terms of power of discrimination, we have noticed in Section 6.4 that the most discriminating preorder differentiates between processes that are for all practical purposes identical (see for example the processes shown in Figures 6.4 on page 145 and 6.5 on page 146). This is not to say that more differentiation is bad either, just look at the coffee machine examples from Figure 6.12 on page 156, which are in a strange implementation relation under refusal testing (only a crooked merchant would accept this) but are not comparable under failure trace preorder.

Another comparison of preorders can be made in terms of the complexity of the tests and their practical feasibility. It is no surprise that the most discriminating preorder, namely the observation preorder, appears to be the least practical of them all. In this respect the award of the most practically realizable preorder seems to go to refusal preorder. This is the only preorder based exclusively on sequential tests. This being said, we are not necessarily better off since in the general case we need a number of tests to figure out the properties of the system, so that the advantage of the tests being sequential pales somehow.

Another practical issue in refusal preorder is the concept of refusal itself. One can wonder how practical such a concept is. Recall that actions are an abstraction; in particular, they do not necessarily represent the acceptance of input. So how does one refuse an action without modifying the process under scrutiny itself? This does not seem realizable in the general case (whenever we cannot access the internals of the process under test). Do we take away the award from refusal preorder?

In all, practical considerations do differentiate between the preorders we talked about, especially for the observation preorder which combines results in a more complex way than other preorders (that simply take the union of the results of various runs and tests) and requires a rather unrealistic concept of global testing. However, when testing systems we are in the realm of the halting problem, so practical considerations

cannot ever make an a priori distinction. The utility of various preorders should thus be estimated by taking all of their features into consideration.

In the same line of thought, namely practical applications, we have presented a practical framework for conformance testing based on the theory of preorders.

Finally, it is worth pointing out that our presentation has been made in terms of labeled transition systems, as opposed to most of the literature, in which process algebraic languages such as CCS, LOTOS, and variants thereof are generally used. Labeled transition systems define however the semantics of all these languages, so the translation of the results surveyed here into various other formalisms should not be a problem. The upside of our approach is the uniform and concise characterization of the preorders, although we lose some expressiveness in doing so (however the literature cited therein always offers a second, most of the time process algebraic view of the domain).

As well, we did not pay attention to contexts. Contexts admit however a relatively straightforward approach once the rest of the apparatus is in place.

Literature

AB99. Paul Ammann and Paul E. Black. A specification-based coverage metric to evaluate test sets. In *Proceedings of the 4th IEEE International Symposium on High-Assurance Systems Engineering (HASE 1999)*, pages 239–248. IEEE Computer Society Press, 1999. [344]

ABD+79. A. Acree, T. Budd, R. DeMillo, R. Lipton, and F. Sayward. Mutation analysis. Technical Report GIT-ICS-79/08, School of Information and Computer Science, Georgia Institute of Technology, September 1979. [21]

ABM98. Paul Ammann, Paul E. Black, and William Majurski. Using model checking to generate tests from specifications. In *Proceedings of the 2nd IEEE International Conference on Formal Engineering Methods (ICFEM 1998)*, pages 46–54. IEEE Computer Society Press, 1998. [344]

Abr87. Samson Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987. [135, 139, 140, 141, 142, 144, 146, 147]

Abr96. J-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996. [336]

ACH+95. R. Alur, C. Courcoubetis, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995. A preliminary version appeared in Guy Cohen, editor, *Proceedings of 11th International Conference on Analysis and Optimization of Systems (ICAOS 1994): Discrete Event Systems*, volume 199 of *Lecture Notes in Control and Information Science*, pages 331–351. Springer-Verlag, 1994. [365, 366]

AD94. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. [215, 217, 218, 219, 226, 229, 351, 358, 362]

ADE+01. R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical hybrid modeling of embedded systems. In T. A. Henzinger and C. M. Kirsch, editors, *Proceedings of the 1st International Workshop on Embedded Software (EMSOFT 2001)*, volume 2211, pages 14–31. Springer-Verlag, 2001. [365, 367]

ADE+03. R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical hybrid modeling and analysis of embedded systems. *Proceedings of the IEEE*, 91(1):11–28, January 2003. [365, 367]

ADG+03. C. Artho, D. Drusinsky, A. Goldberg, K. Havelund, M. Lowry, C. Pasareanu, G. Roşu, and W. Visser. Experiments with test case generation and run-time analysis. In E. Börger, A. Gargantini, and E. Riccobene, editors, *Proceedings of the 10th International Workshop on Abstract State Machines (ASM 2003)*, volume 2589 of *Lecture Notes in Computer Science*, pages 87–107, Taormina, Italy, 2003. Springer-Verlag. Invited paper. [534, 535]

ADLU91. A. Aho, A. Dahbura, D. Lee, and Ü. Uyar. An optimization technique for protocol conformance test generation based on uio sequences and rural chinese postman tours. *IEEE Transactions on Communications*, 39(11):1604–1615, November 1991. [88, 89, 118, 121]

AFH94. Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. In *Proceedings of the 16th International Conference on Computer-aided Verification (CAV 1994)*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 1994. [219, 362]

AGE. AGEDIS homepage. http://www.agedis.de. [416, 417]

AGLS01. Rajeev Alur, Radu Grosu, Insup Lee, and Oleg Sokolsky. Compositional refinement for hierarchical hybrid systems. In Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC 2001)*, volume 2034 of *Lecture Notes in Computer Science*, pages 33–48. Springer-Verlag, 2001. [365, 367]

AGW77. Roy Adler, L. Wayne Goodwyn, and Benjamin Weiss. Equivalence of topological Markov shifts. *Israel Journal of Mathemtics*, 27(1):49–63, 1977. [52]

AH97. Rajeev Alur and Thomas A. Henzinger. Real-time system = discrete system + clock variables. *International Journal on Software Tools for Technology Transfer*, 1(1–2):86–109, 1997. [215]

AHB03. C. Artho, K. Havelund, and A. Biere. High-level data races. *Software Testing, Verification and Reliability*, 03(4):207–227, December 2003. Extended version of a paper in P. T. Isaías, F. Sedes, J. C. Augusto, and U. Ultes-Nitsche, editors, *New Technologies for Information Systems: Proceedings of the 3rd International Workshop on New Developments in Digital Libraries (NDDL 2003) and the 1st International Workshop on Validation and Verification of Software for Enterprise Information Systems (VVEIS 2003)*; in conjunction with the *5th International Conference on Enterprise Information Systems*, pages 82–93. ICEIS Press, 2003. [535]

AHP99. Pavel Atanassov, Stefan Haberl, and Peter Puschner. Heuristic worst-case execution time analysis. In *Proceedings of the 10th European Workshop on Dependable Computing*, pages 109–114. Austrian Computer Society (OCG), May 1999. [374]

AHU74. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974. [590, 591]

Ald90. Rudie Alderden. COOPER – the compositional construction of a canonical tester. In Son T. Vuong, editor, *Proceedings of the IFIP TC/WG6.1 2nd International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE 1989)*, pages 13–17, Vancouver, BC, Canada, 1990. North-Holland. [405, 407]

Alu99. Rajeev Alur. Timed automata. In Nicolas Halbwachs and Doron Peled, editors, *Proceedings of the 11th Internation Conference on Computer Aided Verification (CAV 1999)*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer-Verlag, 1999. [216, 218, 220]

ÁMHS01. Erika Ábrahám-Mumm, Ulrich Hannemann, and Martin Steffen. Verification of hybrid systems: Formalization and proof rules in PVS. In *Proceedings of the 7th International Conference on Engineering of Complex Computer Systems (ICECCS 2001)*, pages 48–57. IEEE Computer Society Press, 2001. [365]

Amt00. Peter Amthor. *Structural Decomposition of Hybrid Systems*. Number 13 in Monographs of the Bremen Institute of Safe Systems. University of Bremen, 2000. [366]

Ang87. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987. [507, 563, 573, 578]

ASMa. ASM homepage. http://www.eecs.umich.edu/gasm/. [403]

ASMb. AsmL download. http://research.microsoft.com/fse/asml/. [403]

Aus99. T. M. Austin. DIVA: A reliable substrate for deep submicron microarchitecture design. In *Proceedings of the 32nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 1999)*, pages 196–207, November 1999. [511]

BB87. Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks*, 14:25–29, 1987. [135, 177, 407, 413]

BBHP03. Kirsten Berkenkötter, Stefan Bisanz, Ulrich Hannemann, and Jan Peleska. HybridUML profile for UML 2.0. In *Proceedings of the Workshop on Specification*

and Validation of UML models for Real Time and Embedded Systems (SVERTS) in conjunction with the ⟨⟨UML⟩⟩ 2003 Conference, October 2003. Available at `http://www-verimag.imag.fr/EVENTS/2003/SVERTS`. [365, 367]

BC85. Gérard Berry and Laurent Cosserat. The ESTEREL synchronous programming language. In Stephen D. Brookes, A. W. Roscoe, and Glynn Winskel, editors, *Proceedings of the Seminar on Concurrency, 1984*, volume 197 of *Lecture Notes in Computer Science*, pages 389–448. Springer-Verlag, 1985. [387]

BC00. M. Bernardo and R. Cleaveland. A theory of testing for Markovian processes. In C. Palamidessi, editor, *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000)*, number 1877 in Lecture Notes in Computer Science, pages 305–319. Springer-Verlag, 2000. [246, 256, 268, 275, 276, 278, 279]

BCG+00. Dahananjay S. Brahme, Steven Cox, Jim Gallo, Mark Glasser, William Grundmann, C. Norris Ip, William Paulsen, John L. Pierce, John Rose, Dean Shea, and Karl Whiting. The transaction-based verification methodology. Technical report, Cadence Design Systems, Inc., August 2000. [444]

BCGM00. Simon Burton, John A. Clark, Andy J. Galloway, and John A. McDermid. Automated V&V for high integrity systems, a target formal methods approach. In C. Michael Holloway, editor, *Proceedings of the 5th NASA Langley Formal Methods Workshop (Lfm 2000)*, number NASA/CP-2000-210100 in NASA Conference Publications, pages 129–140, 2000. [327]

BCK03. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 2nd edition, 2003. [11]

BCL92. J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In A. Halaas and P. B. Denyer, editors, *Proceedings of the International Conference on Very Large Scale Integration (VLSI 1991)*, volume A-1 of *IFIP Transactions*, pages 49–58, Edinburgh, Scotland, 1992. North-Holland. [548]

BCM00. Simon Burton, John A. Clark, and John A. McDermid. Testing, proof and automation. An integrated approach. In *Proceedings of the 1st International Workshop of Automated Program Analysis, Testing and Verification (WAPATV 2000)*, pages 57–63, 2000. In Conjunction with the 22nd International Conference on Software Engineering (ICSE 2000). [324]

BCM01. Simon Burton, John A. Clark, and John A. McDermid. Automatic generation of tests from Statechart specifications. In Ed Brinksma and Jan Tretmans, editors, *Proceedings of the 1st International Workshop on Formal Approaches to Testing of Software (FATES 2001)*, number BRICS NS-01-4 in Basic Research in Computer Science (BRICS) Notes Series, pages 31–46, 2001. [327]

BCMD90. J. R. Burch, E. M. Clarke, K. L. McMillan, and David L. Dill. Sequential circuit verification using symbolic model checking. In *Proceedings of the 27th ACM/IEEE Conference on Design Automation Conference (DAC 1990)*, pages 46–51. ACM Press, 1990. [548]

BCMS01. O. Burkart, D. Caucal, F. Moller, and Bernhard Steffen. Verification on infinite structures. In S. Smolka J. Bergstra, A. Pons, editor, *Handbook on Process Algebra*. North-Holland, 2001. [542]

BDGW94. J. L. Balcázar, J. Díaz, R. Gavaldà, and O. Watanabe. The query complexity of learning DFA. *New Generation Computing*, 12:337–358, 1994. [578]

BDGW97. J. L. Balcázar, J. Díaz, R. Gavaldà, and O. Watanabe. Algorithms for learning finite automata from queries: A unified view. In Ding-Zhu Du, Ker-I Ko, and Dingzhu Du, editors, *Advances in Algorithms, Languages, and Complexity*. Kluwer Academic, February 1997. In Honor of Ronald V. Book. [558, 570, 575]

Bei95. B. Beizer. *Black-Box Testing*. John Wiley & Sons, 1995. [484]

Bel57. R. Bellman. *Dynamic programming*. Princeton University Press, 1957. [220]

BFdV+99. A. Belinfante, J. Feenstra, R. G. de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink. Formal test automation: A simple experiment. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, *Proceedings of the 12th International Workshop on Testing of Communicating Systems (IWTCS 1999)*, volume 147 of *IFIP Conference Proceedings*, pages 179–196. Kluwer Academic, 1999. [410, 413, 424, 426, 429, 433, 436, 437, 438, 445, 448]

BFG+99. Marius Bozga, Jean-Claude Fernandez, Lucian Ghirvu, Susanne Graf, Jean-Pierre Krimm, and Laurent Mounier. IF: An intermediate representation and validation environment for timed asynchronous systems. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *Proceedings of the World Congress on Formal Methods in the Development of Computing Systems, Volume I (FM 1999)*, volume 1708 of *Lecture Notes in Computer Science*, pages 307–327, Toulouse, France, 1999. Springer-Verlag. [409, 410]

BFMW01. D. Bartetzko, C. Fischer, M. Moller, and H. Wehrheim. Jass – Java with assertions. In Klaus Havelund and Grigore Rosu, editors, *Runtime Verification (RV 2001)*, volume 55(2) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2001. [537]

BG96. M. Bernardo and R. Gorrieri. Extended markovian process algebra. In U. Montanari and V. Sassone, editors, *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR 1996)*, volume 1119 of *Lecture Notes in Computer Science*, pages 315–330. Springer-Verlag, 1996. [254]

