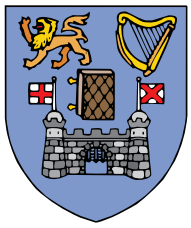

TRINITY COLLEGE DUBLIN
COLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH

Process behaviour: Formulae versus tests

Andrea Cerone and Matthew Hennessy

Computer Science Department Technical Report **TCS-CS-2010-12**
Foundations and Methods Research Group **20th September 2010**

Abstract

Process behaviour is often defined either in terms of the tests they satisfy, or in terms of the logical properties they enjoy. Here we compare these two approaches, using *extensional testing* in the style of DeNicola, Hennessy, and a recursive version of the property logic HML.
We first characterise subsets of the property logic which can be captured by tests. Then we show that those subsets adequately represent the power of tests.

Chapter 1

Introduction

It is very natural to use properties to determine process behaviour; two processes are deemed to be behaviourally equivalent, $p \approx_{\text{prop}} q$ unless there is a property enjoyed by one and not the other. Indeed this is often used as a justification for the use of the well-known *bisimulation equivalence* between processes, [Mil89]. As a property language one can use the modal language commonly referred to as *Hennessy Milner Logic* (HML), which describes the ability of processes to repeatedly interact with each other by performing actions. Then, in an appropriate setting, it can be shown that two processes are *bisimulation equivalent* unless there is some property ϕ such that p enjoys ϕ and q does not, or conversely q enjoys ϕ and not p, [Mil89]; that is the *bisimulation equivalence* coincides with $\approx_{\text{prop}}$.

An alternative approach to process behaviour is based on tests, [DH84]. Intuitively two processes are *testing equivalent*, $p \approx_{\text{test}} q$, relative to a set of tests T if p and q pass exactly the same set of tests from T. Much here depends of course on details, such as the nature of tests, how they are applied and how they succeed. Indeed it has been shown, [Abr87], that if one is sufficiently general with this detail then one can design a scenario in which the property based view $p \approx_{\text{prop}} q$ coincides with the testing view $p \approx_{\text{test}} q$.

A much more restricted view of testing was proposed in [DH84], where observers have very limited ability to manipulate the processes under test; informally processes are conceived as completely independent entities which may or may not react to testing requests; more importantly the application of a test to a process simply consists of a run to completion of the process in a *test harness*. Because processes are in general nondeterministic, formally this leads to two testing based equivalences, $p \approx_{\text{may}} q$ and $p \approx_{\text{must}} q$; the latter is determined by the set of tests a process guarantees to pass, written p *must satisfy* t, while the former by those it is possible to pass, p *may satisfy* t. The *may* equivalence provides a basis for the so-called trace theory of processes [Hoa85] , while the *must* equivalence can be used to justify the various *failures* denotational models used in the theory of CSP, [Hoa85, Old87, DN83].

We take these two different approaches to process behaviour, properties versus tests, for granted. Intuitively the first leads to a branching theory while the latter, in both its variations, leads to a linear theory; see [NV07] for a modern discussion of this dichotomy. Instead the purpose of this paper is to understand more fully the difference

in approach; we investigate the difference in power between the use of properties as expressed in the modal language HML, and the use of tests.

The relationship between properties and tests was first investigated in [AI99] for a recursive version of HML, which we will refer to as *recHML*, for a non-standard notion of testing. Here we revisit this question but this time for the more standard notions of *may* and *must* testing mentioned above.

To explain our results, at least intuitively, let us introduce some informal notation; formal definitions will be given later in the paper. Suppose we have a property ϕ and a test t such that

for every process p, $\quad p$ satisfies ϕ if and only if p *may satisfy* the test t.

Then we say the formula ϕ *may*-represents the test t. We use similar notation with respect to *must* testing. Our first result shows that the power of tests can be captured by properties; for every test t:

(i) there is a formula $\phi_{\text{may}}(t)$ which *may*-represents t; see Theorem 3.2.12,

(ii) there is a formula $\phi_{\text{must}}(t)$ which *must*-represents t; see Theorem 3.1.13.

Properties, or at least those expressed in *recHML*, are more discriminating than tests, and so one would not expect the converse to hold. But we can give simple descriptions of subsets of *recHML*, called *mayHML* and *mustHML* respectively, with the following properties:

(a) every $\phi \in$ *mayHML may*-represents some test $t_{\text{may}}(\phi)$; see Theorem 3.2.9

(b) every $\phi \in$ *mustHML must*-represents some test $t_{\text{must}}(\phi)$; see Theorem 3.1.10

Moreover because the formulae $\phi_{\text{may}}(t)$, $\phi_{\text{must}}(t)$ given in (i), (ii) above are in *mayHML*, *mustHML* respectively, these sub-languages of *recHML* have a pleasing completeness property. For example let ϕ be any formula from *recHML* which can be represented by some *must* test t; that is p satisfies ϕ if and only if p *must satisfy* t. Then up to logical equivalence the formula ϕ is guaranteed to be already in the sub-language *mustHML*; that is there is a formula $\psi \in$ *mustHML* which is logically equivalent to ϕ. The language *mayHML* has a similar completeness property for *may* testing.

We now give a brief overview of the remainder of the paper. In Section 2.1 we recall some basic definitions from concurrency theory. These are required to state our results precisely. In Section 2.2 we present the modal logics that will be used to express properties of concurrent systems. In Section 2.3 we develop two testing frameworks testing frameworks, which are exactly those described in [DH84].
We then set up the formal definition of the question being addressed in the paper in Section 3. In Section 3.1 we analyse such a question when dealing with the must testing relation, while in Section 3.2 we deal with the may case.
Finally, we state our Conclusions in Section 4.
We assume the reader has no previous knowledge in the field; that is, basic definitions are explained in detail, often providing illuminating examples.

Chapter 2

Background

2.1 Modeling Concurrent Systems

The first step that has to be accomplished in order to reason formally about concurrent systems is to provide a mathematical model which allows to give a formal description of their behaviour.

