5 Preorder Relations*

Stefan D. Bruda

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

5.1 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

* 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.

M. Broy et al. (Eds.): Model-Based Testing of Reactive Systems, LNCS 3472, pp. 117-149, 2005.

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 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 5.6 on page 137) 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 5.3 on page 127). The increase in differentiation

[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.

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 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 5.9 on page 145 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 5.3 to 5.8 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 5.9.

5.1.1 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.

5.2 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.

5.2.1 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* $\longrightarrow \subseteq \mathbf{Q} \times \mathsf{Act} \times \mathbf{Q}$.

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 5.4 on page 130 (where states are depicted by nodes, and the relation $\longrightarrow$ 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 5.1. 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 5.2. 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 5.2 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

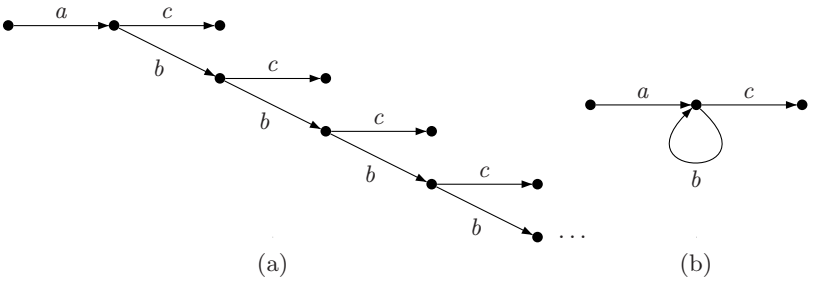

Fig. 5.1. Representation of infinite process trees: an infinite tree (a), and its graph representation (b).

from Figure 5.6 on page 133 (again, states are depicted by nodes, and the relation $\longrightarrow$ 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 $\not\Longrightarrow$. 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 $\longrightarrow$).

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, 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 5.1 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 $(\mathbf{Q}, \mathsf{Act} \cup \{\tau\}, \rightarrow, \uparrow_B)$ with its associated derived transition system $(\mathbf{Q}, \mathsf{Act} \cup \{\tau\}, \Longrightarrow, \uparrow)$, applicable to all the processes under scrutiny; thus a process shall be identified only by its initial state.

5.2.2 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:

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

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 5.3. 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 $\text{obs} : \mathcal{O} \times \mathbf{Q} \to \mathcal{P}$, where $\mathcal{P} \in \{\mathbb{P}_{\text{may}}, \mathbb{P}_{\text{conv}}, \mathbb{P}_{\text{must}}\}$ such that $\text{obs}(o, p)$ 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 $(\mathbf{Q}, \mathsf{Act} \cup \{\tau\}, \rightarrow, \uparrow_B)$ together with its derived transition system $(\mathbf{Q}, \mathsf{Act} \cup \{\varepsilon\}, \Rightarrow, \uparrow)$.

[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. 5.2. Semantics of logical operators on test outcomes.

Definition 5.4. 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 $\mathcal{O}$:

$$o \stackrel{\text{def}}{=} \textsc{Succ} \tag{5.1}$$
$$\mid \textsc{Fail} \tag{5.2}$$
$$\mid a\,o \qquad \text{for } a \in \mathsf{Act} \tag{5.3}$$
$$\mid \widetilde{a}\,o \qquad \text{for } a \in \mathsf{Act} \tag{5.4}$$
$$\mid \varepsilon\,o \tag{5.5}$$
$$\mid o_1 \wedge o_2 \tag{5.6}$$
$$\mid o_1 \vee o_2 \tag{5.7}$$
$$\mid \forall\, o \tag{5.8}$$
$$\mid \exists\, o \tag{5.9}$$

□

Intuitively, Expressions (5.1) and (5.2) 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 (5.3) and (5.4). We can combine tests by means of boolean operators using expressions of form (5.6) and (5.7). By introducing tests of form (5.5) we allow a process to "stabilize" itself through internal actions. Finally, we have universal and existential quantifiers for tests given by (5.8) and (5.9). Nondeterminism is introduced in the tests themselves by the Expressions (5.7) and (5.9), the latter being a generalization of the former.