BGHM96. N. H. Bshouty, S. A. Goldman, T. R. Hancock, and S. Matar. Asking queries to minimize errors. *Journal of Computer and Systems Science*, 52:268–286, 1996. [578]

BGK+02. K. Bhargavan, C. A. Gunter, M. Kim, I. Lee, D. Obradovic, O. Sokolsky, and M. Viswanathan. Verisim: Formal analysis of network simulations. *IEEE Transactions on Software Engineering*, 28(2):129–145, February 2002. [535]

BGM91. Gilles Bernot, Marie-Claude Gaudel, and Bruno Marre. Software testing based on formal specifications: A theory and a tool. *Software Engineering Journal*, 6(6):387–405, 1991. [291, 327, 328, 329]

BGN+03. Mike Barnett, Wolfgang Grieskamp, Lev Nachmanson, Wolfram Schulte, Nikolai Tillman, and Margus Veanes. Towards a tool environment for model-based testing with AsmL. In *Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES 2003)*, volume 2931 of *Lecture Notes in Computer Science*, pages 252–266. Springer-Verlag, 2003. [404, 405]

BGS84. B. Boehm, T. Gray, and T. Seewaldt. Protoyping versus specifying: A multiproject experiment. *IEEE Transactions on Software Engineering*, SE-10(3):290–303, 1984. [11]

BH89. Ferenc Belina and Dieter Hogrefe. The CCITT Specification and Description Language SDL. *Computer Networks and ISDN Systems*, 16(4):311–341, March 1989. [23]

BHKW03. Christel Baier, Holger Hermanns, Joost-Pieter Katoen, and Verena Wolf. Comparative branching time semantics for Markov chains. In *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR 2003)*, volume 2761 of *Lecture Notes in Computer Science*, pages 492–507. Springer-Verlag, 2003. [255, 282]

BIĆP99. Stojan Bogdanović, Balázs Imreh, Miroslav Ćirić, and Tatjana Petković. Directable automata and their generalizations. *Novi Sad Journal of Mathematics*, 29(2):29–69, 1999. [28, 51]

Bin99. R. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison Wesley, 1999. [3]

BJLS03. Therese Berg, Bengt Jonsson, Martin Leucker, and Mayank Saksena. Insights to Angluin's learning. In *Proceedings of the International Workshop on Software Verification and Validation (SVV 2003)*, Electronic Notes in Theoretical Computer Science, December 2003. To appear. [576]

BLL+95. Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL – A tool suite for automatic verification of real-time systems. In *Proceedings of the 3rd DIMACS/SYCON Workshop on Hybrid Systems: Verification and Control*, volume 1066 of *Lecture Notes in Computer Science*, pages 232–243. Springer-Verlag, October 1995. [236, 238]

Blua. Bluetooth Project, `http://www.iti.uni-luebeck.de/Research/MUC/EKG/`. [491]

Blub. Bluetooth Special Interest Group. *Specification of the Bluetooth System (version 1.1)*. `http://www.bluetooth.com`. [491]

BPDG98. Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2–3):142–182, November 1998. [218]

BPR93. L. Boullier, M. Phalippou, and A. Rouger. Experimenting test selection strategies. In *Proceedings of the 6th SDL Forum, 1993*, pages 267–278. Elsevier Science Publishers, 1993. [212]

BR87. P. Berman and R. Roos. Learning one-counter languages in polynomial time. In *Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science (FOCS 1987)*, pages 61–67, Los Alamitos, CA, 1987. IEEE Computer Society Press. [578]

Bri89. E. Brinksma. A theory for the derivation of tests. In S. Aggarwal and K. Sabnani, editors, *Proceedings of the 8th IFIP Symposium on Protocol Specification, Testing and Verification (PSTV 1988)*. North-Holland, 1989. [167, 168, 176, 212, 405, 407]

Bro86. F. Brooks. No silver bullet. In H.-J. Kugler, editor, *Information Processing – Proceedings of the 10th IFIP World Computer Congress (WCC 1986)*, pages 1069–1076. North-Holland/IFIP, 1986. [5]

BRRdS96. Amar Bouali, Anni Ressouche, Valérie Roy, and Robert de Simone. The FCTOOLS user manual. Technical report, INRIA Sophia Antipolis, April 1996. [226]

BRV95. Ed Brinksma, Arend Rensink, and Walter Vogler. Fair testing. In Insup Lee and Scott A. Smolka, editors, *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR 1995)*, volume 962 of *Lecture Notes in Computer Science*, pages 313–327. Springer-Verlag, 1995. [157, 158, 159]

Bry85. R. E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *Proceedings of the 22nd ACM/IEEE Design Automation Conference (DAC 1985)*, pages 688–694. IEEE Computer Society Press, June 1985. [389]

BS01a. Mike Barnett and Wolfram Schulte. Spying on components: A runtime verification technique. In *Proceedings of the OOPSLA 2001 Workshop on Specification and Verification of Component-Based Systems (SAVBS 2001)*, 2001. Published as Technical Report ISU TR #01-09a, Iowa State University. [515]

BS01b. M. Broy and K. Stølen. *Specification and Development of Interactive Systems – Focus on Streams, Interfaces, and Refinement*. Springer-Verlag, 2001. [5]

Büc62. J. R. Büchi. On a decision method in restricted second-order arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Proceedings of the 1st International Congress for Logic, Methodology, and Philosophy of Science (LMPS 1960)*, pages 1–12. Stanford University Press, 1962. [541, 548]

Bur00. Simon Burton. Automated testing from Z specifications. Technical Report YCS-2000-329, University of York, 2000. [324, 327]

Bur02. Simon Burton. *Automated Generation of High Integrity Test Suites from Graphical Specifi cations*. PhD thesis, University of York, March 2002. [324, 327]

BvdLV95. Tommaso Bolognesi, Jeroen van de Lagemaat, and Chris Vissers, editors. *LOTOSphere: Software Development with LOTOS*. Kluwer Academic, 1995. [405]

BY01. Luciano Baresi and Michal Young. Test oracles. Technical Report CIS-TR-01-02, University of Oregon, Deptartment of Computer and Information Science, Eugene, Oregon, U.S.A., August 2001. [514]

CC91. Linda Christoff and Ivan Christoff. Efficient algorithms for verifi cation of equivalences for probabilistic processes. In Kim Guldstrand Larsen and Arne Skou, editors, *Proceedings of the 3rd International Conference on Computer Aided Verifi cation (CAV 1991)*, volume 575 of *Lecture Notes in Computer Science*, pages 310–321. Springer-Verlag, 1991. [286]

CCG+02. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An opensource tool for symbolic model checking. In E. Brinksma and K. Guldstrand Larsen, editors, *Proceedings of the 14th International Conference on Computer-Aided Verifi cation (CAV 2002)*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364, Copenhagen, Denmark, July 2002. Springer-Verlag. [556]

CCG+03. Sagar Chaki, Edmund M. Clarke, Alex Groce, Somesh Jha, and Helmut Veith. Modular verifi cation of software components in C. In *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, pages 385–395. IEEE Computer Society Press, 2003. [319]

CEP95. CEPSCO. GSM 11.11, Digital cellular telecommunications systems (phase2+); Specifi cations of the subscriber identity module – mobile equipment (SIM-ME) interface (GSM 11.11), 1995. [434]

CEP00. CEPSCO. Common electronic purse specifi cation: Technical specifi cation, 2000. `http://www.cepsco.org`. [434]

Čer64. Ján Černý. Poznámka k. homogénnym experimentom s konecnými automatmi. *Matematicko-fysikalny Casopis SAV*, 14:208–215, 1964. [40, 51]

CGPT96. M. Clatin, R. Groz, M. Phalippou, and R. Thummel. Two approaches linking test generation with verifi cation techniques. In A. Cavalli and S. Budkowski, editors, *Proceedings of the 8th International Workshop on Protocol Test Systems (IWPTS 1996)*. Chapman & Hall, 1996. [212, 213, 401, 402]

Che02. Albert M. K. Cheng. *Real-Time Systems; Scheduling, Analysis, and Verifi cation*. John Wiley & Sons, 2002. [351]

Cho78. Tsun S. Chow. Testing software design modeled by fi nite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, May 1978. Special collection based on the 2nd International Computer Software and Applications Conference (COMPSAC 1978). [111, 113, 125, 126, 127, 173, 233, 361, 573, 578]

Chr90. Ivan Christoff. Testing equivalences and fully abstract models for probabilistic processes. In *Proceedings of the 1st International Conference on Concurrency Theory (CONCUR 1990)*, volume 458 of *Lecture Notes in Computer Science*, pages 126–140. Springer-Verlag, 1990. [245, 256, 258, 261, 275, 276, 278, 279]

CJR96. Zhou Chaochen, Wang Ji, and Anders P. Ravn. A formal description of hybrid systems. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the DIMACS/SYCON 1995 Workshop on Hybrid Systems III: Verifi cation and Control*, volume 1066 of *Lecture Notes in Computer Science*, pages 511–530. Springer-Verlag, 1996. [365]

CJRZ01. Duncan Clarke, Thierry Jéron, Vlad Rusu, and Elena Zinovieva. Automated test and oracle generation for Smart-Card applications. In *Smart Card Programming and*

Security. Proceedings of the International Conference on Research in Smart Cards (E-smard 2001), volume 2140 of *Lecture Notes in Computer Science*, pages 58–70. Springer-Verlag, 2001. [414, 416, 429, 434, 436, 437, 438, 439, 445, 446, 448]

CJRZ02. D. Clarke, T. Jéron, V. Rusu, and E. Zinovieva. STG: A symbolic test generation tool. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *Lecture Notes in Computer Science*, pages 470–475. Springer-Verlag, 2002. [413, 414, 415, 416]

CJSP93. H. Cho, S.-W. Jeong, F. Somenzi, and C. Pixley. Multiple observation time single reference test generation using synchronizing sequences. In *Proceedings of the European Conference on Design Automation (EDAC 1993) with the European Event in ASIC Design*, pages 494–498. IEEE Computer Society Press, February 1993. [29, 52]

CKK02. Karel Culik, Juhani Karhumäki, and Jarkko Kari. A note on synchronized automata and road coloring problem. In Werner Kuich, Grzegorz Rozenberg, and Arto Salomaa, editors, *Proceedings of the 5th International Conference on Developments in Language Theory (DLT 2001)*, volume 2295 of *Lecture Notes in Computer Science*, pages 175–185. Springer-Verlag, 2002. [28, 52]

CL95. Duncan Clarke and Insup Lee. Testing real-time constraints in a process algebraic setting. In *Proceedings of the 17th International Conference on Software Engineering (ICSE 1995)*, pages 51–60. ACM Press, 1995. [355]

CL97a. Duncan Clarke and Insup Lee. Automatic generation of tests for timing constraints from requirements. In *Proceedings of the 3rd International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 1997)*, pages 199–206. IEEE Computer Society Press, 1997. [355]

CL97b. Duncan Clarke and Insup Lee. Automatic test generation for the analysis of a real-time system: Case study. In *Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium (RTAS 1997)*, pages 112–124. IEEE Computer Society Press, 1997. [355, 357]

Cla76. Lori A. Clarke. A system to generate test data and symbolically execute programs. *IEEE Transactions on Software Engineering*, SE-2(3):215–222, September 1976. [332, 336]

CLRS01. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill Book Company, Cambridge, MA, 2 edition, 2001. [37]

CO00. Rachel Cardell-Oliver. Conformance tests for real-time systems with timed automata specifi cations. *Formal Aspects of Computing*, 12(5):350–371, 2000. [235, 238, 242, 243, 358]

COG98. Rachel Cardell-Oliver and Tim Glover. A practical and complete algorithm for testing real-time systems. In A. P. Ravn and H. Rischel, editors, *Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 1998)*, volume 1486 of *Lecture Notes in Computer Science*, pages 251–261. Springer-Verlag, 1998. [235, 241]

Coh. S. Cohen. JTrek. Compaq, `http://www.compaq.com/java/download/jtrek`. [525]

CPB98. S. Barbey C. Peraire and D. Buchs. Test selection for object-oriented software based on formal specifi cations. In D. Gries and W. P. de Roever, editors, *Proceedings of the International Conference on Programming Concepts and Methods (PROCOMET 1998)*, volume 125 of *IFIP Conference Proceedings*, pages 385–403. Chapman and Hall, 1998. [340]

CPHP87. P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. LUSTRE: A declarative language for programming synchronous systems. In *Conference Record of the 14th Annual*

ACM Symposium on Principles of Programming Languages (POPL 1987), pages 178–188, Munich, Germany, January 21–23, 1987. ACM SIGACT-SIGPLAN, ACM Press. [386]

ČPR71. Ján Černý, Alica Pirická, and Blanka Rosenauerová. On directable automata. *Kybernetika*, 7(4):289–298, 1971. [28, 51]

CPS93. R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based verification tool for finite state systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 15(1):36–72, January 1993. [556]

CS92. Rance Cleaveland and Bernhard Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In Kim G. Larsen and Arne Skou, editors, *Proceedings of the 3rd Conference on Computer Aided Verification (CAV 1991)*, volume 575, pages 48–58, Berlin, Germany, 1992. Springer-Verlag. [547]

CSE96. J. Callahan, F. Schneider, and S. Easterbrook. Specification-based testing using model checking. In *Proceedings of the 2nd SPIN Workshop 1996*, pages 193–207, 1996. [344]

CSZ92. R. Cleaveland, S. Smolka, and A. Zwarico. Testing preorders for probabilistic processes. In W. Kuich, editor, *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP 1992)*, volume 623 of *Lecture Notes in Computer Science*, pages 708–719. Springer-Verlag, 1992. [245, 256, 258, 263, 274, 276, 278]