At a descriptive level, we can think of systems as devices which can access different states; for example, if we consider a personal computer the set of states it can access coincides with the set of all its memory configurations. Further, concurrent systems can usually interact with the environment that surrounds them, by performing some kind of activity which can be detected by a component which is external to the system, or by receiving inputs from such a component. In general we can assume there is a set of actions that allows the system to interact with the external environment. We expect that the execution of one of those actions will result in an evolution of the state of the system. If we consider again the personal computer example, then the external environment can be a user typing the name of a program to be executed on the keyboard; when the enter key is pressed, the command will be sent to the computer. On the other hand, the computer will receive the name of the program to be executed and will load the instructions of such a program in its memory, thereby causing an evolution of the system state.

Finally, it is also the case that the state of a system evolves even when there is no interaction with the external environment; in other words, we must take into account the possibility for unobservable activities to be performed by a system. In the computer example above, once the program code has been loaded into the memory, instructions will start to be executed. Each time an instruction is executed, the content of the computer's memory is updated. However, this activity is the result of an internal computation which cannot be directly detected by any user which is interacting with the computer.

This discussion suggests that a possible mathematical description of a concurrent system should include

- its set of states,
- the set of actions it can perform to interact with a component external to the system,

- a special action which denotes unobservable ability
- a description of the evolution of the system states when some action (either observable or unobservable) is performed.

The mathematical model used to represent such information takes the name of *labeled transition system* (LTS).

Definition 2.1.1 (Labeled transition System)**.** *A LTS over a set of actions Act is a triple* $\mathcal{L} = \langle S,\ Act_\tau,\ \longrightarrow \rangle$ *where:*

- S *is a countable set of states*
- $Act_\tau = Act \cup \{\tau\}$ *is a countable set of actions, where* τ *does not occur in Act*
- $\longrightarrow \subseteq S \times Act_\tau \times S$ *is a transition relation.*

The special action τ *denotes unobservable or internal activity.*
We use $a, b, \cdots$ *to range over the set of external actions Act, and* $\alpha, \beta, \cdots$ *to range over* Act_τ*. The standard notation* $s \xrightarrow{\alpha} s'$ *will be used in lieu of* $(s, \alpha, s') \in \longrightarrow$*. States of a LTS* $\mathcal{L}$ *will also be referred to as (term) processes and ranged over by* $s,\ s',\ p,\ q.$ □

First we look at an example of LTS which is standard in all concurrency theory.

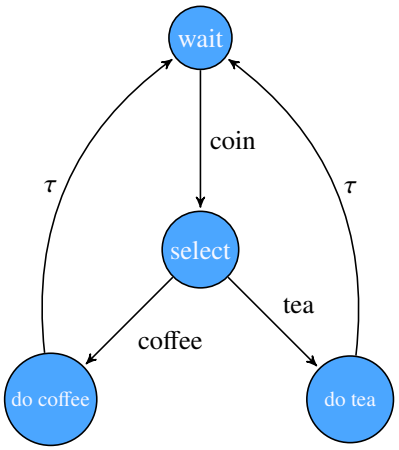

Table 2.1: LTS for the vending machine: graphical representation

Example 2.1.2. *Suppose we want to model a vending machine which can provide a customer either coffee or tea. The vending machine is initially waiting for a customer to insert a coin. When this event occurs, the vending machine enables two selection buttons, respectively for coffee and tea, and waits for the customer to choose one of them. Once the selection button has been pressed, the vending machine will start producing the selected beverage; when this process has finished, the vending machine will perform an unobservable action to return in the initial state.*
The set of states of the vending machine can then be defined as $\{wait, select, do\ coffee, do\ tea\}$*, while the set of external actions it can perform can be defined as* $\{coin, coffee, tea\}$.

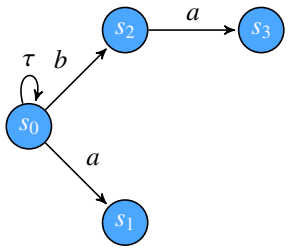

Table 2.2: a very simple LTS

Finally, we can model the behaviour of the vending machine by building the transition relation for the above sets of states and actions. The relation $\longrightarrow$ *for the vending machine is then given by*

$$\begin{aligned} \textit{wait} &\xrightarrow{\textit{coin}} \textit{select} \\ \textit{select} &\xrightarrow{\textit{coffee}} \textit{do coffee} \\ \textit{select} &\xrightarrow{\textit{tea}} \textit{do tea} \\ \textit{do coffee} &\xrightarrow{\tau} \textit{wait} \\ \textit{do tea} &\xrightarrow{\tau} \textit{wait} \end{aligned}$$

□

Often it is useful to give a graphical representation of a LTS; states are represented by balls labeled with the name of the corresponding state. Whenever $p \xrightarrow{\alpha} q$ for some state p, q and action α, we draw a directed arrow labeled with the name of the action α from the ball representing p to the ball representing q. The graphical representation of the LTS for the coffee vending machine illustrated in Example 2.1 is given in Table 2.1.

Let us recall some standard notation associated with LTSs. We write $s \xrightarrow{\alpha}$ if there exists some s' such that $s \xrightarrow{\alpha} s'$, $s \longrightarrow$ if there exists $\alpha \in Act_\tau$ such that $s \xrightarrow{\alpha}$, and $s \not\xrightarrow{\alpha}$, $s \not\longrightarrow$ for their respective negations. We use $\mathrm{Succ}(\alpha, s)$ to denote the set $\{s' | s \xrightarrow{\alpha} s'\}$, and $\mathrm{Succ}(s)$ for $\bigcup_{\alpha \in Act_\tau} \mathrm{Succ}(\alpha, s)$. If $\mathrm{Succ}(s)$ is finite for every state $s \in S$ the LTS is said to be *finite branching*. Finally, a state s diverges, denoted $s \Uparrow$, if there is an infinite path of internal moves

$$s \xrightarrow{\tau} s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_n \xrightarrow{\tau} s_{n+1} \xrightarrow{\tau} \cdots$$

while it converges, denoted $s \Downarrow$, otherwise.