Definition 5.5. With the semantics of logical operators as defined in Figure 5.2, the function obs inducing the observable testing equivalence, $\text{obs} : \mathcal{O} \times \mathbf{Q} \to \mathbb{P}_{\text{conv}}$, is defined as follows:

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

□

The function from Definition 5.5 follows the syntax of test expressions faithfully, so most cases should need no further explanation. We note that tests of form (5.3) 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 (5.4) 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 5.7 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.

5.2.3 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* $\simeq$.

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

Definition 5.6. 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 5.2.2. 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 $\sqsubseteq_\alpha$.

5.3 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.

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.

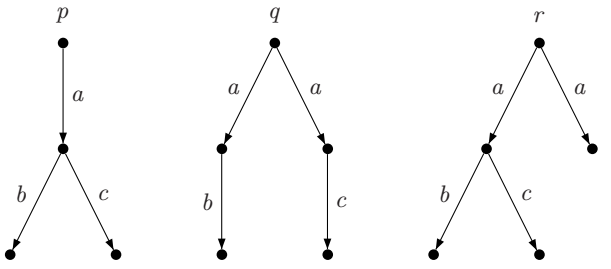

Fig. 5.3. Three sample processes ($p \simeq_{CT} q \simeq_{CT} r$; $q \not\sqsubseteq_B p$).

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 5.7. $p \sqsubseteq_{CT} q$ 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 5.3. 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 5.3 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.

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

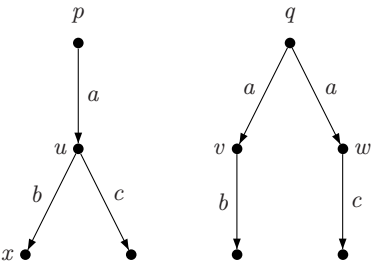

Fig. 5.4. Processes not equivalent under observation preorder ($p \not\simeq_B q$; $p \simeq_{CT} q$; $p \simeq_R q$).

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 5.3, shown again in Figure 5.4 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 5.4 and 5.5 (in Section 5.2.2). 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 (5.1), (5.2), and (5.3).

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 5.4. 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 (5.4) introduces the concept of *refusals*, which allow one to obtain information about the failure of the process to perform some action (as opposed to its ability to perform something). The expressions of form (5.6) and (5.7) 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 (5.8) and (5.9). 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 (5.5). 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 5.8. *With the set $\mathcal{O}$ of tests as defined in the above testing scenario, $p \sqsubseteq_B q$ iff* $\mathrm{obs}(o,p) \subseteq \mathrm{obs}(o,q)$ *for any test $o \in \mathcal{O}$.*

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

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]:

$$\begin{array}{ll} (\top)^* = \textsc{Succ} & (\bot)^* = \textsc{Fail} \\ (\psi \wedge \phi)^* = (\psi)^* \wedge (\phi)^* & (\psi \vee \phi)^* = (\psi)^* \vee (\phi)^* \\ ([a]\psi)^* = \forall\, \widetilde{a}(\psi)^* & (\langle a \rangle \psi)^* = \exists\, a(\psi)^* \\ ([\varepsilon]\psi)^* = \forall\, \varepsilon(\psi)^* & (\langle \varepsilon \rangle \psi)^* = \exists\, \varepsilon(\psi)^* \end{array} \tag{5.10}$$

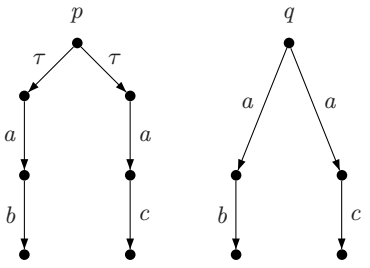

Fig. 5.5. More processes not equivalent under observation preorder ($p \not\simeq_B q$; $p \simeq_{CT} q$; $p \simeq_{\text{must}} q$; $p \simeq_R q$).