DAV93. K. Drira, P. Azéma, and F. Vernadat. Refusal graphs for conformance tester generation and simplification: A computational framework. In A. Danthine, G. Leduc, and P. Wolper, editors, *Proceedings of the 8th International Symposium on Protocol Specification, Testing and Verification (PSTV 1993)*, volume C-16 of *IFIP Transactions*, pages 257–272. North-Holland, 1993. [181, 184]

Dav02. Gordon B. Davis. Anytime/anyplace computing and the future of knowledge work. *Communications of the ACM*, 45(12):67–73, December 2002. [509]

DBG01. Julia Dushina, Mike Benjamin, and Daniel Geist. Semi-formal test generation with Genevieve. In *Proceedings of the Design Automation Conference (DAC 2001)*, pages 617–622. ACM Press, 2001. [429, 432, 433, 434, 437, 438, 439, 443, 444, 445, 446, 447, 448]

dBORZ99. Lydie du Bousquet, Farid Ouabdesselam, Jean-Luc Richier, and N. Zuanon. Lutess: A specification-driven testing environment for synchronous software. In *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*, pages 267–276. ACM Press, 1999. [386, 388, 390]

dBRS+00. L. du Bousquet, S. Ramangalahy, S. Simon, C. Viho, A. Belinfante, and R. G. de Vries. Formal test automation: The conference protocol with TGV/Torx. In H. Ural, R. L. Probert, and G. von Bochmann, editors, *Proceedings of the IFIP 13th International Conference on Testing of Communicating Systems (TestCom 2000)*, volume 176 of *IFIP Conference Proceedings*, pages 221–228. Kluwer Academic, 2000. [410, 411, 413, 424, 426, 433]

dBZ99. L. du Bousquet and N. Zuanon. An overview of Lutess, a specification-based tool for testing synchronous software. In *Proceedings of the 14th IEEE International Conference on Automated Software Engineering (ASE 1999)*, pages 208–215. IEEE Computer Society Press, October 1999. [424]

Den91. Richard Denney. Test-case generation from Prolog-based specifications. *IEEE Software*, 8(2):49–57, 1991. [327, 329]

DF93. Jeremy Dick and Alain Faivre. Automating the generation and sequencing of test cases from model-based specifications. In Jim C. P. Woodcock and Peter G. Larsen, editors, *Proceedings of the 1st International Symposium of Formal Methods Europe:*

Industrial-Strength Formal Methods (FME 1993), volume 670 of *Lecture Notes in Computer Science*, pages 268–284. Springer-Verlag, 1993. [321]

DGJV01. S. Dudani, J. Geada, G. Jakacki, and D. Vainer. Dynamic assertions using TXP. In K. Havelund and G. Roşu, editors, *Proceedings of the 1st Workshop on Run-time Verifi cation (RV 2001)*, volume 55(2) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2001. [537]

DGN02. Z. R. Dai, J. Grabowski, and H. Neukirchen. Timed TTCN-3 – A Real-Time Extension for TTCN-3. In I. Schieferdecker, H. König, and A. Wolisz, editors, *Testing of Communicating Systems*, volume 14, Berlin, March 2002. Kluwer Academic. [491]

DGNP04. Z. R. Dai, J. Grabowski, H. Neukirchen, and H. Pals. From design to test with UML. In R. Groz and R. Hierons, editors, *Proceedings of the 16th IFIP International Conference on Testing of Communication Systems (TestCom 2004)*, Lecture Notes in Computer Science, pages 33–49. Springer-Verlag, March 2004. [483]

DH03a. F. Drewes and J. Högberg. Learning a regular tree language from a teacher. In Z. Ésik and Z. Fülöp, editors, *Proceedings of the 7th International Conference on Developments in Language Theory (DLT 2003)*, volume 2710 of *Lecture Notes in Computer Science*, pages 279–291. Springer-Verlag, 2003. [578]

DH03b. F. Drewes and J. Högberg. Learning a regular tree language from a teacher even more efficiently. Technical Report 03.11, Umeå University, 2003. [578]

DJ01. M. Ducasse and E. Jahier. Efficient automated trace analysis: Examples with morphine. In K. Havelund and G. Rosu, editors, *Proceedings of the 1st Workshop on Runtime Verifi cation (RV 2001)*, volume 55(2) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2001. [537]

DJC94. Michel Diaz, Guy Juanole, and Jean-Pierre Courtiat. Observer – a concept for formal on-line validation of distributed systems. *IEEE Transactions on Software Engineering*, 20(12):900–913, 1994. [515]

DLP77. R. DeMillo, R. Lipton, and A. Perlis. Social processes and proofs of theorems and programs. In *Conference Record of the 4th ACM SIGACT-SIGPLAN Symposium of Principles of Programming Languages (POPL 1977)*, pages 206–214, 1977. [17]

DN84. J. Duran and S. C. Ntafos. An Evaluation of Random Testing. *IEEE Transactions on Software Engineering*, SE-10(4):438–444, July 1984. [7, 18, 299, 301, 302, 303]

dN87. Rocco de Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24:211–237, 1987. [142, 161]

dNH84. Rocco de Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984. [139, 148, 149, 245, 270, 276]

DS95. Jim Davies and Steve Schneider. A brief history of timed CSP. *Theoretical Computer Science*, 138(2):243–271, 1995. [351]

DS03. D. Drusinsky and M. T. Shing. Monitoring temporal logic specifi cations combined with time series constraints. *Journal of Universal Computer Science*, 9(11):1261–1276, November 2003. [514]

Duc90. M. Ducasse. Opium: An extendable trace analyser for Prolog. *The Journal of Logic Programming*, 39:177–223, 1990. [537]

Duc99. M. Ducasse. Coca: An automated debugger for C. In *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*, pages 504–513. ACM Press, 1999. [537]

dVBF02. René G. de Vries, Axel Belinfante, and Jan Feenstra. Automated testing in practice: The highway tolling system. In Ina Schieferdecker, Harmut König, and Adam Wolisz, editors, *Proceedings of the IFIP 14th International Conference on Testing Communicating Systems (TestCom 2002)*, volume 210 of *IFIP Conference Proceedings*, pages 219–234. Kluwer Academic, 2002. [410, 413]

dVT98. R. G. de Vries and J. Tretmans. On-the-fly conformance testing using SPIN. In G. Holzmann, E. Najm, and A. Serhrouchni, editors, *Proceedings of the 4th Workshop on Automata Theoretic Verification with the* SPIN *Model Checker (SPIN 1998)*, number 98 S 002 in ENST Technical Report, pages 115–128, Paris, France, November 1998. Ecole Nationale Supérieure des Télécommunications. [213, 413]

ECGN01. Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, February 2001. [531, 532, 536, 537]

ECla. EClipse. `http://www.eclipse.org/`. [488]

ECLb. Eclipse constraint logic programming system. `http://www.icparc.ic.ac.uk/eclipse`. [396]

Eer94. E. H. Eertink. *Simulation Techniques for the Validation of LOTOS Specifications*. PhD thesis, University of Twente, Enschede, Netherlands, March 1994. [413]

EFM97. A. Engels, L. Feijs, and S. Mauw. Test generation for intelligent networks using model checking. In Ed Brinksma, editor, *Proceeedings of the 3rd International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS 1997)*, volume 1217 of *Lecture Notes in Computer Science*, pages 384–398. Springer-Verlag, 1997. [344]

EGKN99. Michael D. Ernst, William G. Griswold, Yoshio Kataoka, and David Notkin. Dynamically discovering pointer-based program invariants. Technical Report UW-CSE-99-11-02, University of Washington, Seattle, WA, November 1999. [532]

EH00. K. Etessami and G. J. Holzmann. Optimizing Büchi automata. In C. Palamidessi, editor, *Proceedings of 11th International Conference on Concurrency Theory (CONCUR 2000)*, volume 1877 of *Lecture Notes in Computer Science*, pages 153–167. Springer-Verlag, 2000. [548]

EJ73. J. Edmonds and E. L. Johnson. Matching, Euler tours and the chinese postman. *Mathematical Programming*, 5:88–124, 1973. [110]

EL85. E. Allen Emerson and Chin-Laung Lei. Modalities for model checking (extended abstract): Branching time strikes back. In *Conference Record of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 1985)*, pages 84–96. ACM Press, 1985. [547]

Epp90. David Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510, June 1990. [28, 29, 37, 38, 46, 51, 52]

EW92. E. H. Eertink and D. Wolz. Symbolic Execution of LOTOS Specifications. In M. Diaz and R. Groz, editors, *Proceedings of the 5th International Conference on Formal Description Techniques (FORTE 1992)*, pages 295–310. North-Holland, 1992. [413]

Fel68. W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley and Sons, 1968. [248]

Fet88. J. Fetzer. Program verification: The very idea. *Communications of the ACM*, 37(9):1048–1063, September 1988. [6, 17]

FGK+96. J-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A protocol validation and verification toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verification (CAV 1996)*, volume 1102 of *Lecture Notes in Computer Science*, pages 437–440. Springer-Verlag, 1996. [408, 409, 410, 413]

FGM+92. J.-C. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez, and J. Sifakis. A toolbox for the verification of LOTOS programs. In *Proceedings of the 14th International Conference on Software Engineering (ICSE 1992)*, pages 246–259. ACM Press, 1992. [556]

FHP02. Eitan Farchi, Alan Hartman, and Shlomit Pinter. Using a model-based test generator to test for standard conformance. *IBM Systems Journal*, 41(1):89–110, 2002. [429, 433, 436, 437, 438, 439, 443, 445, 446, 447, 448]

FHS96. A. Schmidt F. Huber, B. Schätz and K. Spies. Autofocus – A tool for distributed systems specification. In *Proceedings of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 1996)*, volume 1135 of *Lecture Notes in Computer Science*, pages 467–470. Springer-Verlag, 1996. [336]

FJJV96. J.-C. Fernandez, C. Jard, T. Jéron, and G. Viho. Using on-the-fly verification techniques for the generation of test suites. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verification (CAV 1996)*, volume 1102 of *Lecture Notes in Computer Science*, pages 348–359, New Brunswick, NJ, USA, 1996. Springer-Verlag. [291, 408]

FK00. D. Freitag and N. Kushmerick. Boosted wrapper induction. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000) and 12th Conference on Innovative Applications of Artificial Intelligence (IAAI 2000)*, Austin, Texas, August 2000. American Association for Artificial Intelligence, The AAAI Press. Copublished and distributed by The MIT Press. [533]

FKL99. Lauret Fournier, Anatoly Koyfman, and Moshe Levinger. Developing an architecture validation suite – application to the PowerPC architecture. In *Proceedings of the 36th ACM Design Automation Conference (DAC 1999)*, pages 189–194. ACM Press, 1999. [429, 433, 435, 437, 438, 440, 445, 446, 449]

FKR+93. Y. Freund, M. Kearns, D. Ron, R. Rubinfeld, R. Schapire, and L. Sellie. Efficient learning of typical finite automata from random walks. In *Proceedings of the 25th ACM Symposium on the Theory of Computing (STOC 1993)*, pages 315–324, New York, NY, 1993. ACM Press. [578]

Fou03. Apache Software Foundation. Byte code engineering library (BCEL). `http://jakarta.apache.org/bcel/`, 2003. Subproject of Jakarta. [514]

Fra. France Telecom R&D website. `http://www.rd.francetelecom.com/`. [401]

Fri90. Joel Friedman. On the road coloring problem. *Proceedings of the American Mathematical Society*, 110(4):1133–1135, December 1990. [28]

FvBK+91. S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, June 1991. [114, 115, 118]

FW88. P. G. Frankl and E. J. Weyuker. An applicable family of data flow testing criteria. *IEEE Transactions on Software Engineering*, SE-14:1483–1498, October 1988. [18, 294, 297]

FW93. P. G. Frankl and E. J. Weyuker. A formal analysis of the fault-detecting ability of testing methods. *IEEE Transactions on Software Engineering*, 19(3):202–213, 1993. [304]

Gar98. Hubert Garavel. OPEN/CAESAR: An open software architecture for verification, simulation, and testing. In B. Steffen, editor, *Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1998)*, volume 1384 of *Lecture Notes in Computer Science*, pages 68–84. Springer-Verlag, 1998. [409, 410, 413]

Gei01. M. Geilen. On the construction of monitors for temporal logic properties. In K. Havelund and G. Roşu, editors, *Proceedings of the 1st International Workshop on Run-time Verification (RV 2001)*, volume 55(2) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2001. [537]

GG75. John B. Goodenough and Susan L. Gerhart. Toward a theory of test data selection. *IEEE Transactions on Software Engineering*, 1(2):156–173, June 1975. [7, 17, 106]

GG93. Matthias Grochtmann and Klaus Grimm. Classifi cation trees for partition testing. *Software Testing, Verifi cation and Reliability*, 3(2):63–82, 1993. [323]

GGSV02. Wolfgang Grieskamp, Yuri Gurevich, Wolfram Schulte, and Margus Veanes. Generating fi nite state machines from abstract state machines. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2002)*, pages 112–122. ACM Press, 2002. [404, 405]

GH99. A. Gargantini and C. Heitmeyer. Using model checking to generate tests from requirements specifi cations. In O. Nierstrasz and M. Lemoine, editors, *Proceedings of the 7th European Software Engineering Conference, held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Enigneegin (ESEC/FSE 1999)*, volume 1687 of *Lecture Notes in Computer Science*, pages 146–163. Springer-Verlag, 1999. [344]

GH03. A. Goldberg and K. Havelund. Instrumentation of java bytecode for runtime analysis. In S. Eisenbach, G. T. Leavens, Peter Müller, A. Poetzsch-Heffter, and E. Poll, editors, *Proceedings of the 5th ECOOP Workshop on Formal Techniques for Java-like Programs (FTfJP 2003)*, pages 151–159, July 2003. Technical Report tr_408, ETH Zürich. [514]