Example 2.1.3. *Consider the LTS depicted in Table 2.2. In this case we have* $s_0 \xrightarrow{a}$, *since* $s_0 \xrightarrow{a} s_1$. *Moreover it holds* $s_0 \xrightarrow{b}$, *as* $s_0 \xrightarrow{b} s_2$. *It is also the case that* $s_0 \longrightarrow$ *for there exists an action* α *(either a or b) such that* $s \xrightarrow{\alpha}$. *For state* s_0 *we find that* $Succ(a, s_0) = \{s_1\}, Succ(b, s_0) = \{s_2\}$, *and thereby* $Succ(s_0) = \{s_1, s_2\}$. *Finally, notice that it is possible to produce an infinite path rooted in* s_0 *whose form is*

$$s_0 \xrightarrow{\tau} s_0 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_0 \xrightarrow{\tau} \cdots$$

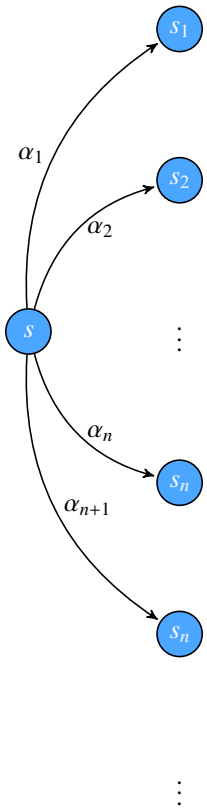

Table 2.3: LTS with a non finite branching state

so that $s_0 \Uparrow$.
If we repeat this procedure for state s_2 *we now find that it is also the case that* $s_2 \xrightarrow{a}$, *as* $s_2 \xrightarrow{a} s_3$; *further we can compute* $Succ(a, s_2)$ *to find out that such a set is exactly* $\{s_3\}$. *However, for state* s_2 *there exists no state* s *such that* $s_2 \xrightarrow{b} s$. *Indeed,* $Succ(b, s_2) = \emptyset$; *in this case we infer that* $s_2 \not\xrightarrow{b}$. *Finally, since* $s_2 \xrightarrow{a}$ *we obtain that* $s_2 \longrightarrow$. *It is trivial to notice that* $s_2 \Downarrow$, *as it cannot perform any internal transition* $\xrightarrow{\tau}$.
Finally, let us look at state s_3. *It is easy to notice that both for actions* a *and* b *we have* $s \not\xrightarrow{a}$ *and* $s \not\xrightarrow{b}$. *Therefore, since there is no action that such a state can perform, we conclude that* $s_3 \not\longrightarrow$. *For such a state we have in fact* $Succ(s_3) = \emptyset$. *Again, it is the case that* $s_3 \Downarrow$. *All the states in the LTS of Table 2.2 have a finite number of derivatives, so that they are all finite branching.* □

Example 2.1.4. *Look at state* s *in picture 2.3. The set of successors of such a state is* $\{s_1, s_2, \cdots, s_n, s_{n+1}, \cdots\}$, *which is countable. Therefore, we have that such a state is not branching finite.* □

When analysing the behaviour of a system by giving its description as a LTS, it is often the case that we are interested in those activities which can be detected by the external environment. This give rise to the standard notation for weak actions $\stackrel{\alpha}{\Longrightarrow}$. Intuitively speaking, if a system performs an unobservable activity which causes it to evolve from

a state s to a state s', and then it performs another unobservable ability which makes it evolve from s' to s'', then the result of these two activities can still be considered as some activity that cannot be detected by the environment. Formally, we say that $s \overset{\tau}{\Longrightarrow} s''$. This procedure applies to arbitrary long sequences of unobservable activities, so that we say that $s \overset{\tau}{\Longrightarrow} s'$ whenever it is the case that $s \overset{\tau}{\longrightarrow}^* s'$, where we recall that $\overset{\tau}{\longrightarrow}^*$ is the reflexive transitive closure of $\overset{\tau}{\longrightarrow}$.

Further, consider the case when a system performs an arbitrary sequence of unobservable activities; then it performs another activity, represented in a LTS by action a, which can be detected by the external environment, and finally it performs another arbitrary sequence of unobservable activities. Again, this can be considered as an unique activity of the system where the only visible action that has been performed is a. Formally, for a given LTS we say that $s \overset{a}{\Longrightarrow} s'$ if and only if there exist s_1, s_2 such that $s \overset{\tau}{\Longrightarrow} s_1 \overset{\tau}{\longrightarrow} s_2 \overset{\tau}{\Longrightarrow} s'$.

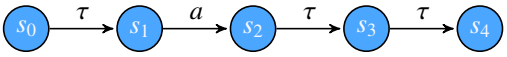

Table 2.4: Another simple LTS

Example 2.1.5. *Look at the LTS depicted in Table 2.4. Since* $s_0 \overset{\tau}{\longrightarrow} s_1$*, we have that* $s_0 \overset{\tau}{\Longrightarrow} s_1$*. Analogously, we obtain that* $s_2 \overset{\tau}{\Longrightarrow} s_4$*, for* $s_2 \overset{\tau}{\longrightarrow} s_3 \overset{\tau}{\longrightarrow} s_4$*. Finally, since* $s_1 \overset{\tau}{\Longrightarrow} s_2 \overset{a}{\longrightarrow} s_2 \overset{\tau}{\Longrightarrow} s_4$ *we obtain* $s_0 \overset{\tau}{\Longrightarrow} s_4$*. A similar procedure shows that* $s_0 \overset{\tau}{\Longrightarrow} s_3$ *also.* □

When $s \overset{\alpha}{\Longrightarrow} s'$ we say that s' is an α-derivative of s. The associated notation $s \overset{\alpha}{\Longrightarrow}$, $s \Longrightarrow$, $s \overset{\alpha}{\not\Longrightarrow}$ and $s \not\Longrightarrow$ have the obvious definitions.