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 (5.8) and (5.9) 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 5.4. 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 5.5, which are slight modifications of the processes from Figure 5.4. 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 5.3). Then the test $\widetilde{b}\textsc{Succ}$ introduces a $\top$ outcome in q but not in p according to Definition 5.5; 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 5.6 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

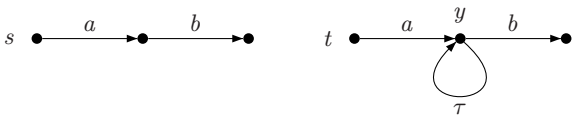

Fig. 5.6. Processes equivalent under observation preorder ($s \simeq_B t$; $s \simeq_R t$; $s \not\simeq_{\text{must}} t$; $s \simeq_{\text{fmust}} t$).

the 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.

5.5 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 5.2.2. 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 (5.3) from Definition 5.4 on page 125 to

$$\text{obs}(ao, p) = \bigcup \{\text{obs}(o, p') \mid p \stackrel{a}{\Longrightarrow} p'\} \cup \{\bot \mid p \uparrow\} \cup \{\bot \mid p \not\Rightarrow\} \tag{5.11}$$

Then we look at two alternative ways to restrict the set of tests $\mathcal{O}$:

(1) Let $\mathcal{O}_{\text{may}}$ be defined only by expressions of form (5.1), (5.3), and (5.5). We do not need any test that signifies failure; instead, failure under test happens

whenever we reach a deadlock, according to Expression (5.11). 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 (5.11) 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 $\mathcal{O}_{\text{must}}$ be defined only by expressions of form (5.1), (5.2), (5.3), and (5.5). 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 $\mathcal{O}$:

$$o = o_1 + o_2 \tag{5.12}$$

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 $\mathcal{O} = \mathcal{O}_{\text{may}} \cup \mathcal{O}_{\text{must}}$.

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

Definition 5.10. 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 5.10 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] $\sqsubseteq_{\text{may}}, \sqsubseteq_{\text{must}}, \sqsubseteq_{\text{conv}} \subseteq \mathbf{Q} \times \mathbf{Q}$:

[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.

(1) $p \sqsubseteq_{\text{may}} q$ if for any $o \in \mathcal{O}_{\text{may}}$, p **may** o implies that q **may** o.
(2) $p \sqsubseteq_{\text{must}} q$ if for any $o \in \mathcal{O}_{\text{must}}$, 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 $\mathcal{O} = \mathcal{O}_{\text{may}} \cup \mathcal{O}_{\text{must}}$. Also note that actually we do not need three sets of observers, since all the three preorders make sense under $\mathcal{O}$. 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 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 5.5 on page 132, 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 5.6 on page 133 should be considered the same. We also argued the other way around, but for now we stick 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 5.7 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.

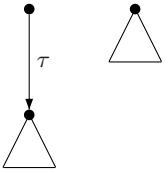

Fig. 5.7. Processes equivalent under $\simeq_{\text{may}}$.

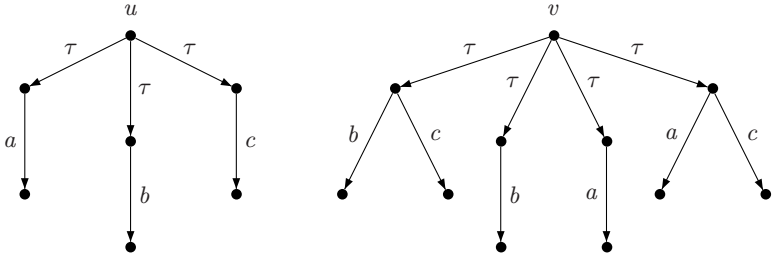

Fig. 5.8. Two processes not equivalent under testing preorder ($u \not\simeq_{\text{must}} v$; $u \simeq_{CT} v$).