GHHD04. Q. Guo, R. M. Hierons, M. Harman, and K. Derderian. Computing unique input/output sequences using genetic algorithms. In A. Petrenko and A. Ulrich, editors, *Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES 2003)*, volume 2931 of *Lecture Notes in Computer Science*, pages 164–177. Springer-Verlag, 2004. [101, 102, 103]

GHR93. N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specifi cation and performance analysis using stochastic process algebras. In Lorenzo Donatiello and Randolph D. Nelson, editors, *Performance Evaluation of Computer and Communication Systems, Joint Tutorial Papers of Performance 1993 and Sigmetrics 1993*, volume 729 of *Lecture Notes in Computer Science*, pages 121–146. Springer-Verlag, 1993. [254]

Gil61. Arthur Gill. State-identifi cation experiments in fi nite automata. *Information and Control*, 4(2–3):132–154, September 1961. [33, 45, 51, 56]

Gin58. Seymour Ginsburg. On the length of the smallest uniform experiment which distinguishes the terminal states of a machine. *Journal of the ACM (JACM)*, 5(3):266–280, July 1958. [29, 30, 34, 51]

GJ79. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, New York, 1979. [46, 48]

GJL04. O. Grinchtein, B. Jonsson, and M. Leucker. Learning of event-recording automata. Technical report, Uppsala University, 2004. [576, 578]

GJS97. J. Gosling, B. Joy, and G. Steele. *The Java Language Specifi cation*. The Sunsoft Press Java Series. Addison-Wesley, New York, 1997. [434]

GL02. Hubert Garavel and Frédéric Lang. NTIF: A general symbolic model for communicating sequential processes with data. In Doron Peled and Moshe Y. Vardi, editors, *Proceedings on the 22nd IFIP/WG6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2002)*, volume 2529 of *Lecture Notes in Computer Science*, pages 276–291, Houston, Texas, USA, November 2002. Springer-Verlag. Full version available as INRIA Research Report RR-4666. [416]

GO01. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of the 13th Conference on Computer Aided Verifi cation (CAV 2001)*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer-Verlag, 2001. [548]

Gog01. Nicolae Goga. Comparing TorX, Autolink, TGV and UIO test algorithms. In Rick Reed and Jeanne Reed, editors, *Meeting UML – Proceedings of the 10th International*

SDL Forum, 2001, volume 2078 of *Lecture Notes in Computer Science*, pages 379–402. Springer-Verlag, 2001. [422, 423]

Göh98. Wolf Göhring. Minimal initializing word: a contribution to Černý's conjecture. *Journal of Automata, Languages and Combinatorics*, 2(4):209–226, 1998. [28, 51]

Gol89. David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, 1989. [373]

Gon01. L. Gong. JXTA: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, May/June 2001. [511]

GPY02. A. Groce, D. Peled, and M. Yannakakis. Adaptive model checking. In J.-P. Katoen and P. Stevens, editors, *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *Lecture Notes in Computer Science*, pages 357–370. Springer-Verlag, 2002. [507, 578, 580, 581]

GRMD01. A. Q. Gates, S. Roach, O. Mondragon, and N. Delgado. DynaMICs: Comprehensive support for run-time monitoring. In K. Havelund and G. Rosu, editors, *Proceedings of the 1st International Workshop on Runtime Verification (RV 2001)*, volume 55(2) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2001. [537]

GRR03. A. Gargantini, E. Riccobene, and S. Rinzivillo. Using Spin to generate tests from ASM specifications. In E. Börger, A. Gargantini, and E. Riccobene, editors, *Proceedings of the 10th International Workshop on Abstract State Machines (ASM 2003)*, volume 2589 of *Lecture Notes in Computer Science*, pages 263–277. Springer-Verlag, 2003. [344]

Gru68. F. Gruenberger. *Computers and communication: Toward a computer utility*. Prentice-Hall, 1968. [536]

GS97. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification (CAV 1997)*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83, Haifa, Israel, 1997. Springer-Verlag. [14]

GSDH97. Jens Grabowski, Rudolf Scheuer, Zhen Ru Dai, and Dieter Hogrefe. Applying SAMSTAG to the B-ISDN protocol SSCOP. In M. Kim, S. Kang, and K. Hong, editors, *Proceedings of the 10th International Workshop on Testing of Communication Systems (IWTCS 1997)*. Chapman & Hall, 1997. [420, 422]

GSSL99. D. F. Gordon, W. M. Spears, O. Sokolsky, and Insup Lee. Distributed spatial control, global monitoring and steering of mobile physical agents. In *Proceedings of the IEEE International Conference on Information, Intelligence, and Systems (ICIIS 1999)*, pages 681–688. IEEE Computer Society Press, November 1999. [535]

Gur94. Yuri Gurevich. Evolving algebras 1993: Lipari Guide. In Egon Börger, editor, *Specification and Validation Methods*, pages 9–37. Oxford University Press, 1994. [23, 403, 405]

Gut99. W. Gutjahr. Partition testing versus random testing: The influence of uncertainty. *IEEE Transactions on Software Engineering*, 25(5):661–674, 1999. [7, 18, 299, 303]

GW98. Mathias Grochtmann and Joachim Wegener. Evolutionary testing of temporal correctness. In *Proceedings of the 2nd Software Quality Week Europe (QWE 1998)*, Brussel, Belgium, 1998. [374]

Ham94. R. Hamlet. Random testing. In J. J. Marciniak, editor, *Encyclopedia of Software Testing*, volume 2. Addison-Wesley, 1994. [7, 18]

Har87. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987. [23]

Har00. Jerry J. Harrow, Jr. Runtime checking of multithreaded applications with visual threads. In K. Havelund, J. Penix, and W. Visser, editors, *SPIN Model Checking and Software Verifi cation: Proceedings of the 7th International SPIN Workshop, 2000*, volume 1885 of *Lecture Notes in Computer Science*, pages 331–342. Springer-Verlag, 2000. [517]

HCL+03. Hyoung Seok Hong, Sung Deok Cha, Insup Lee, Oleg Sokolsky, and Hasan Ural. Data fbw testing as model checking. In *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, pages 232–242. IEEE Computer Society Press, 2003. [344]

HCRP91. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-fbw programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991. [386, 387]

Hen64. F. C. Hennie. Fault detecting experiments for sequential circuits. In *Proceedings of the 5th Annual Symposium on Switching Circuit Theory and Logical Design*, pages 95–110, Princeton, New Jersey, 11–13 November 1964. IEEE Computer Society Press. [45, 118, 121]

Hen88. M. Hennessy. *Algebraic Theory of Processes*. MIT Press, Cambridge, 1988. [197]

Hen96. Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS 1996)*, pages 278–292. IEEE Computer Society Press, 1996. [365, 366]

HFT00. L. Heerink, J. Feenstra, and J. Tretmans. Formal Test Automation: The Conference Protocol with PHACT. In H. Ural, R. L. Probert, and G. von Bochmann, editors, *Proceedings of the 13th IFIP International Conference on Testing of Communicating Systems (TestCom 2000)*, pages 211–220. Kluwer Academic, 2000. [400, 401, 424, 426, 433]

HGW04. M. Heimdahl, D. George, and R. Weber. Specifi cation test coverage adequacy criteria = specifi cation test generation inadequacy criteria? In *Proceedings of the 8th IEEE High Assurance in Systems Engineering Workshop (HASE 2004)*, pages 178–186. IEEE Computer Society Press, March 2004. [21]

HHK96. R. H. Hardin, Zvi Har'El, and Robert P. Kurshan. COSPAN. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verifi cation (CAV 1996)*, volume 1102 of *Lecture Notes in Computer Science*, pages 423–427. Springer-Verlag, 1996. [556]

HHWT97. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Sotftware Tools for Technology Transfer*, 1(1–2):110–122, 1997. [415, 416]

Hib61. Thomas N. Hibbard. Least upper bounds on minimal terminal state experiments for two classes of sequential machines. *Journal of the ACM (JACM)*, 8(4):601–612, October 1961. [29, 35, 42, 51]

Hil96. Jane Hillston. *A compositional approach to performance modelling*. Cambridge University Press, 1996. [254]

HJGP99. Wai-Ming Ho, Jean-Marc Jézéquel, Alain Le Guennec, and François Pennaneac'h. UMLAUT: An extendible UML transformation framework. In *Proceedings of the 14th IEEE International Conference on Automated Software Engineering (ASE 1999)*, pages 275–278, Florida, 1999. IEEE Computer Society Press. [409, 410]

HJL96. C. Heitmeyer, R. Jeffords, and B. Labaw. Automated Consistency Checking of Requirements Specifi cations. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, July 1996. [23]

HJL03. John Hakansson, Bengt Jonsson, and Ola Lundqvist. Generating on-line test oracles from temporal logic specifi cations. *International Journal on Software Tools for Technology Transfer*, 4(4):456–471, 2003. [537]

HK87. Z. Har'El and R. P. Kurshan. *COSPAN User Guide*. AT&T Bell Laboratories, October 1987. [556]

HKWT95. Thomas A. Henzinger, Peter W. Kopke, and Howard Wong-Toi. The expressive power of clocks. In *Automata, Languages and Programming*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer-Verlag, 1995. [217]

HL02a. Sudheendra Hangal and Monica S. Lam. Tracking down software bugs using automatic anomaly detection. In *Proceedings of the 24th International Conference on Software Engineering (ICSE 2002)*, pages 291–301. ACM Press, 2002. [534, 536]

HL02b. R. Hightower and N. Lesiecki. *Java Tools for eXtreme Programming*. Wiley Computer Publishing. John Wiley & Sons, 2002. [487]

HLSU02. H. S. Hong, I. Lee, O. Sokolsky, and H. Ural. A temporal logic based theory of test coverage and generation. In J.-P. Katoen and P. Stevens, editors, *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *Lecture Notes in Computer Science*, pages 327–341. Springer-Verlag, 2002. [7, 344]

HM80. M. Henessy and R. Milner. Observing nondeterminism and concurrency. In J. de Bakker and M. van Leeuwen, editors, *Proceedings of the 7th International Colloquium on Automata, Languages, and Programming (ICALP 1980)*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer-Verlag, 1980. [144]

HM85. Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985. [282, 547]

HME03. Michael Harder, Jeff Mellen, and Michael D. Ernst. Improving test suites via operational abstraction. In *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, pages 60–71, Portland, Oregon, 2003. IEEE Computer Society Press. [531]

HMP92. Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *Proceedings of the 19th International Colloquium on Automata, Languages, and Programming (ICALP 1992)*, volume 632 of *Lecture Notes in Computer Science*, pages 545–558. Springer-Verlag, 1992. [215]

HN83. M. Hennessy and R. De Nicola. Testing equivalences for processes. In *Proceedings of the 10th Interantional Colloquium on Automata, Languages and Programming (ICALP 1983)*, 1983. [130, 221, 224]

HN99. A. Hartman and K. Nagin. TCBeans, software test toolkit. In *Proceedings of the 12th International Software Quality Week (QW 1999)*, 1999. [434, 439]

HNS97. Steffen Helke, Thomas Neustupny, and Thomas Santen. Automating test case generation from Z specifi cations with Isabelle. In Jonathan P. Bowen, Michael G. Hinchey, and David Till, editors, *Proceedings of the 10th International Conference of Z Users: The Z Formal Specifi cation Notation (ZUM 1997)*, volume 1212 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1997. [321, 322, 323, 324]

HNS03. H. Hungar, O. Niese, and B. Steffen. Domain-specifi c optimization in automata learning. In W. A. Hunt Jr. and F. Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verifi cation (CAV 2003)*, Lecture Notes in Computer Science, pages 315–327. Springer-Verlag, 2003. [575, 577]

HNSY92. Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. In *Proceedings of the 7th Symposium of Logics in Computer Science (LICS 1992)*, pages 394–406, Santa Cruz, California, 1992. IEEE Computer Scienty Press. [217]

Hoa85. C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice-Hall, 1985. [351]

Hol91. Gerard J. Holzmann. *Design and Validation of Protocols*. Prentice-Hall Software Series, 990157918X. Prentice-Hall, Englewood Cliffs, N. J., 1991. [25, 108, 110, 413]

Hol97. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997. [556]

Hol01. G. Holzmann. From code to models. In *Proceedings of the 2nd International Conference on Applications of Concurrency to System Design (ACSD 2001)*, pages 3–10. IEEE Computer Society Press, 2001. [14]

Hop71. J. E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi and A. Paz, editors, *Proceedings of the International Symposium on Theory of Machines and Computations, 1971*, pages 189–196, New York, 1971. Academic Press. [84, 85, 590, 591]

How77. W. Howden. Symbolic testing and the dissect symbolic evaluation system. *IEEE Trans. on Software Engineering*, SE-3(4):266–278, July 1977. [335]

HPPS03a. G. Hahn, J. Philipps, A. Pretschner, and T. Stauner. Prototype-based tests for hybrid reactive systems. In *Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping (RSP 2003)*, pages 78–86. IEEE Computer Society Press, 2003. [368, 369, 370]

HPPS03b. G. Hahn, J. Phillips, A. Pretschner, and T. Stauner. Tests for mixed discrete-continuous reactive systems. Technical Report TUM-I0301, Institut für Informatik, TU München, 2003. [368, 369]

HR01a. K. Havelund and G. Roşu. Monitoring Java programs with Java PathExplorer. In K. Havelund and G. Roşu, editors, *Proceedings of the 1st Workshop on Runtime Verification (RV 2001)*, volume 55(2) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2001. [537]

HR01b. Klaus Havelund and Grigore Roşu. Java PathExplorer – a runtime verification tool. In *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2001): A New Space Odyssey*. 2001. Montreal, Canada. [510, 517]