As we are dealing with systems which can communicate with the external environment, it is often the case that we want to analyse the behaviour of a system when it is put in composition with another one. If both of them are represented as LTSs, then we expect to model their composition as a LTS as well. Formally we can define a parallel composition operator as follows:

Definition 2.1.6 (Parallel composition)**.** *Let* $\mathcal{L}_1 = \langle S_1, Act_\tau^1, \longrightarrow \rangle$, $\mathcal{L}_2 = \langle S_2, Act_\tau^2, \longrightarrow \rangle$ *be LTSs. The parallel composition of* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ *is a LTS* $\mathcal{L}_1|\mathcal{L}_2 = \langle S_1 \times S_2, \{\tau\}, \longrightarrow \rangle$*, where* $\longrightarrow$ *is defined by the following SOS rules:*

$$\frac{s \overset{\tau}{\longrightarrow} s'}{s|t \overset{\tau}{\longrightarrow} s'|t} \qquad \frac{t \overset{\tau}{\longrightarrow} t'}{s|t \overset{\tau}{\longrightarrow} s|t'} \qquad \frac{s \overset{a}{\longrightarrow} s' \quad t \overset{a}{\longrightarrow}_2 t'}{s|t \overset{\tau}{\longrightarrow} s'|t'}$$

$s|t$ is used as a conventional notation for (s, t). □

The first two rules models the possibility for each component of a LTS to perform their internal actions independently from the other one. This is needed, as internal activities of a component cannot be detected by the other one. The third rule corresponds to a synchronization between the two components upon performing the same action; such a synchronization will result in an internal activity which cannot be detected by an external environment.

Notice that the parallel composition operator we introduced does not allow any external action for the composition of two LTSs. This is non standard with respect to other definitions of parallel composition that can be found in Concurrency Theory literature; however, this choice will allow a simple presentation of extensional testing, which is covered in Section 2.3.

Example 2.1.7. *Consider again the vending machine whose LTS is depicted in Table 2.1. Suppose a customer wants to interact with the vending machine to obtain a coffee. The customer will then insert a coin into the vending machine, then he will press the coffee button. The LTS that models a customer is straightforward and is depicted in Table 2.5. We can then apply Definition 2.1.6 to obtain the LTS which models the interaction between the vending machine and the customer. The LTS for the new composed system is given in Table 2.6; there w, s and c are used as abbreviations for states wait, select and do coffee respectively.* □

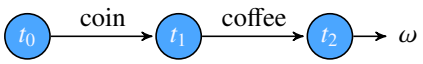

Table 2.5: LTS for a customer of the vending machine

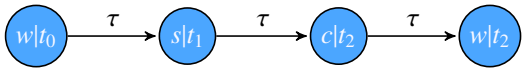

Table 2.6: composition between the vending machine and the customer

2.2 Formalising Properties: Recursive HML

The next topic we address concerns how to express properties of interest for an LTS. To this end, we need to define both a formal language for the formulae which will be used to express properties, and an interpretation function that defines the set of states of a LTS that satisfies a given formula.

The **Hennessy Milner Logic** (HML) [HM85] has proven to be a very expressive property language based on a minimal set of modalities to capture the actions a process can perform, and what the effects of performing such actions are. Here we use a variant in which the interpretation depends on the weak actions of a LTS.

Definition 2.2.1 (Syntax of *recHML*)**.** *Let Var be a countable set of variables. The language recHML is defined as the set of closed formulae generated by the following grammar:*

$$\begin{aligned} \phi \quad ::= \quad & tt \mid ff \mid X \mid Acc(A) \mid \langle\alpha\rangle\phi \mid [\alpha]\phi \mid \\ \mid \quad & \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid min(X,\phi) \mid max(X,\phi) \end{aligned}$$

Here X is chosen from the countable set of variables Var. The operators $min(X,\phi)$, $max(X,\phi)$ act as binders for variables and we have the standard notions of free and bound variables, and associated binding sensitive substitution of formulae for variables. □

Let us recall the informal meaning of *recHML* operators. A formula of the form $\langle\alpha\rangle\phi$ expresses the need for a process to have an α-derivative which satisfies formula ϕ, while formula $[\alpha]\phi$ expresses the need for all α-derivatives (if any) of a converging process to satisfy formula ϕ.
Formula $Acc(A)$ is defined when A is a finite subset of Act, and is satisfied exactly by those converging processes for which each τ derivative has at least an a-derivative for some $a \in Act$. The formulae $min(X, \phi)$ and $max(X, \phi)$ allow the description of recursive properties, respectively being the least and largest solution of the equation $X = \phi$ over the powerset domain of the state space.

Formally, given a LTS $\langle S, Act_\tau, \longrightarrow\rangle$, we interpret each (closed) formula as a subset of 2^S. The set 2^s is a complete lattice and the semantics is determined by interpreting each operator in the language as a monotonic operator over this complete lattice. The binary operators $\vee$, $\wedge$ are interpreted as set theoretic union and intersection respectively while the unary operators are interpreted as follows:

$$\langle\cdot\alpha\cdot\rangle P = \{\ s \mid s \stackrel{\alpha}{\Longrightarrow} s' \text{ for some } s' \in P\ \}$$
$$[\cdot\alpha\cdot]P = \{\ s \mid s \Downarrow, \text{ and } s \stackrel{\alpha}{\Longrightarrow} s' \text{ implies } s' \in P\ \}$$

where P ranges over subsets of 2^S.

Open formulae in *recHML* can be interpreted by specifying, for each variable X, the set of states for which the atomic formula X is satisfied. Such a mapping from Var to 2^S is called environment. Let Env be the set of environments, mappings $\rho : \text{Var} \to 2^S$. A formula ϕ of *recHML* will be interpreted as a function $[\![\ \phi\ ]\!] : Env \to 2^S$. We will use the standard notation $\rho[X \mapsto P]$ to refer to the environment ρ' such that $\rho'(X) = P$ and $\rho'(Y) = \rho(Y)$ for all variables Y such that $X \neq Y$.
The definition of the interpretation $[\![\ \cdot\ ]\!]$ is given in Table 2.7.

The interpretation of a formula $min(X, \phi)$ in the environment ρ is defined as the smallest pre fixpoint of a monotonic functional $\mathcal{F}^\rho_\phi : 2^S \to 2^S$ such that
$\mathcal{F}^\rho_\phi(P) = [\![\ \phi\ ]\!]\rho[X \mapsto P]$. When dealing with closed formulae, Tarski's fixed point Theorem [Win93] ensures that such a set coincides with the least solution of the equation $X = \phi$, as described in our informal explanation of the meaning of *recHML* formulae. A similar argument applies to formulae of the form $max(X, \phi)$, whose interpretation in an environment ρ is defined as the greatest post fixpoint of the monotonic functional considered above. We defer the proof of Tarski's fixed point Theorem until the end of the section, for it is first necessary to prove some simple properties enjoyed by language *recHML*.