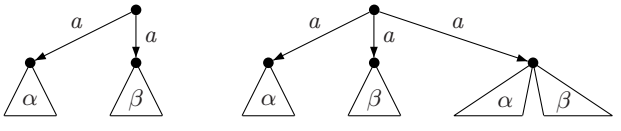

Fig. 5.9. Processes equivalent under any testing preorder.

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 5.6 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 $ab\text{SUCC}$ 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 5.8. No matter what internal move is chosen by v, it can always perform either a or b. It follows that v **must** $(a\text{SUCC} + b\text{SUCC})$. 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** $(a\text{SUCC} + b\text{SUCC})$, but it is not the case that u **must** $(a\text{SUCC} + b\text{SUCC})$. 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

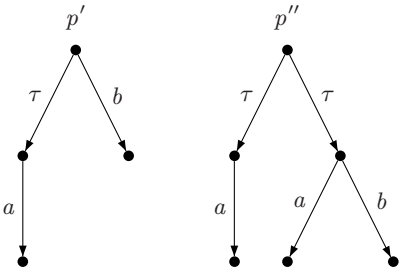

Fig. 5.10. More processes equivalent under testing preorders ($p' \simeq_T p''$; $p' \not\simeq_R p''$).

was shown in our discussion based on Figure 5.5. More generally, the processes depicted in Figure 5.9 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 5.10, 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).

5.6 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 5.2.2 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 $\mathcal{O}$ introduced in Definition 5.4 on page 125 by allowing only expressions of form (5.1)–(5.5), and a restricted variant of (5.12) on page 134 as follows:

$$o = ao_1 + \widetilde{a}o_2 \tag{5.13}$$

The semantics of this kind of expressions is immediately obtained by the semantics of Expressions (5.12) and (5.4) (since we are starting here from the scenario of the testing preorder, the semantics of tests of form (5.4) is given by Expression (5.11)). 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 (5.2), (5.3), and (5.4) to

$$\begin{aligned}
\mathrm{obs}(\textsc{Fail}, p) &= \{0\} \\
\mathrm{obs}(ao, p) &= \bigcup\{\mathrm{obs}(o, p') \mid p \xrightarrow{a} p'\} \cup \{\bot \mid p \uparrow\} \cup \{0 \mid p \not\xrightarrow{a} \} \\
\mathrm{obs}(\widetilde{a}o, p) &= \bigcup\{\mathrm{obs}(o, p) \mid p \not\xrightarrow{a}, p \not\xrightarrow{\tau} \} \cup \{\bot \mid p \uparrow\} \cup \{0 \mid p \xrightarrow{a} \text{ or } p \xrightarrow{\tau} \}
\end{aligned}$$

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 (5.5) 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 $\longrightarrow$ 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 $\mathcal{O}$. 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 $\mathcal{O}_1$ contain exactly all the expressions of form (5.1) and a restricted version of form (5.13) where either $o_1 = \textsc{Fail}$ or $o_2 = \textsc{Fail}$.
Let then p **may** o iff $\top \in \mathrm{obs}(p, o)$.
(2) Let the set $\mathcal{O}_2$ contain exactly all the expressions of form (5.2) and a restricted version of form (5.13) where either $o_1 = \textsc{Succ}$ or $o_2 = \textsc{Succ}$.
Let then p **must** o iff $\{\top\} = \mathrm{obs}(p, o)$.
(3) As usual, put $\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2$.

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 (5.5). Well, this expression was not that "real" 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 $o \in \mathcal{O}_1$, and (b) p **must** o implies q **must** o for any $o \in \mathcal{O}_2$. 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 5.4 on page 130, 5.5 on page 132, and 5.9 on page 136 continue to be equivalent under refusal preorders.

On the other hand, consider the processes shown in Figure 5.10 on page 137 which are equivalent under testing preorder. We then notice that under refusal preorder we have $\mathrm{obs}(b\textsc{Succ}, p') = \{0\}$, 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 $\mathrm{obs}(b\textsc{Succ}, p'') = \{\top, 0\}$. We do not even use refusals here, the two processes become non-equivalent because our convention that test expressions of form (5.5) shall always be performed first.