HR02. K. Havelund and G. Roşu. Synthesizing monitors for safety properties. In J.-P. Katoen and P. Stevens, editors, *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *Lecture Notes in Computer Science*, pages 324–356. Springer-Verlag, 2002. [507, 521]

HR04. K. Havelund and G. Roşu. An overview of the runtime verification tool Java PathExplorer. *Formal Methods in System Design*, 24(2):189–215, 2004. [519, 534]

HSE97. F. Huber, B. Schätz, and G. Einert. Consistent graphical specification of distributed systems. In *Proceedings of the 4th International Symposium of Formal Methods Europe (FME 1997)*, volume 1313 of *Lecture Notes in Computer Science*, pages 122–141. Springer-Verlag, 1997. [397]

Hsi71. E. P. Hsieh. Checking experiments for sequential machines. *IEEE Transactions on Computers*, C-20:1152–1166, October 1971. [87, 89]

HT90. D. Hamlet and R. Taylor. Partition test does not inspire confidence. *IEEE Transactions on Software Engineering*, 16(12):1402–1411, December 1990. [7, 18, 299, 301]

HT92. Dung T. Huynh and Lu Tian. On some equivalence relations for probabilistic processes. *Fundamenta Informaticae*, 17:211–234, 1992. [286]

HU79. J. Hopcroft and J. Ullman. *Introduction to automata theory, languages and computation*. Addison-Wesley, 1979. [210]

IBM. IBM. Gotcha users guide – release 4.0.0. [435]

ID84. M. Ito and Jürgen Duske. On cofinal and definite automata. *Acta Cybernetica*, 6(2):181–189, 1984. [28]

Ins03. ETSI (European Telecommunication Standards Institute). The testing and test control notation version 3. In *Methods for Testing and Specifi cation (MTC)*. ETSI, 2003. [455, 456, 467, 473, 483, 487]

IS95. Balázs Imreh and Magnus Steinby. Some remarks on directable automata. *Acta Cybernetica*, 12(1):23–35, 1995. [40, 51]

IS99. Balázs Imreh and Magnus Steinby. Directable nondeterministic automata. *Acta Cybernetica*, 14(1):105–115, 1999. [52]

ISO88. ISO/IEC. LOTOS – a formal description technique based on the temporal ordering of observational behaviour. International Standard 8807, International Organization for Standardization – Information Processing Systems – Open Systems Interconnection, Genève, September 1988. [176, 407, 413]

ISO94. ISO/IEC. Information technology – open systems interconnection – conformance testing methodology and framework, 1994. International ISO/IEC multi-part standard No. 9646. [485]

ISO02. ISO/IEC. Information technology – Z formal specifi cation notation – syntax, type system, and semantics. International Organization for Standardization ISO/IEC 13568, 2002. [319, 321]

ITU99. ITU. ITU-T recommendation Z.120: Message sequence charts (MSC). ITU Telecommunication Standard Sector, Geneva (Switzerland), 1999. [456]

ITU02. ITU. The evolution of TTCN. `http://www.itu.int/ITU-T/studygroups/com07/ttcn.html`, 2002. [453]

Jac01. M. Jackson. *Problem Frames: Analyzing and structuring software development problems*. Addison Wesley, 2001. [9]

JBu. JBuilder. `http://www.borland.com/jbuilder/personal/index.html`. [488]

JG90. F. Jahanian and A. Goyal. A formalism for monitoring real-time constraints at runtime. In *Proceedings of the 20th International Symposium on Fault-Tolerant Computing Systems (FTCS 1990)*, pages 148–155, 1990. [514]

JGL91. Bengt Jonsson and Kim Guldstrand Larsen. Specifi cation and refi nement of probabilistic processes. In *Proceedings of the 6th IEEE International Symposium on Logic in Computer Science (LICS 1991)*, pages 266–277. IEEE Computer Society Press, 1991. [246, 281]

JJ02. Claude Jard and Thierry Jéron. TGV: Theory, principles and algorithms. In *Proceedings of the 6th World Conference on Integrated Design and Process Technology (IDPT 2002)*. Society for Design and Process Science, June 2002. [408, 410]

Jon91. Bengt Jonsson. Simulations between specifi cations of distributed systems. In *Proceedings of the 2nd International Conference on Concurrency Theory (CONCUR 1991)*, volume 527 of *Lecture Notes in Computer Science*, pages 346–360. Springer-Verlag, 1991. [267, 281]

JUn. JUnit. `http://www.junit.org/`. [487]

JY95. Bengt Jonsson and Wang Yi. Compositional testing preorders for probabilistic processes. In *Proceedings of the 10th IEEE International Symposium on Logic in Computer Science (LICS 1995)*, pages 431–441. IEEE Computer Society Press, 1995. [246, 263, 267, 276]

JY02. Bengt Jonsson and Wang Yi. Testing preorders for probabilistic processes can be characterized by simulations. *Theoretical Computer Science*, 282(1):33–51, 2002. [246, 256, 257, 263, 266, 267, 276, 280, 281]

Kar03. Jarkko Kari. Synchronizing fi nite automata on eulerian digraphs. *Theoretical Computer Science*, 295(1–3):223–232, 2003. [51, 52]

KCS98. K. Narayan Kumar, Rance Cleaveland, and Scott Smolka. Infi nite probabilistic and non-probabilistic testing. In *Proceedings of the 18th Conference on Foundations*

of Software Technology and Theoretical Computer Science (FSTTCS 1998), volume 1530 of *Lecture Notes in Computer Science*, pages 209–220. Springer-Verlag, 1998. [246, 256]

Kfo70. Denis J. Kfoury. Synchronizing sequences for probabilistic automata. *Studies in Applied Mathematics*, 49(1):101–103, March 1970. [52]

KGHS98. B. Koch, J. Grabowski, D. Hogrefe, and M. Schmitt. AutoLink – a tool for automatic test generation from SDL specifications. In *Proceedings of the IEEE International Workshop on Industrial Strength Formal Specification Techniques (WIFT 1998)*, October 1998. [420, 421, 422]

KHMP94. Arjun Kapun, Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Proving safety properties of hybrid systems. In Hans Langmaack, Willem P. de Roever, and Jan Vytopil, editors, *Proceedings of the 3rd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 1994)*, volume 863 of *Lecture Notes in Computer Science*, pages 431–454. Springer-Verlag, 1994. [365]

Kin76. James C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, July 1976. [331, 334, 335]

KJG99. A. Kerbrat, T. Jéron, and R. Groz. Automated test generation from SDL specifications. In R. Dssouli, G. von Bochmann, and Y. Lahav, editors, *The Next Millennium – Proceedings of the 9th SDL Forum, 1999)*, pages 135–152. Elsevier Science Publishers, 1999. [409, 410]

KKL+01. M. Kim, S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan. Java-MaC: A run-time assurance tool for Java programs. In *Proccedings of the 1st International Workshop on Run-Time Verification (RV 2001)*, volume 55(2) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, July 2001. [510, 525]

KKL+02. M. Kim, S. Kannan, I. Lee, M. Viswanathan, and O. Sokolsky. Computational analysis of run-time monitoring. In K. Havelund and G. Roşu, editors, *Proccedings of the 2nd Workshop on Run-Time Verification (RV 2002)*, volume 70(4). Elsevier Science Publishers, 2002. [530, 535]

KKLS01. Moonjoo Kim, Sampath Kannan, Insup Lee, and Oleg Sokolsky. Java-MaC: a run-time assurance tool for Java. In *Prodeedings of the 1st International Workshop on Run-time Verification (RV 2001)*, volume 55 of *Electronic Notes in Theoretical Computer Science*, Paris, France, July 2001. Elsevier Science Publishers. [514]

KLS+02. M. Kim, I. Lee, U. Sammapun, J. Shin, and O. Sokolsky. Monitoring, checking, and steering of real-time systems. In K. Havelund and G. Roşu, editors, *Proceedings of the 2nd Workshop on Run-time Verification (RV 2002)*, volume 70(4) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2002. [530, 535, 537]

KMP+95. W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpiesman, and D. Wonnacott. The Omega library interface guide. Technical Report UMIACS-TR-95-41, University of Maryland at College Park, 1995. `http://www.cs.umd.edu/projects/omega/`. [416]

Koh78. Zvi Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, New York, NY, second edition, 1978. [51, 59, 60, 66, 85, 87, 88, 89, 174]

Kop97. Hermann Kopetz. *Real-Time Systems – Design Principles for Distributed Embedded Applications*. Kluwer Academic, 1997. ISBN: 0-7923-9894-7. [374]

Koz77. Dexter Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 254–266, Providence, Rhode Island, October 1977. IEEE Computer Society Press. [48, 68]

Koz83. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, December 1983. [547]

KPN. KPN website. http://www.kpn.com/. [399]

KPV03. Sarfraz Khurshid, Corina S. Pasareanu, and Willem Visser. Generalized symbolic execution for model checking and testing. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2003. [335, 342, 343]

KRS87. A. A. Klyachko, I. K. Rystsov, and M. A. Spivak. An extremal combinatorial problem associated with the bound on the length of a synchronizing word in an automaton. *Kibernetika*, 25(2):165–171, March–April 1987. Translation from Russian. [28, 40, 51]

KS76. J. G. Kemeny and J. L. Snell. *Finite Markov Chains.* Springer-Verlag, 1976. [249]

KSW96. Kolyang, Thomas Santen, and Burkhart Wolff. A structure preserving encoding of Z in Isabelle/HOL. In Joakim von Wright, Jim Grundy, and John Harrison, editors, *Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 1996)*, volume 1125 of *Lecture Notes in Computer Science*, pages 283–298. Springer-Verlag, 1996. [321]

KV94. M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, Cambridge, Massachusetts and London, England, 1994. [570, 574]

KVZ98. Hakim Kahlouche, Cesar Viho, and Massimo Zendri. An industrial experiment in automatic generation of executable test suites for a cache coherency protocol. In A. Petrenko and N. Yevtushenko, editors, *Proceedings of the 11th IFIP/TC6 International Workshop on Testing of Communicating Systems (TestCom 1998)*. Chapman & Hall, September 1998. [429, 433, 435, 437, 438, 439, 444, 445, 446, 448]

Kwa62. Mei-Ko Kwan. Graphic programming using odd or even points. *Chinese Math*, 1:273–277, 1962. [110]

Lai02. Richard Lai. A survey of communication protocol testing. *Journal of Systems and Software*, 62(1):21–46, May 2002. [25]

Lal85. P. Lala. *Fault Tolerant and Fault Testable Hardware Design*. Prentice-Hall International, 1985. [88]

Lan90. Rom Langerak. A testing theory for lotos using deadlock detection. In Ed Brinksma, Giuseppe Scollo, and Chris A. Vissers, editors, *Proceedings of the IFIP/WG6.1 9th International Symposium on Protocol Specification, Testing and Verification (PSTV 1989)*, pages 87–98. North-Holland, 1990. [156]

LBGG94. I. Lee, P. Brémond-Grégoire, and R. Gerber. A process algebraic approach to the specification and analysis of resource-bound real-time systems. *Proceedings of the IEEE*, 82(1):158–171, January 1994. [351]

LDW03. Tessa Lau, Pedro Domingos, and Daniel S. Weld. Learning programs from traces using version space algebra. In *Proceedings of the International Conference on Knowledge Capture (K-CAP 2003)*, Sanibel Island, FL, USA, 2003. ACM Press. [534]

Lit. Lite ftp and web sites. ftp://ftp.cs.utwente.nl/pub/src/lotos-tools/ and http://fmt.cs.utwente.nl/tools/lite/. [405, 407]

LKK+99. I. Lee, S. Kannan, M. Kim, O. Sokolsky, and M. Viswanathan. Runtime assurance based on formal specifications. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 1999), Volume 1*, pages 279–287. CSREA Press, 1999. [525, 528]

LLC03. Glen McCluskey & Associates LLC. Java™ test coverage and instrumentation toolkits. http://www.glenmccl.com/, 2003. [514]

LP81. H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, 1981. [110]

LP01. B. Legeard and F. Peureux. Génération de séquences de tests à partir d'une spécification B en PLC ensembliste. In *Actes des Approches Formelles dans*

l'Assistance au Développement de Logiciels (AFADL 2001), pages 113–130, June 2001. [332, 337, 341]

LPU02. B. Legeard, F. Peureux, and M. Utting. Automated Boundary Testing from Z and B. In *Proceedings of the International Conference on Formal Methods Europe (FME 2002)*, volume 2391 of *Lecture Notes in Computer Science*, pages 21–40, Copenhagen, Denmark, July 2002. Springer-Verlag. [332, 337]

LPU04. B. Legeard, F. Peureux, and M. Utting. Controlling test case explosion in test generation from B formal models. *Software Testing, Verification and Reliability (STVR)*, 14(2):81–103, 2004. [337]

LPY97. Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, 1997. [235, 358]

LS91. Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991. [256, 263, 282, 284]

LSV01. Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata revisited. In Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC 2001)*, volume 2034 of *Lecture Notes in Computer Science*, pages 403–417. Springer-Verlag, 2001. [365]

LSW97. K. G. Larsen, B. Steffen, and C. Weise. Continuous modelling of real time and hybrid systems: From concepts to tools. *International Journal on Software Tools for Technology Transfer*, 1(1–2):64–85, 1997. [365]

LT87. N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC 1987)*, pages 137–151. ACM Press, 1987. Also: Technical Report MIT/LCS/TM-387, Massachusetts Institute of Technology, Cambridge, U.S.A., 1987. [187, 189]

LT89. N. A. Lynch and M. R. Tuttle. An introduction to Input/Output Automata. *CWI Quarterly*, 2(3):219–246, 1989. Also: Technical Report MIT/LCS/TM-373 (TM-351 revised), Massachusetts Institute of Technology, Cambridge, U.S.A., 1988. [189]