When referring to the interpretation of a closed formula $\phi \in$ *recHML*, we will omit the environment application, and sometimes use the standard notation $p \models \phi$ for $p \in [\![\ \phi\ ]\!]$.

Example 2.2.2. *Consider a LTS with a single state p and a unique transition $p \stackrel{b}{\longrightarrow} p$. Let us analyse whether or not state s satisfies the properties $min(X, [a]ff \wedge [b]X)$ and $max(X, [a]ff \wedge [b]X)$.*
To do this, we apply directly the interpretation of recHML formulae given in Table 2.7. For the first formula, consider the empty set $\emptyset$. It is simple to show that

$$
\begin{array}{rcl}
[\![\ \mathrm{tt}\ ]\!]\rho & \triangleq & S \\
[\![\ \mathrm{ff}\ ]\!]\rho & \triangleq & \emptyset \\
[\![\ X\ ]\!]\rho & \triangleq & \rho(X) \\
[\![\ Acc(A)\ ]\!]\rho & \triangleq & \{s | s \Downarrow, s \stackrel{\tau}{\Longrightarrow} s' \text{ implies } \exists a \in A . s' \stackrel{a}{\Longrightarrow}\} \\
[\![\ \langle\alpha\rangle\phi\ ]\!]\rho & \triangleq & \langle\cdot\alpha\cdot\rangle([\![\ \phi\ ]\!]\rho) \\
[\![\ [\alpha]\phi\ ]\!]\rho & \triangleq & [\cdot\alpha\cdot]([\![\ \phi\ ]\!]\rho) \\
[\![\ \phi_1 \vee \phi_2\ ]\!]\rho & \triangleq & [\![\ \phi_1\ ]\!]\rho \cup [\![\ \phi_2\ ]\!]\rho \\
[\![\ \phi_1 \wedge \phi_2\ ]\!]\rho & \triangleq & [\![\ \phi_1\ ]\!]\rho \cap [\![\ \phi_2\ ]\!]\rho \\
[\![\ min(X,\phi)\ ]\!]\rho & \triangleq & \bigcap \{P \mid [\![\ \phi\ ]\!]\rho[X \mapsto P] \subseteq P\} \\
[\![\ max(X,\phi)\ ]\!]\rho & \triangleq & \bigcup \{P \mid P \subseteq [\![\ \phi\ ]\!]\rho[X \mapsto P]\}
\end{array}
$$

Table 2.7: Interpretation of *recHML*

$[\![\ [a]ff \wedge [b]X\ ]\!][X \mapsto \emptyset] \subseteq \emptyset$. *The calculation is carried out below:*

$$
\begin{array}{rcl}
[\![\ [a]ff \wedge [b]X\ ]\!][X \mapsto \emptyset] & = & [\![\ [a]ff\ ]\!][X \mapsto \emptyset] \cap [\![\ [b]X\ ]\!][X \mapsto \emptyset] \\
& = & [\cdot a\cdot]([\![\ ff\ ]\!][X \mapsto \emptyset]) \cap [\cdot b\cdot][\![\ X\ ]\!][X \mapsto \emptyset] \\
& = & [\cdot a\cdot]\emptyset \cap [\cdot b\cdot]\emptyset \\
& = & \{s \in S | s \Downarrow, s \stackrel{a}{\not\Longrightarrow}\} \cap \{s \in S | s \Downarrow, s \stackrel{b}{\not\Longrightarrow}\} \\
& = & \{p\} \cap \emptyset \ = \ \emptyset
\end{array}
$$

Therefore $\emptyset \in \{P \mid [\![\ \phi\ ]\!]\rho[X \mapsto P] \subseteq P\}$*, or equivalently* $[\![\ min(X, [a]ff \wedge [b]X)\ ]\!] \subseteq \emptyset$*. As* $\emptyset$ *is the least element of the complete lattice* $\{\emptyset, \{p\}\}$ *we have that the inclusion above is actually an equality. Thus* $p \not\models min(X, [a]ff \wedge [b]X)$*.*
Next consider formula $max(X, [a]ff \wedge [b]X)$*. In this case we show that* $\{p\} \subseteq [\![\ [a]ff \wedge [b]X\ ]\!][X \mapsto \{p\}]$*, and therefore (being* $\{p\}$ *the greatest element in the complete lattice* $\{\emptyset, \{p\}\}$*) we have that* $[\![\ max(X, [a]ff \wedge [b]X)\ ]\!] = \{p\}$*, i.e.* $p \models max(X, [a]ff \wedge [b]X)$*. Again, the whole calculation is carried out below.*

$$
\begin{array}{rcl}
[\![\ [a]ff \wedge [b]X\ ]\!][X \mapsto \{p\}] & = & [\cdot a\cdot]\emptyset \cap [\cdot b\cdot]\{p\} \\
& = & \{s \in S \mid s \Downarrow, s \stackrel{a}{\not\Longrightarrow}\} \cap \{s \in S \mid s \Downarrow, \forall s' : s \stackrel{b}{\Longrightarrow} s' . \ s' \in \{p\}\} \\
& = & \{p\} \cap \{p\} \\
& = & \{p\}
\end{array}
$$

□

Our version of HML is non-standard, as we have added a convergence requirement for the interpretation of the box operator $[\alpha]$. The intuition here is that, as in the *failures model* of CSP [Hoa85], divergence represents *underdefinedness*. So if a process does not converge all of its capabilities have not yet been determined; therefore one can not quantify over all of its α derivatives, as the totality of this set has not yet been determined.

Further, the operator $Acc(\cdot)$ is also non-standard. It has been introduced for the sake of simplicity, as it will be useful later; in fact it does not add any expressive power to the logic, since for each finite set $A \subseteq Act$ the formula $Acc(A)$ is logically equivalent to

$$[\tau](\bigvee_{a \in A} \langle a \rangle \, \mathrm{tt}\,).$$

As usual, we will write $\phi\{\psi/X\}$ to denote the formula ϕ where all the free occurrences of the variable X are replaced with ψ. We will use the congruence symbol $\equiv$ for syntactic equivalence.