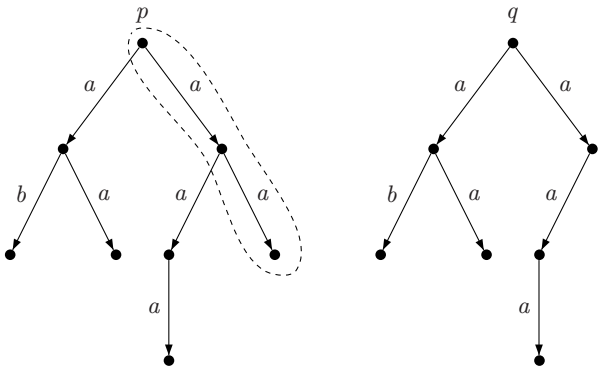

Fig. 5.11. Processes not equivalent under refusal preorder ($p \not\preceq_R q$; $p \simeq_T q$).

Even in the absence of such a convention we have a more precise preorder. Consider for instance the processes from Figure 5.11. They are immediately equivalent under testing preorder, but not so under refusal preorder. Indeed, it holds that $\mathrm{obs}(a\widetilde{b}a\widetilde{a}\textsc{Succ}, p) = \{\top, 0\}$ and $\mathrm{obs}(a\widetilde{b}a\widetilde{a}\textsc{Succ}, q) = \{0\}$ (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 $(\cdot)^* : \mathcal{L}_{HM} \to \mathcal{O}$ translating between expressions in $\mathcal{L}_{HM}$ and tests and given by:

$$\begin{array}{ll} (\top)^* = \textsc{Succ} & (\bot)^* = \textsc{Fail} \\ ([a]\psi)^* = a(\psi)^* & ([a]\psi)^* = \underline{\widetilde{a}}(\psi)^* \\ (\langle\varepsilon\rangle([a]\bot \wedge [\varepsilon]\psi))^* = \widetilde{a}(\psi)^* & ([\varepsilon](\langle a\rangle\top \vee \langle\varepsilon\rangle\psi))^* = \underline{a}(\psi)^* \end{array} \tag{5.14}$$

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

It then follows that:

Theorem 5.12. *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 5.11 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 5.5 on page 132. □

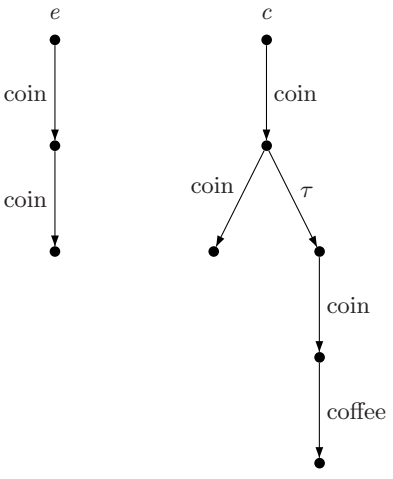

Fig. 5.12. Two vending machines ($e \sqsubseteq_R c$; $e \not\sqsubseteq_{FT} c$).

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 5.6 on page 133 are equivalent under observation preorder (and then according to Proposition 5.12 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 5.13. *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 $\mathcal{O}_1$ and $\mathcal{O}_2$ (and thus in $\mathcal{O}$). The only possible practical downside (of refusal preorder compared with testing preorder) is that we need the ability to block actions.

5.7 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 5.12. 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 $\mathcal{O}$ as we did initially in the previous section, but we revert the semantics of expressions of form (5.4) 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 $o \in \mathcal{O}$ 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 5.12, and consider the test

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

As opposed to refusal testing, we now have $\text{obs}(o, e) = \{0\}$ (the action "coffee" is not available for the test), whereas $\text{obs}(o, c) = \{\top, 0\}$ (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 5.14. *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.

5.8 Fair Testing

Recall the processes depicted in Figure 5.6 on page 133 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 5.6 equivalent, **fair testing** is available [BRV95].