LvBP94. G. Luo, G. von Bochmann, and A. Petrenko. Selecting test sequences for partially-specified non deterministic finite state machines. In *Proceedings of the 7th International Workshop on Protocol Test Systems (IWPTS 1994)*, pages 91–106, Japan, February 1994. [204]

LY94. David Lee and Mihalis Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, March 1994. [25, 50, 59, 60, 66, 68, 69, 70, 74, 78, 81, 82, 83, 84, 85, 87, 90, 93, 96, 103, 589]

LY96. David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines – a survey. *Proceedings of the IEEE*, 84(8):1090–1126, 1996. [25, 33, 51, 60, 85, 99, 105, 107, 111, 124, 589]

MA00. B. Marre and A. Arnould. Test sequence generation from lustre descriptions: GATEL. In *Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE 2000)*, Grenoble, 2000. IEEE Computer Society Press. [393]

Mah99. Savi Maharaj. Towards a method of test case extraction from correctness proofs. Presented at the *14th International Workshop on Algebraic Development Techniques (WADT 1999)*, 1999. [329, 330]

Mah00. Savi Maharaj. Test case extraction from correctness proofs. University of Stirling, 2000. Case for Support. [329]

ME03. Stephen McCamant and Michael D. Ernst. Predicting problems caused by component upgrades. In *Proceedings of the 9th European Software Engineering Conference*

(ESEC 2003). Held jointly with the *11th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2003)*, pages 287–296. ACM Press, 2003. [533, 536]

Mel88. Thomas Melham. Abstraction mechanisms for hardware verifi cation. In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specifi cation, Verifi cation, and Synthesis*, volume 35 of *The Kluwer International Series in Engineering and Computer Science*, pages 129–157. Kluwer Academic, Boston, 1988. [444]

Mey79. G. Meyer. *The Art of Software Testing*. John Wiley & Sons, Inc., 1979. [294, 295]

Mey92. Bertrand Meyer. Design by contract. *IEEE Computer*, 25(10):40–52, October 1992. [514]

Mey01. Oliver Meyer. *Structural Decomposition of Timed-CSP and its Application in Real-Time Testing*. Dissertation, University of Bremen, 2001. Number 16 in *Monographs of the Bremen Institute of Safe Systems*. [351]

MH99. V. Matena and M. Hapner. Enterprise javabeans^{TM} specifi cation. Public Draft version 1.1, Sun Microsystems, 1999. [511]

Mil80. R. Milner. *A Calculus for Communicating Processes*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980. [135, 144, 159]

Mil89. Robin Milner. *Communication and concurrency*. Prentice-Hall, 1989. [353, 547, 581]

MK99. Jeff Magee and Jeff Kramer. *Concurrency: State Models & Java Programs*. John Wiley & Sons, 1999. [413]

ML97. Aloysius K. Mok and Guangtian Liu. Efficient run-time monitoring of timing constraints. In *IEEE Real-Time Technology and Applications Symposium*, June 1997. [514]

Moo56. Edward F. Moore. Gedanken-experiments on sequential machines. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, number 34 in Annals of Mathematics Studies, pages 129–153. Princeton University Press, Princeton, NJ, 1956. [33, 34, 35, 43, 51, 56, 59, 107]

MOSS99. M. Müller-Olm, D. Schmidt, and B. Steffen. Model checking: A tutorial introduction. In G. File A. Cortesi, editor, *Proceedings of the 6th Static Analysis Symposium (SAS'99)*, volume 1694 of *Lecture Notes in Computer Science*, pages 330–354, Heidelberg, Germany, September 1999. Springer-Verlag. [542]

MP93. R. Miller and S. Paul. On the generation of minimal-length conformance tests for communication protocols. *IEEE/ACM Transactions on Networking*, 1(1):116–129, February 1993. [89, 94]

MP95. O. Maler and A. Pnueli. On the learnability of infi nitary regular sets. *Information and Computation*, 118(2):316–326, 1 May 1995. [578]

MP03. Leonardo Mariani and Mauro Pezzè. Behavior capture and test for controlling the quality of component-based integrated systems. In *Proceedings of the Workshop on Tool-Integration in System Development (TIS 2003) at the 9th European Software Engineering Conference / 11th Symposium on Foundations of Software Engineering (ESEC/FSE 2003)*, pages 23–28, Helsinki, Finland, September 2003. [533, 534, 536]

MRS+97. J. R. Moonen, J. M. T. Romijn, O. Sies, J. G. Springintveld, L. G. M. Feijs, and R. L. C. Koymans. A two-level approach to automated conformance testing of VHDL designs. Technical Report SEN-R9707, CWI – Centrum voor Wiskunde en Informatica, Amsterdam, 1997. [401]

MS99. Alexandru Mateescu and Arto Salomaa. Many-valued truth functions, černý's conjecture and road coloring. *Bulletin of the EATCS*, 68:134–150, June 1999. [52]

MSF. Microsoft Research – Foundations of Software Engineering. `http://research.microsoft.com/fse/`. [403]

Müh97. H. Mühlenbein. Genetic algorithms. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 137–171. John Wiley & Sons, 1997. [101]

Mus93. J. D. Musa. Operational profiles in software-reliability engineering. *IEEE Software*, 10(2):14–32, March 1993. [386]

Nai97. K. Naik. Efficient computation of unique input/output sequences in finite-state machines. *IEEE/ACM Transactions on Networking*, 5(4):585–599, August 1997. [87, 89, 94, 95, 96, 100, 101, 103]

Nat86. B. K. Natarajan. An algorithmic approach to the automated design of parts orienters. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS 1986)*, pages 132–142, Toronto, Ontario, Canada, October 1986. IEEE. [29, 52]

NdFL95. Manuel Nuthez, David de Frutos, and Luis Llana. Acceptance trees for probabilistic processes. In Insup Lee and Scott A. Smolka, editors, *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR 1995)*, volume 962 of *Lecture Notes in Computer Science*, pages 249–263. Springer-Verlag, 1995. [246]

NH84. R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984. [193]

Nie00. B. Nielsen. *Speification and Test of Real-Time Systems*. PhD thesis, Department of Computer Science, Aalborg University, 2000. [362, 363]

NPW02. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002. [321, 324]

NS03. Brian Nielsen and Arne Skou. Automated test generation from timed automata. *International Journal on Software Tools for Technology Transfer*, 5:59–77, 2003. [218, 220, 226, 243]

NT81. S. Naito and M. Tsunoyama. Fault detection for sequential machines by transition tours. In *Proceedings of the 11th IEEE Fault Tolerant Computing Conference (FTCS 1981)*, pages 238–243. IEEE Computer Society Press, 1981. [110]

Nta88. S. C. Ntafos. A comparison of some structural testing strategies. *IEEE Transactions on Software Engineering*, SE-11:367–375, April 1988. [18, 297]

Nta98. Simeon Ntafos. On random and partition testing. In *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 1998)*, pages 42–48. ACM Press, 1998. [7, 18, 299, 302, 431]

OBG01. D. Buchs O. Biberstein and N. Guelfi. Object-oriented nets with algebraic specifications: The CO-OPN/2 formalism. In Gul Agha, Fiorella de Cindio, and Grzegorz Rozenberg, editors, *Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets*, volume 2001 of *Lecture Notes in Computer Science*, pages 73–130. Springer-Verlag, 2001. [336]

OMG02. Object Management Group (OMG). *UML Testing Profile – Request For Proposal*, April 2002. OMG Document (ad/01-07-08). [483]

OP95. F. Ouabdesselam and I. Parissis. Constructing operational profiles for synchronous crtitical software. In *Proceedings of the 6th International Symposium on Software Reliability Engineering (ISSRE 1995)*, pages 286–293. IEEE Computer Society Press, 1995. [390]

ORR+96. Sam Owre, Sreeranga Rajan, John M. Rushby, Natarajan Shankar, and Mandayam K. Srivas. PVS: Combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verification (CAV 1996)*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414. Springer-Verlag, 1996. [319]

ORS92. Sam Owre, John M. Rushby, and Natarajan Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction (CADE 1992)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer-Verlag, 1992. [330]

ORSvH95. S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification of fault-tolerant architerctures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, 1995. [415, 416]

PAD+98. Jan Peleska, Peter Amthor, Sabine Dick, Oliver Meyer, Michael Siegel, and Cornelia Zahlten. Testing reactive real-time systems. Tutorial, held at the *5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 1998)*, Denmark Technical University, Lyngby, 1998. Updated revision. Available as `http://www.informatik.uni-bremen.de/agbs/jp/papers/ftrtft98.ps`. [348, 349, 351, 366]

Pap94. Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, New York, 1994. [48]

Pat70. Michael S. Paterson. Unsolvability in 3×3 matrices. *Studies in Applied Mathematics*, 49(1):105–107, March 1970. [52]

Pau94. Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994. [321, 324]

PBD93. A. Petrenko, G. Bochmann, and R. Dussouli. Conformance relations and test derivation. In O. Rafiq, editor, *Proceedings of the 6th International Workshop on Protocol Test Systems (IWPTS 1993)*, volume C-19 of *IFIP Transactions*, pages 157–178, Pau, France, September 1993. North-Holland. [203]

PDGN03. H. Pals, Z. R. Dai, J. Grabowski, and H. Neukirchen. UML-based modeling of roaming with bluetooth devices. In Chun Chen, Walter Dosch, Yuntao Qian, and Huaizhong Lin, editors, *First Hangzhou-Lübeck Conference on Software Engineering (HL-SE'03)*, 2003. [491]

Pel02. Jan Peleska. Formal methods for test automation – hard real-time testing of controllers for the airbus aircraft family. In *Proceedings of the 6th Biennial World Conference on Integrated Design & Process Technology (IDPT 2002)*. Society for Design and Process Science, June 2002. [351]

Per98. Cecile Peraire. *Formal testing of object-oriented software: From the method to the tool*. PhD thesis, École Polytechnique Fédéral de Lausanne (EPFL), Switzerland, 1998. [341]

PF90. David H. Pitt and David Freestone. The derivation of conformance tests from LOTOS specifications. *IEEE Transactions on Software Engineering*, 16(12):1337–1343, December 1990. [177, 178, 179]

Pha91. M. Phalippou. Tveda: An experiment in computer-aided test case generation from formal specification of protocols. Technical Note NT/LAA/SLC/347, France Telecom – CNET, 1991. [212]

Pha93. M. Phalippou. The limited power of testing. In Gregor von Bochmann, Rachida Dssouli, and Anindya Das, editors, *Proceedings of the 5th International Workshop on Protocol Test Systems (IWPTS 1992)*, volume C-11 of *IFIP Transactions*, pages 43–54, Montréal, September 1993. North-Holland. [203, 209]

Pha94a. M. Phalippou. Executable testers. In Omar Rafiq, editor, *Proceedings of the 6th International Workshop on Protocol Test Systems (IWPTS 1993)*, volume C-19 of *IFIP Transactions*, pages 35–50, Pau, France, September 1994. North-Holland. [212]

Pha94b. M. Phalippou. *Relations d'Implantation et Hypothèses de Test sur des Automates à Entrées et Sorties*. PhD thesis, L'Université de Bordeaux I, France, 1994. [187, 189, 190, 201, 402]

Pha95. M. Phalippou. Abstract testing and concrete testers. In S. T. Vuong and S. T. Chanson, editors, *Proceedings of the 14th IFIP/WG6.1 International Symposium on Protocol Specifi cation, Testing and Verifi cation (PSTV 1994)*, volume 1 of *IFIP Conference Proceedings*, pages 221–236, Vancouver, June 1995. Chapman & Hall. [212]

Phi. Philips website. `http://www.philips.com/`. [400]

Phi87. Iain Phillips. Refusal testing. *Theoretical Computer Science*, 50:241–284, 1987. [141, 152, 153, 155]

Pin78a. Jean-Eric Pin. Sur les mots synchronisants dans un automate fi ni. *Elektronische Informationsverarbeitung und Kybernetik (EIK)*, 14:297–303, 1978. [51]

Pin78b. Jean-Eric Pin. Sur un cas particulier de la conjecture de černý. In Giorgio Ausiello and Corrado Böhm, editors, *Proceedings of the 5th Colloquium on Automata, Languages and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 345–352, Udine, Italy, July 1978. Springer-Verlag. [51, 52]

Pir95. L. Ferreira Pires. *Protocol Implementation: Manual for Practical Exercises 1995–1996*. University of Twente, the Netherlands, 1995. Lecture notes. [433]

PJH92. Carl Pixley, Seh-Woong Jeong, and Gary D. Hachtel. Exact calculation of synchronization sequences based on binary decision diagrams. In *Proceedings of the 29th Design Automation Conference (DAC 1992)*, pages 620–623. IEEE Computer Society Press, June 1992. [28, 52]

PLP03. A. Pretschner, H. Lötzbeyer, and J. Philipps. Model based testing in incremental system development. *Journal of Systems and Software*, 70(3):315–329, 2003. [15]

PN98. Peter Puschner and Roman Nossal. Testing the results of static worst-case execution-time analysis. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS 1998)*, pages 134–143. IEEE Computer Society Press, December 1998. [374]

Pnu77. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium Foundations of Computer Science (FOCS 1977)*, pages 46–57. IEEE Computer Society Press, 1977. [510, 513, 544]

PO96. I. Parissis and F. Ouabdesselam. Specifi cation-based testing of synchronous software. In D. Garlan, editor, *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE 1996)*, volume 21(6) of *ACM SIGSOFT Software Engineering Notes*, pages 127–134. ACM Press, 1996. [390]

PP04. Wolfgang Prenninger and Alexander Pretschner. Abstractions for model-based testing. In *Proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS 2004)*, Electronic Notes in Theoretical Computer Science. Elsevier Science Publishers, 2004. [11, 15, 441]