Next, we show some useful properties which relate syntactic substitution in *recHML* formulae with environments. These lemmas are particularly useful when dealing with recursive formula of the form $min(X, \phi)$ and $max(X, \phi)$.

Proposition 2.2.1.

(i) Let ϕ, ψ be formulae such that Y does not occur free in ψ, let ρ be an environment and $P \subseteq 2^S$. Then

$$[\![\, \phi \,]\!]\rho[X \mapsto [\![\, \psi \,]\!]\rho][Y \mapsto P] = [\![\, \phi \,]\!]\rho[Y \mapsto P][X \mapsto [\![\, \psi \,]\!]\rho[Y \mapsto P]\,]$$

(ii) Let $\phi, \psi \in recHML$, and ρ be an environment: then

$$[\![\, \phi\{\psi/X\} \,]\!]\rho = [\![\, \phi \,]\!]\rho[X \mapsto [\![\, \psi \,]\!]\rho].$$

Proof. Both proofs can be performed by induction on the structure of the formula ϕ. For (i) three different sub cases should be handled when dealing with the case $\phi \equiv Z$ (namely $Z \equiv X$; $Z \equiv Y$ and $Z \not\equiv X, Z \not\equiv Y$).
For (ii) we will only outline the details for the case $\phi \equiv min(Y, \phi_1)$: in this case we need to prove

$$[\![\, min(Y, \phi_1)\{\psi/X\} \,]\!]\rho = [\![\, min(Y, \phi_1) \,]\!]\rho[X \mapsto [\![\, \psi \,]\!]\rho].$$

By α-renaming we can choose Y to be a fresh variable, that is $Y \not\equiv X$ and Y does not appear free in ψ.
Since $Y \not\equiv X$ we have that $min(Y, \phi_1)\{\psi/X\} \equiv min(Y, \phi_1\{\psi/X\})$. By inductive hypothesis we have

$$[\![\, \phi_1\{\psi/X\} \,]\!]\rho = [\![\, \phi_1 \,]\!]\rho[X \mapsto [\![\, \psi \,]\!]\rho]$$

and, therefore,

$$\begin{aligned}
[\![\, min(Y, \phi_1\{\psi/X\}) \,]\!]\rho &= \bigcap \{P : [\![\, \phi_1\{\psi/X\} \,]\!]\rho[Y \mapsto P] \subseteq P\} \\
&\overset{\text{IH}}{=} \bigcap \{P : [\![\, \phi_1 \,]\!]\rho[Y \mapsto P][X \mapsto [\![\, \psi \,]\!]\rho[Y \mapsto P]] \subseteq P\} \\
&\overset{(i)}{=} \bigcap \{P : [\![\, \phi_1 \,]\!]\rho[X \mapsto [\![\, \psi \,]\!]\rho][Y \mapsto P] \subseteq P\} \\
&= [\![\, min(Y, \phi_1) \,]\!]\rho[X \mapsto [\![\, \psi \,]\!]\rho],
\end{aligned}$$

where i can be applied as Y does not appear free in ψ. □

The language *recHML* can be extended conservatively by adding simultaneous fixpoints, leading to the language *recHML*$^+$. Given a sequence of variables $(\overline{X})$ of length $n > 0$, and a sequence of formulae $\overline{\phi}$ of the same length, we allow the formula $min_i(\overline{X}, \overline{\phi})$ for $1 \leq i \leq n$, where the only variables allowed to occur in each ϕ_i are those in $(\overline{X})$. This formula will be interpreted as the i-th projection of the simultaneous fixpoint formula.

Definition 2.2.3 (Interpretation of simultaneous fixpoints)**.** *Let $\overline{X}$ and $\overline{\phi}$ respectively be sequences of variables and formulae of length n.*

$$\begin{aligned} [\![min(\overline{X}, \overline{\phi})]\!]\rho &\triangleq \bigcap \{\overline{P} \mid [\![\phi_i]\!]\rho[\overline{X} \mapsto \overline{P}] \subseteq P_i \ \forall 1 \leq i \leq n\} \\ [\![min_i(\overline{X}, \overline{\phi})]\!]\rho &\triangleq \pi_i([\![min(\overline{X}, \overline{\phi})]\!]\rho) \end{aligned}$$

where π_i is the i-th projection operator, and intersection over vectors of sets is defined to be the point wise intersection:

$$\langle P_1, \cdots, P_n \rangle \cap \langle Q_1, \cdots, Q_n \rangle = \langle P_1 \cap Q_1, \cdots, P_n \cap Q_n \rangle$$

□

Intuitively, an interpretation $[\![min(\overline{X}, \overline{\phi})]\!]$, where $\overline{X} = \langle X_1, \cdots, X_n \rangle$ and $\overline{\phi} = \langle \phi_1, \cdots, \phi_n \rangle$, is the least solution (over the set of vectors of length n over 2^S) of the equation system whose form is

$$\begin{aligned} X_1 &= \phi_1 \\ &\vdots \\ X_n &= \phi_n. \end{aligned}$$

If the formula $min(\overline{X}, \overline{\phi})$ is open, then its interpretation in environment ρ, $[\![min(\overline{X}, \overline{\phi})]\!]\rho$, can be thought as the least solution of the system of equations above extended, for every variable Y which appears free in the formula, with an equation of the form $Y = \rho(Y)$. The interpretation of a formula of the form $min_i(\overline{X}, \overline{\phi})$ in environment ρ is the i-th projection of the vector obtained as the least solution of the system of equations above; that is

$$[\![min_i(\overline{X}, \overline{\phi})]\!]\rho = \pi_i([\![min(\overline{X}, \overline{\phi})]\!]\rho).$$

Let $\overline{P} = \langle P_1, \cdots, P_n \rangle$ be the least solution for a system of equations as above. The following theorem states that, for each index i, there exists an equation $X = \psi$ such that its least solution coincides with P_i.