We have the same *testing scenario* for fair testing as the one used in Section 5.5, except that the operator **must** is enhanced, such that it chooses a visible action whenever such an action is available. With the same set $\mathcal{O}$ 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 $o' \in \mathcal{O}$ 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 $a \in \mathsf{Act} \cup \{\text{SUCC}\}$ such that $o' = ao''$, $o'' \in \mathcal{O} \cup \{\varepsilon\}$.

The preorder $\sqsubseteq_{\mathrm{fmust}}$, as well as the equivalence $\simeq_{\mathrm{fmust}}$ 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 $\sqsubseteq_{\mathrm{fmust}}$ in terms of other preorders is easily obtained from the results presented elsewhere [BRV95]:

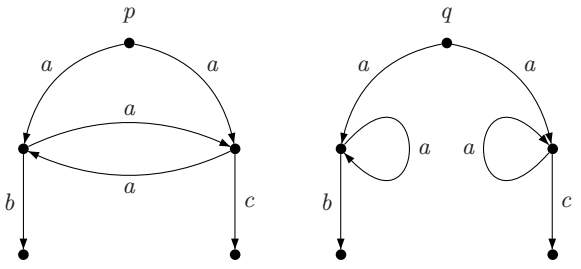

Fig. 5.13. Processes different after hiding $\{a\}$.

Proposition 5.15. *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 $p \xrightarrow{a} q$ for some $a \in A$ are replaced by $p \xrightarrow{\tau}_h q$.

Under a suitable transition system B, consider now the processes depicted in Figure 5.13, 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 5.5, with the exception that the operators **must** and **may** are replaced by the operator **should** defined as follows (again, we have the same set $\mathcal{O}$ of observers as the one used to define the testing preorder):

> p **should** o iff for any $\sigma \in \mathsf{Act}^*$ and $o' \in \mathcal{O}$ with $o = \sigma o'$, it holds that: $\text{obs}(o, p) = \text{obs}(o', p')$ 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 $\top \in \text{obs}(o', p')$.

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 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 5.16. *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 5.17. *The relation $\sqsubseteq_{\text{should}}$ is the largest relation contained in $\sqsubseteq_{\text{fmust}}$ that is a pre-congruence under synchronization and hiding.*

5.9 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**.

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

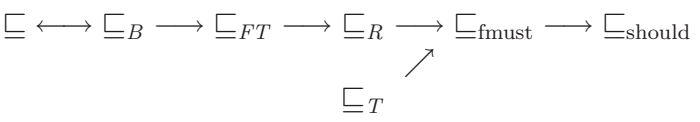

Fig. 5.14. Relations between preorders. The arrows $\sqsubseteq_\alpha \longrightarrow \sqsubseteq_\beta$ stand for "$p \sqsubseteq_\alpha q$ implies $p \sqsubseteq_\beta q$, but not the other way around."

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.

5.10 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 5.2.3
$\sqsubseteq_{CT}$	the complete trace preorder, presented in Section 5.3;
$\sqsubseteq_B$	observation preorder, the subject of Section 5.4;
$\sqsubseteq_T$	(aka $\sqsubseteq_{\text{conv}}$, together with $\sqsubseteq_{\text{may}}$ and $\sqsubseteq_{\text{must}}$), surveyed in Section 5.5;
$\sqsubseteq_R$	refusal preorder, presented in Section 5.6;
$\sqsubseteq_{FT}$	failure trace preorder, in Section 5.7;
$\sqsubseteq_{\text{fmust}}$	fair testing preorder, the subject of Section 5.8;
$\sqsubseteq_{\text{should}}$	should-testing preorder, a variant of $\sqsubseteq_{\text{fmust}}$, also a precongruence.

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 alright. This comparison has been shown throughout the chapter by examples and propositions, and is summarized in Figure 5.14.

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 5.4 that the most discriminating preorder differentiates between processes that are for all practical purposes identical (see for example the processes shown in Figures 5.4 on page 130 and 5.5 on page 132). This is not to say that more differentiation is bad either, just look at the coffee machine examples from Figure 5.12 on page 141, 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.