PPS+03. J. Philipps, A. Pretschner, O. Slotosch, E. Aiglstorfer, S. Kriebel, and K. Scholl. Model-based test case generation for smart cards. In *In Proceedings of the 8th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2003)*, Electronic Notes in Theoretical Computer Science. Elsivier, 2003. to appear. [5, 332, 336, 339, 340, 397, 429, 434, 436, 437, 438, 439, 440, 443, 444, 445, 446, 447, 448]

PR94. Irith Pomeranz and Sudhakar M. Reddy. Application of homing sequences to synchronous sequential circuit testing. *IEEE Transactions on Computers*, 43(5):569–580, May 1994. [43, 52]

Pre01. A. Pretschner. Classical search strategies for test case generation with constraint logic programming. In E. Brinksma and J. Tretmans, editors, *Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES 2001)*, number NS/01/4 in BRICS Notes Series, pages 47–60, 2001. Satellite Workshop on CONCUR 2001. [398]

Pre03. A. Pretschner. Compositional generation for MC/DC test suites. In *Proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS*

2003), volume 82(6) of *Electronic Notes in Theoretical Computer Science*, pages 1–11. Elsevier Science Publishers, 2003. [7, 15, 398]

PS01. Tatjana Petković and Magnus Steinby. On directable automata. *Journal of Automata, Languages and Combinatorics*, 6(2):205–220, 2001. [28]

PST96. Ben Potter, Jane Sinclair, and David Till. *An Introduction to Formal Specifi cation and Z*. Prentice Hall, second edition, 1996. [319, 321]

PTLP99. S. Prowell, C. Trammell, R. Linger, and J. Poore. *Cleanroom Software Engineering*. Addison Wesley, 1999. [11]

Put94. Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc, 1994. [251, 253]

PVY99. D. Peled, M. Vardi, and M. Yannakakis. Black box checking. In *Proceedings of the Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE 1999) and Protocol Specifi cation, Testing and Verifi cation (PSTV 1999)*, volume 156 of *IFIP Conference Proceedings*, pages 225–240. Kluwer Academic, 1999. [578]

PZ91. A. Parrish and S. Zweben. Analysis and refi nement of software test data adequacy properties. *IEEE Transactions on Software Engineering*, 17(6):565–581, June 1991. [7]

Rav96. Bala Ravikumar. A deterministic parallel algorithm for the homing sequence problem. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing (SPDP 1996)*, pages 512–520, New Orleans, LA, October 1996. IEEE Computer Society Press. [51]

Rav98. Bala Ravikumar. Parallel algorithms for fi nite automata problems. In José D. P. Rolim, editor, *Proceedings of the 10 Workshops of the 12th International Parallel Processing Symposium (IPPS 1998) and 9th Symposium on Parallel and Distributed Processing (SPDS 1998)*, volume 1388 of *Lecture Notes in Computer Science*, page 373. Springer-Verlag, 1998. [52]

RdBJ00. V. Rusu, L. du Bousquet, and T. Jéron. An approach to symbolic test generation. In W. Grieskamp, T. Santen, and B. Stoddart, editors, *Proceedings of the 2nd International Conference on Integrated Formal Methods (IFM 2000)*, volume 1945 of *Lecture Notes in Computer Science*, pages 338–357. Springer-Verlag, 2000. [188, 408, 414, 415, 416]

RG95. Anil S. Rao and Kenneth Y. Goldberg. Manipulating algebraic parts in the plane. *IEEE Transactions on Robotics and Automation (IEEETROB)*, 11(4):598–602, August 1995. [29, 52]

RH01a. S. Rayadurgam and M.P. Heimdahl. Coverage based test case generation using model checkers. In *Proceedings of the 8th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2001)*, pages 83–91. IEEE Computer Society Press, 2001. [7, 344]

RH01b. S. Rayadurgam and M. P. Heimdahl. Test-sequence generation from formal requirements models. In *Proceedings of the 6th IEEE International Symposium on High-Assurance Systems Engineering (HASE 2001)*, pages 23–31. IEEE Computer Society Press, 2001. [344]

RH01c. G. Roşu and K. Havelund. Synthesizing dynamic programming algorithms from linear temporal logic formulae. Technical report, RIACS, 2001. [517, 518]

RHC76. C. V. Ramamoorthy, S. F. Ho, and W. T. Chen. On the automated generation of program test data. In *Proceedings of the 2nd International Conference on Software Engineering (ICSE 1976)*, page 636. IEEE Computer Society Press, 1976. Abstract only. [335, 336]

Ris93. N. Risser. TVEDA V2 user guide. Technical Document DT/LAA/SLC/EVP/5, France Telecom – CNET, March 1993. [212]

RJB99. James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Massachusetts, USA, first edition, 1999. [23, 483]

RKS02. Orna Raz, Philip Koopman, and Mary Shaw. Semantic anomaly detection in online data sources. In *Proceedings of the 24th International Conference on Software Engineering (ICSE 2002)*, pages 302–312. ACM Press, 2002. [510, 533, 536]

RLNS00. K. Rustan, M. Leino, G. Nelson, and J. B. Saxe. ESC/Java user's manual. Technical Report 2000-002, Compaq Systems Research Center, Palo Alto, 2000. [537]

RNHW98. P. Raymond, X. Nicollin, N. Halbwachs, and D. Weber. Automatic testing of reactive systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS 1998)*, pages 200–209. IEEE Computer Society Press, 1998. [390, 391, 392]

RP92. A. Rouger and M. Phalippou. Test cases generation from formal specifications. In *Proceedings of the 14th International Switching Symposium (ISS 1992)*, page C10.2, Yokohama, October 1992. [211]

RS93. Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, April 1993. [30, 33, 47, 566, 578]

RSP93. June-Kyung Rho, Fabio Somenzi, and Carl Pixley. Minimum length synchronizing sequences of finite state machine. In *Proceedings of the 30th ACM/IEEE Design Automation Conference (DAC 1993)*, pages 463–468. ACM Press, June 1993. [45, 52]

Rus02. Vlad Rusu. Verification using test generation techniques. In L.-H. Eriksson and P. Lindsay, editors, *Getting IT Right: Proceedings of the 11th International Symposium of Formal Methods Europe (FME 2002)*, volume 2381 of *Lecture Notes in Computer Science*, pages 252–271. Springer-Verlag, 2002. [415]

RVL+97. Ted Romer, Geoff Voelker, Dennis Lee, Alec Wolman, Wayne Wong, Hank Levy, Brian Bershad, and Brad Chen. Instrumentation and optimization of Win32/Intel executables using Etch. In *Proceedings of the First USENIX Windows NT Workshop*, Seattle, WA, August 1997. [514]

RW85. S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, SE-11:367–375, April 1985. [294, 297]

RW88. C. Rich and R. Waters. The programmer's apprentice: A research overview. *IEEE Computer*, 21(11):10–25, 1988. [534]

RX96. Bala Ravikumar and Xuefeng Xiong. Randomized parallel algorithms for the homing sequence problem. In Adam W. Bojanczyk, editor, *Proceedings of the 25th International Conference on Parallel Processing (ICPP 1996), volume 2: Algorithms & Applications*, pages 82–89. IEEE Computer Society Press, August 1996. [52]

RX97. Bala Ravikumar and Xuefeng Xiong. Implementing sequential and parallel programs for the homing sequence problem. In Darrell R. Raymond, Derick Wood, and Sheng Yu, editors, *Proceedings of the 1st Workshop on Implementing Automata (WIA 1996)*, volume 1260 of *Lecture Notes in Computer Science*, pages 120–131. Springer-Verlag, 1997. [52]

Rys83. Igor K. Rystsov. Polynomial complete problems in automata theory. *Information Processing Letters*, 16(3):147–151, April 1983. [29, 41, 48, 50, 51, 52]

Rys92. Igor K. Rystsov. Rank of a finite automaton. *CYBERNETICS: Cybernetics and Systems Analysis*, 28(3):323–328, May 1992. Translation of *Kibernetika i Sistemnyi Analiz*, pages 3–10 in non-translated version. [28, 50, 52]

Rys97. Igor K. Rystsov. Reset words for commutative and solvable automata. *Theoretical Computer Science*, 172:273–279, February 1997. [28, 51]

SA99. Jian Shen and Jacob Abraham. An RTL abstraction technique for processor microarchitecture validation and test generation. *Journal of Electronic Testing: Theory & Application*, 16(1–2):67–81, February 1999. [14, 429, 433, 434, 435, 437, 438, 439, 444, 445, 446, 447]

Sad98. Sadegh Sadeghipour. *Testing Cyclic Software Components of Reactive Systems on the Basis of Formal Specifications*, volume 40 of *Forschungsergebnisse zur Informatik*. Verlag Dr. Kovač, Hamburg, 1998. [323, 324, 325]

Sal02. Arto Salomaa. Synchronization of finite automata. Contributions to an old problem. In I. Hal Sudborough T. Æ. Mogensen, D. A. Schmidt, editor, *The Essence of Computation. Complexity, Analysis, Transformation: Essays Dedicated to Neil D. Jones*, volume 2566 of *Lecture Notes in Computer Science*, pages 37–59. Springer-Verlag, 2002. [29]

Sav70. Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970. [48]

SBN+97. S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. E. Anderson. Eraser: A dynamic data race detector for multithreaded programs. *ACM Transactions on Computer Systems*, 14(4):391–411, November 1997. [510, 515, 516, 517]

Sch00. Johann M. Schumann. Automated theorem proving in high-quality software design. In Steffen Hölldobler, editor, *Intellectics and Computational Logic*, volume 19 of *Applied Logic Series*, pages 295–312. Kluwer Academic, 2000. [319]

Sch01. Johann M. Schumann. *Automated Theorem Proving in Software Engineering*. Springer-Verlag, 2001. [319]

SCK+95. B. Steffen, A. Claßen, M. Klein, J. Knoop, and T. Margaria. The fixpoint analysis machine. In J. Lee and S. Smolka, editors, *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR 1995)*, volume 962 of *Lecture Notes in Computer Science*, pages 72–87, Heidelberg, Germany, 1995. Springer-Verlag. [556]

SCS97. Harbhajan Singh, Mirko Conrad, and Sadegh Sadeghipour. Test case design based on Z and the classification-tree method. In Michael G. Hinchey and Shaoying Liu, editors, *Proceedings of the 1st International Conference on Formal Engineering Methods (ICFEM 1997)*, pages 81–90. IEEE Computer Society Press, 1997. [323, 324, 325]

SD85. K. Sabnani and A. Dahbura. A new technique for generating protocol tests. In *Proceedings of the 9th Data Communication Symposium (SIGCOMM 1985)*, pages 36–43. IEEE Computer Society Press, 1985. Also appeared in *Computer Communication Review*, volume 15(4), September 1985. [89]

SD88. Krishan Sabnani and Anton Dahbura. A protocol test generation procedure. *Computer Networks and ISDN Systems*, 15(4):285–297, September 1988. [87, 89, 96, 101, 115, 116]

SDGR03. I. Schieferdecker, Z. R. Dai, J. Grabowski, and A. Rennoch. The UML 2.0 testing profile and its relation to TTCN-3. In D. Hogrefe and A. Wiles, editors, *Proceedings of the 15th IFIP International Conference on Testing of Communicating Systems (TestCom2003)*, volume 2644 of *Lecture Notes in Computer Science*. Springer-Verlag, May 2003. [483]

Seg92. R. Segala. A process algebraic view of Input/Output Automata. Technical Memo MIT/LCS/TR-557, Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, U.S.A., 1992. [187]

Seg96. Roberto Segala. Testing probabilistic automata. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of the 7th Conference on Concurrency Theory (CONCUR 1996)*, volume 1119 of *Lecture Notes in Computer Science*, pages 299–314. Springer-Verlag, 1996. [246, 256, 257, 263, 264, 266, 272, 276, 280]

Seg97. R. Segala. Quiescence, fairness, testing and the notion of implementation. *Information and Computation*, 138(2):194–210, 1997. [194, 196, 197]

SEG00. M. Schmitt, M. Ebner, and J. Grabowski. Test generation with Autolink and Test-Composer. In E. Sherratt, editor, *Proceedings of the 2nd Workshop on SDL and MSC (SAM 2000)*. VERIMAG, IRISA, 2000. [420, 421, 422]

SL88. D. Sidhu and T. Leung. Experience with test generation for real protocols. In *Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM 1988)*, pages 257–261. ACM Press, 1988. [126]

SL89. D. Sidhu and T.-K. Leung. Formal methods for protocol testing: A detailed study. *IEEE Transactions on Software Engineering*, 15(4):413–426, April 1989. [25, 117]

SL94. Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. In Bengt Jonsson and Joachim Parrow, editors, *Proceedings of the 5th Conference on Concurrency Theory (CONCUR 1994)*, volume 836 of *Lecture Notes in Computer Science*, pages 481–496. Springer-Verlag, 1994. [251, 281]

SM93. Sriram Sankar and Manas Mandal. Concurrent runtime monitoring of formally specifi ed programs. *IEEE Computer*, 26(3):32–41, March 1993. [514]

SMIM89. F. Sato, J. Munemori, T. Ideguchi, and T. Mizuno. Test sequence generation method based on fi nite automata – single transition checking using W Set. *Transactions of EIC (in Japanese)*, J72-B-I(3):183–192, 1989. [110]

Sok71. M. N. Sokolovskii. Diagnostic experiments with automata. *Kibernetika*, 6:44–49, 1971. [59]

Sos92. R. Sosič. Dynascope: A tool for program directing. In *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation (PLDI 1992)*, pages 12–21, 1992. Appeared in *SIGPLAN Notices*, volume 27(7), July 1992. [514]