Theorem 2.2.2 (Bekíc)**.**

(i) Let $\overline{X} = \langle X_1, X_2 \rangle$ and $\overline{\phi} = \langle \phi_1, \phi_2 \rangle$. Then, for any environment ρ,

$$\begin{aligned} [\![min_1(\overline{X}, \overline{\phi})]\!]\rho &= [\![min(X_1, \phi_1\{min(X_2, \phi_2)/X_2\})]\!]\rho \\ [\![min_2(\overline{X}, \overline{\phi})]\!]\rho &= [\![min(X_2, \phi_2\{min(X_1, \phi_1)/X_1\})]\!]\rho \end{aligned}$$

(ii) For each formula $\phi \in$ recHML$^+$ there is a formula $\psi \in$ recHML such that $[\![\phi]\!] = [\![\psi]\!]$.

Proof. (i) By straightforward calculations: we will show only the case for $min_1(\overline{X}, \overline{\phi})$, as the other one is obtained by symmetry:

$$\begin{aligned}
&[\![min(X_1, \phi_1\{min(X_2, \phi_2)/X_2\})]\!]\rho && = \\
&\bigcap\{P : [\![\phi_1\{min(X_2, \phi_2)/X_2\}]\!]\rho[X \mapsto P] \subseteq P\} && \overset{2.2.1}{=} \\
&\bigcap\{P : [\![\phi_1]\!]\rho[X_1 \mapsto P][X_2 \mapsto [\![min(X_2, \phi_2)]\!]\rho[X_1 \mapsto P]] \subseteq P\} && = \\
&\bigcap\{P : [\![\phi_1]\!]\rho[X_1 \mapsto P][X_2 \mapsto \bigcap Q : [\![\phi_2]\!]\rho[X_1 \mapsto P][X_2 \mapsto Q] \subseteq Q\] \subseteq P\} && = \\
&\pi_1(\bigcap\{\langle P, Q\rangle : [\![\phi_2]\!]\rho[X_1 \mapsto P][X_2 \mapsto Q] \subseteq Q, [\![\phi_1]\!]\rho[X_1 \mapsto P][X_2 \mapsto Q] \subseteq P\})
\end{aligned}$$

(ii) Let $n \geq 2$, and let $\phi = min_i(\overline{X}, \overline{\phi})$ be a (possibly open) simultaneous fixpoint formula with $\overline{X} = \langle X_1, \cdots, X_n\rangle$ and $\overline{\phi} = \langle \phi_1, \cdots, \phi_n\rangle$.
Without loss of generality, assume $i < n$, as if $i = n$ it is possible to order the vectors of variables and formulae in a consistent way.
Consider the formula

$$\psi = min_i(\langle X_1, \cdots, X_{n-1}\rangle, \langle \phi_1\{min(\phi_n, X_n)/X_n\}, \cdots, \phi_{n-1}\{min(\phi_n, X_n)/X_n\}\rangle),$$

which is a simultaneous fixpoint formula defined over a vector of variables of length $n-1$. In the same style of i it is possible to show that, for any environment ρ, it holds $[\![\phi]\!]\rho = [\![\psi]\!]\rho$. Further, it is straightforward to notice that the free variables of ϕ are the same of ψ. We can therefore iterate this procedure until obtaining a fixpoint formula of the form $min(X, \varphi)$; if the original formula ϕ is closed, and therefore included in *recHML*$^+$, then $min(X, \varphi)$ will also be closed, so that it will belong to *recHML*.

□

The properties of these simultaneous least fixpoints which we will require are summarised in the following theorem:

Theorem 2.2.3 (Fixpoint properties)**.**

(i) Let $(\overline{P})$ be a vector of sets from 2^S satisfying $[\![\phi_i]\!]\rho[\overline{X} \mapsto \overline{P}] \subseteq P_i$ for every $1 \leq i \leq n$. Then $[\![min_i(\overline{X}, \overline{\phi})]\!]\rho \subseteq P_i$

(ii) Given an environment ρ, let ρ_{min} be the environment satisfying $\rho_{min}(X_i) = [\![min_i(\overline{X}, \overline{\phi})]\!]\rho$. Then $[\![min_i(\overline{X}, \overline{\phi})]\!]\rho = [\![\phi_i]\!]\rho_{min}$.

Proof.

(i) This follows from the definition of $[\![min(\overline{X}, \overline{\phi})]\!]$. Let $\overline{P}$ be a vector of sets from 2^S such that
$[\![\phi_i]\!]\rho[\overline{X} \mapsto \overline{P}] \subseteq P_i$. Then

$$\begin{aligned}
[\![min(\overline{X}, \overline{\phi})]\!]\rho &= \bigcap\{\overline{Q} \mid [\![\phi_i]\!]\rho[\overline{X} \mapsto \overline{Q}] \subseteq Q_i,\ 1 \leq i \leq n\} \\
&= \overline{P} \cap \bigcap\{\overline{Q} \mid [\![\phi_i]\!]\rho[\overline{X} \mapsto \overline{Q}] \subseteq Q_i,\ 1 \leq i \leq n\}
\end{aligned}$$

we have therefore that

$$[\![min_i(\overline{X}, \overline{\phi})]\!] = P_i \cap \pi_i(\bigcap\{\overline{Q} \mid [\![\phi_i]\!]\rho[\overline{X} \mapsto \overline{Q}] \subseteq Q_i,\ 1 \leq i \leq n\}) \subseteq Q_i$$

(ii) Let $1 \leq i \leq n$. By the definition of $[\![min_i(\overline{X}, \overline{\phi})]\!]$ it holds

$$\begin{aligned} [\![\phi_i]\!]\rho_{min} &= [\![\phi_i]\!]\rho[\overline{X} \mapsto [\![min(\overline{X}, \overline{\phi})]\!]\rho] \\ &\subseteq [\![min_i(\overline{X}, \overline{\phi})]\!]\rho \end{aligned}$$

The inclusion shows that $[\![\phi_i]\!]\rho_{min} \subseteq [\![min_i(\overline{X}, \overline{\phi})]\!]\rho$. Moreover, since $[\![\phi_i]\!]\rho_{min} \subseteq \rho_{min}$, the converse inclusion follows from (i).

□

Theorem 2.2.3 and Proposition 2.2.1 lead to this useful Corollary which enables us to reason about recursive properties using syntactic substitutions.