SPHP02. B. Schätz, A. Pretschner, F. Huber, and J. Philipps. Model-based development of embedded systems. In J.-M. Bruel and Z. Bellahsene, editors, *Proceedings of the Workshops of the 8th International Conference on Advances in Object-Oriented Information Systems (OOIS 2002)*, volume 2426 of *Lecture Notes in Computer Science*, pages 298–311. Springer-Verlag, 2002. [16]

Spi92. J. Michael Spivey. *The Z Notation*. Prentice Hall, second edition, 1992. [319, 321]

SSD+03. U. Sammapun, R. Sharykin, M. DeLap, M. Kim, and S. Zdancewic. Formalizing Java-MaC. In O. Sokolsky and M. Viswanathan, editors, *Proccedings of the 3rd Workshop on Run-Time Verifi cation (RV 2003)*, volume 89(2) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2003. [510, 530]

Sta. Statefbw. `http://www.mathworks.com/products/stateflow/`. [23]

Sta66. Peter H. Starke. Eine Bemerkung über homogene Experimente. *Elektronische Informationverarbeitung und Kybernetic*, 2:257–259, 1966. [51]

Sta72. Peter. H. Starke. *Abstract Automata*. North-Holland, Amsterdam, 1972. Translation from German. [29, 36]

Sta73. H. Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, 1973. [9]

Sto02. M. I. A. Stoelinga. An introduction to probabilistic automata. In G. Rozenberg, editor, *EATCS bulletin*, volume 78, pages 176–198, 2002. [245]

SV03. M. I. A. Stoelinga and F. W. Vaandrager. A testing scenario for probabilistic automata. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proceedings of the 30th International Colloquium on Automata, Lnaguages, and Programming (ICALP 2003)*, volume 2719 of *Lecture Notes in Computer Science*, pages 464–477. Springer-Verlag, 2003. Also published as Technical Report of the nijmeegs instituut voor informatica en informatiekunde, number NIII-R0307. [246, 280]

SVD01. Jan Springintveld, Frits Vaandrager, and Pedro R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1–2):225–257, March 2001. [216, 220, 226, 228, 234, 235, 243]

SVG02. S. Schulz and T. Vassiliou-Gioles. Implementation of TTCN-3 test systems using the TRI. In I. Schieferdecker, H. König, and A. Wolisz, editors, *Applications to Internet Technologies and Service – Proceedings of the 14th International Conference on Testing Communication Systems (TestCom 2002)*, volume 210 of *IFIP Conference Proceedings*, pages 425–442. Kluwer Academic, 2002. [467]

SVG03. Ina Schieferdecker and Theofanis Vassiliou-Gioles. Realizing distributed TTCN-3 test systems with TCI. In Dieter Hogrefe and Anthony Wiles, editors, *Proceedings of the 15th International Conference on Testing of Communicating Systems (TestCom 2003)*, volume 2644 of *Lecture Notes in Computer Science*, pages 95–109. Springer-Verlag, 2003. [467]

SVW87. A. P. Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with application to temporal logics. *Theoretical Computer Science*, 49:217–237, 1987. [548]

SW91. Colin Stirling and David Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, October 1991. [548]

TB02. J. Tretmans and E. Brinksma. Côte de Resyste – Automated Model Based Testing. In M. Schweizer, editor, *Progress 2002 – 3rd Workshop on Embedded Systems*, pages 246–255, Utrecht, The Netherlands, October 24 2002. STW Technology Foundation. [410, 413]

Tel. Telelogic website. `http://www.telelogic.com/`. [401]

Tho90. Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 4, pages 133–191. Elsevier Science Publishers, 1990. [217]

TJ98. Kevin S. Templer and Clinton L. Jeffrey. A configurable automatic instrumentation tool for Ansi C. In *Proceedings of the 13th IEEE Conference on Automated Software Engineering (ASE 1998)*, pages 249–258. IEEE Computer Society Press, October 1998. [514]

TM95. Ian Toyn and John A. McDermid. CADiℤ: an architecture for Z tools and its implementation. *Software – Practice and Experience*, 25(3):305–330, 1995. [324]

Tor. Torx website. `http://www.purl.org/net/torx/`. [410]

Toy96. Ian Toyn. Formal reasoning in the Z notation using cadiℤ. In Nicholas A. Merriam, editor, *Proceedings of the 2nd International Workshop on User Interface Design for Theorem Proving Systems (UITP 1996)*, 1996. [324]

Toy98. Ian Toyn. A tactic language for reasoning about Z specifications. In David Duke and Andy Evans, editors, *Proceedings of the 3rd Northern Formal Methods Workshop (NFMW 1998)*, Electronic Workshops in Computing. British Computer Society, 1998. [324]

TP98. Q. M. Tan and A. Petrenko. Test generation for specifications modeled by input/output automata. In A. Petrenko and N. Yevtushenko, editors, *Proceedings of the 11th International Workshop on Testing of Communication Systems (IWTCS 1998)*, volume 131 of *IFIP Conference Proceedings*, pages 83–100. Kluwer Academic, 1998. [188]

TPvB96. Q. M. Tan, Alexandre Petrenko, and Gregor von Bochmann. Modeling basic LOTOS by FSMs for conformance testing. In Piotr Dembinski and Marek Sredniawa, editors, *Proceedings of the 15th IFIP/WG6.1 International Symposium on Protocol Specification, Testing and Verification (PSTV 1995)*, volume 38 of *IFIP Conference Proceedings*, pages 137–152. Chapman & Hall, 1996. [170, 171, 172, 173]

TPvB97. Q. M. Tan, A. Petrenko, and Gregor v. Bochmann. Checking experiments with labeled transition systems for trace equivalence. In *Proceedings of the 10th International Workshop on Testing Communicating Systems (IWTCS 1997)*, 1997. [174, 175, 176, 177]

Tra02. Avraham N. Trakhtman. The existence of synchronizing word and černý conjecture for some finite automata. In *Proceedings of the 2nd Haifa Workshop on Graph Theory, Combinatorics and Algorithms (GTCA 2002)*, June 2002. [51]

Tre94. Jan Tretmans. A formal approach to conformance testing. In *Proceedings of the 6th IFIP TC6/WG6.1 International Workshop on Protocol Test Systems (IWPTS 1993)*, volume C-19 of *IFIP Transactions*, pages 257–276. North-Holland, 1994. [134, 159, 161]

Tre96a. J. Tretmans. Conformance testing with labelled transisition systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29:49–79, 1996. [408, 414]

Tre96b. J. Tretmans. Test generation with inputs, outputs, and quiescence. In T. Margaria and B. Steffen, editors, *Proceedings of the 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1996)*, volume 1055 of *Lecture Notes in Computer Science*, pages 127–146. Springer-Verlag, 1996. [187, 193, 197, 198, 200, 207]

Tre96c. J. Tretmans. Test Generation with Inputs, Outputs and Repetitive Quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996. [189, 191, 408, 410, 411, 413]

U2T. U2TP Consortium. `http://www.fokus.fraunhofer.de/u2tp/`. [483]

U2T04. U2TP Consortium. *UML Testing Profile*, March 2004. Final Adopted Specification at OMG (ptc/2004-04-02). [483, 484]

UMLa. UML 2.0. `http://www.omg.org/uml`. [483]

UMLb. UMLAUT website. `http://www.irisa.fr/UMLAUT/`. [409, 410]

UML03a. *UML 2.0 Infrastructure Specification*, November 2003. OMG Adopted Specification (ptc/03-09-15). [483]

UML03b. *UML 2.0 Superstructure*, September 2003. OMG Adopted Specification (ptc/03-08-02). [483]

Ura92. H. Ural. Formal methods for test sequence generation. *Computer Communications*, 15(5):311–325, June 1992. [18]

UWZ97. Hasan Ural, Xiaolin Wu, and Fan Zhang. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, 46(1):93–99, 1997. [121]

Vaa91. F. Vaandrager. On the relationship between process algebra and Input/Output Automata. In *Proceeedings on the 6th IEEE Symposium on Logic in Computer Science (LICS 1991)*, pages 387–398. IEEE Computer Society Press, 1991. [194]

Val84. L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. [302]

Var96. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. M. Birtwistle, editors, *Logics for Concurrency – Structure versus Automata. Proceedings of the 8th Banff Higher Order Workshop (Banff 1995)*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996. [548, 549]

Var01. M. Y. Vardi. Branching vs. linear time: Final showdown. In W. Yi T. Margaria, editor, *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, January 2001. [544]

Vas73. M. P. Vasilevski. Failure diagnosis of automata. *Cybernetic*, 9(4):653–665, 1973. [118, 124, 573, 578]

VB01. S. A. Vilkomir and J. P. Bowen. Formalization of software testing criteria using the Z notation. In *Proceedings of the 25th International Computer Software and Applications Conference (COMPSAC 2001)*, pages 351–356. IEEE Computer Society Press, 8–12 October 2001. [294, 295]

VB02. S. A. Vilkomir and J. P. Bowen. Reinforced condition/decision coverage (RC/DC): A new criterion for software testing. In D. Bert, J. P. Bowen, M. Henson, and K. Robinson, editors, *Proceedings of the 2nd International Conference of B and Z Users: Formal Specifi cation and Development in Z and B (ZB 2002)*, volume 2272 of *Lecture Notes in Computer Science*, pages 295–313. Springer-Verlag, 2002. [296]

VCI90. S. T. Vuong, W. Y. L. Chan, and M. R. Ito. The UIOv-method for protocol test sequence generation. In *Proceedings of the 2nd International Workshop on Protocol Test Systems (IWPTS 1990)*, pages 161–176. North-Holland, 1990. [116]

vG01. Rob J. van Glabbeek. The linear time – branching time spectrum I: The semantics of concrete, sequential processes. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. Elsevier Science Publishers, 2001. [135, 142, 144, 243]

vGSS95. Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative and stratifi ed models of probabilistic processes. *Information and Computation*, 121:59–80, 1995. [252]

VT01. R. G. de Vries and J. Tretmans. Towards Formal Test Purposes. In E. Brinksma and J. Tretmans, editors, *Proceedings of the 1st International Workshop on Formal Approaches to Testing of Software (FATES 2001)*, number BRICS NS-01-4 in BRICS Notes Series, pages 61–76, 2001. [412, 413]

WBS02. Joachirn Wegener, André Baresel, and Harrnen Sthamer. Suitability of evolutionary algorithms for evolutionary testing. In *Proceedings of the 26th IEEE International Computer Software and Applications Conference: Prolonging Software Life: Development and Redevelopment (COMPSAC 2002)*, pages 287–289, Oxford, England, August 2002. IEEE Computer Society Press. [374]

Weg01. J. Wegener. *Evolutionärer Test des Zeitverhaltens von Realzeit-Systemen*. Dissertation, Humboldt Universität zu Berlin, 2001. [374]

Wes89. C. H. West. Protocol validation in complex systems. In *Proceedings of the ACM Symposium on Communications Architectures & Protocols (SIGCOMM 1989)*, pages 303–312, Austin, TX, September 1989. ACM Press. [411, 425]

Wey86. E. Weyuker. Axiomatizing software test data adequacy. *IEEE Transactions on Software Engineering*, SE-12(12):1128–1138, December 1986. [7]

Wez90. Clazien D. Wezeman. The CO-OP method for compositional derivation of canonical testers. In E. Brinksma, G. Scollo, and C. A. Vissers, editors, *Proceedings of the 9th International Symposium on Protocol Specifi cation, Testing and Verifi cation (PSTV 1990)*, pages 145–158. North-Holland, 1990. [179, 181, 405, 407]

Wez95. Clazien D. Wezeman. Deriving test from LOTOS specifi cations. In Tommaso Bolognesi, Jeroen van de Lagemaat, and Chris Vissers, editors, *LOTOSphere: Software Development with LOTOS*, pages 295–315. Kluwer Academic, 1995. [405, 407]

WGS94. E. Weyuker, T. Goradia, and A. Singh. Automatically generating test data from a boolean specifi cation. *IEEE Transactions on Software Engineering*, 20(5):353–363, May 1994. [21]

Wil01. A. Wiles. ETSI testing activities and the use of TTCN-3. In *Proceedings of the 10th International SDL Forum, 2001*, volume 2078 of *Lecture Notes in Computer Science*, pages 123–128. Springer-Verlag, 2001. [454]

Wol99. Mario Wolczko. Using a tracing java^{TM} virtual machine to gather data on the behavior of java programs. Technical report, Sun Microsystems, March 1999. [514]

WSS94. S.-H. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviors of probabilistic I/O automata. In B. Jonsson and J. Parrow, editors, *Proceedings of the 5th International Conference on Concurrency Theory (CONCUR 1994)*, volume 836 of *Lecture Notes in Computer Science*, pages 513–528, Uppsala, Sweden, August 1994. Springer-Verlag. [246]

WVS83. P. Wolper, M. Y. Vardi, and A. P. Sistla. Reasoning about infi nite computations paths. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science (FOCS 1983)*, pages 185–194. IEEE Computer Society Press, 1983. Extended abstract. [548]

XP. eXtreme Programming website. `http://www.extremeprogramming.org/`. [487]

YL91. M. Yannakakis and D. Lee. Testing fi nite state machines. In Baruch Awerbuch, editor, *Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing (STOC 1991)*, pages 476–485, New Orleans, LS, May 1991. ACM Press. Extended abstract. An extended version appeared in the *Journal of Computer and System Sciences*, 50(2):209–227, April 1995. [115]

ZC93. Jinsong Zhu and Samuel T. Chanson. Fault coverage evaluation of protocol test sequences. Technical Report TR-93-19, Department of Computer Science, University of British Columbia, June 1993. [126]

ZHM97. H. Zhu, P. Hall, and J. May. Software unit test coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427, December 1997. [7, 17, 294, 296, 298]

Index