Corollary 2.2.4. *Let $\phi \equiv min(X, \psi)$ be a formula in recHML. Then ϕ is logically equivalent to $\psi\{min(X, \psi)/X\}$, that is $[\![\phi]\!] = [\![\psi\{min(X, \psi)/X\}]\!]$.*

Proof. Given a closed formula $\phi \equiv min(X, \psi)$ and an arbitrary environment ρ, we have $[\![min(X, \psi)]\!]\rho = [\![\psi]\!]\rho[X \mapsto [\![min(X, \psi)]\!]]$ by an application of Theorem 2.2.3(ii). Further, $[\![\psi]\!]\rho[X \mapsto [\![min(X, \psi)]\!]] = [\![\psi\{min(X, \psi)/X\}]\!]$ by Proposition 2.2.1(ii). □

We conclude this section by giving a proof of Tarski's Fixpoint Theorem for *recHML*; we consider only formulae of the form $min(X, \phi)$, since we will not deal with greatest fixpoints in what follows. The proof can be easily extended to prove that, given a vector of variables $\overline{X}$ of length n, and a vector of formulae of length $\overline{\varphi}$ of the same length, then formula $min(\overline{X}, \overline{\phi})$ is the least solution of the system of equations $X_i = \phi_i$ for all $1 \leq i \leq n$.

Theorem 2.2.5 ([Win93])**.** *Let $\phi \equiv min(X, \psi)$ a formula in recHML. Then $[\![\phi]\!]$ is the least solution of the equation*

$$X = \psi$$

Proof. Corollary 2.2.4 ensures that $[\![\phi]\!]$ is a solution of the equation $X = \psi$. Moreover, let P be a solution to such an equation; we have

$$[\![\psi]\!][X \mapsto P] = P,$$

therefore $P \in \{P \mid [\![\psi]\!][X \mapsto P] \subseteq P\}$. Now it is trivial to notice $[\![min(X, \psi)]\!] \subseteq P$. □

2.3 Testing Concurrent Systems

Another way to analyse the behaviour of a process is given by testing. Testing a process can be thought as an experiment in which another process, called a test, detects the actions performed by such a process, reacting to them by allowing or forbidding the execution of a subset of observables. After observing the behaviour of the process, the test could decree that it satisfied some property for which it was designed for, thus reporting the success of the experiment through the execution of a special action ω.

Formally speaking, a test is a state from a LTS $\mathcal{T} = \langle T, Act_\tau^\omega, \longrightarrow \rangle$, where $Act_\tau^\omega = Act_\tau \cup \{\omega\}$ and ω is an action not contained in Act_τ.
Given a LTS of processes $\mathcal{L} = \langle S, Act_\tau, \longrightarrow \rangle$, an experiment consists of a pair $p \mid t$ from the product LTS ($\mathcal{L} \mid \mathcal{T}$). We refer to a maximal path of $p \mid t$

$$p \mid t \xrightarrow{\tau} p_1 \mid t_1 \xrightarrow{\tau} \ldots\ldots \xrightarrow{\tau} p_k \mid t_k \xrightarrow{\tau} \ldots$$

as a *computation*; it may be finite or infinite. It is successful if there exists some $n \geq 0$ such that $t_n \xrightarrow{\omega}$. It is important to notice here that a computation is successful it contains a configuration in which the test component can perform a ω action; however, it is not required that such an action has to be actually executed.
As only τ-actions can be performed in a computation, as well as in a computation prefixes, henceforth we will avoid to use the symbol τ in computations.
Computations and successful computations lead to the definition of two well known *testing relations*, [DH84]:

Definition 2.3.1 (May Satisfy, Must Satisfy)**.** *Assuming a LTS of processes and a LTS of tests, let s and t be a state and a test from such LTSs, respectively. We say*

(a) s may satisfy t *if there exists a successful computation for the experiment $s \mid t$.*

(b) s must satisfy t *if each computation of the experiment $s|t$ is successful.* □

Processes can now be compared in terms of the set of test that they may/must pass.
Before continuing our discussion about testing, let us illustrate the ideas behind testing relations with some useful example.

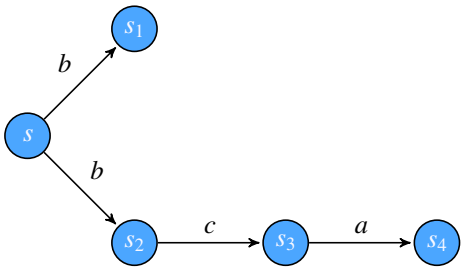

Table 2.8: The tested LTS

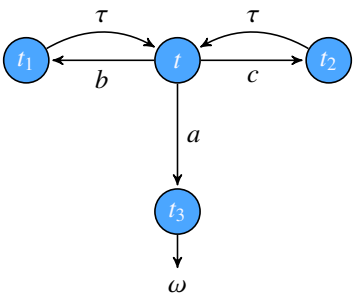

Table 2.9: The test

Example 2.3.2. *Consider the process LTS in Table 2.8 and the test LTS in Table 2.9. We can build the experiment $s \mid t$ to analyse whether the statements*

- s may satisfy t *and*
- s must satisfy t

hold. For the first one, we consider the computation

$$s \mid t \to s_2 \mid t_1 \to s_2 \mid t \to s_3 \mid t_2 \to s_3 \mid t \to s_4 \mid t_3.$$

As $t_3 \xrightarrow{\omega}$ we can conclude that this computation is successful, and hence s may satisfy *t. On the other hand, we can consider the path*

$$s \mid t \to s_1 \mid t_1 \to s_1 \mid t.$$

Such a path is maximal, and therefore it is also a computation. As there is no configuration in such a computation for which the test component can perform an ω action, we can conclude that it is not the case that s must satisfy *t.* □

Later in the paper we will use a specific LTS of tests, whose states are all the closed terms generated by the grammar

$$t \ ::= \ 0 \ \mid \ \alpha.t \ \mid \ \omega.0 \ \mid \ X \ \mid \ t_1 + t_2 \ \mid \ \mu X.t \ . \tag{2.1}$$

Again in this language X is bound in $\mu X.t$, and the test $t\{t'/X\}$ denotes the test t in which each free occurrence of X is replaced by t'. The transition relation defined by the following rules:[1]

$$\frac{}{\alpha.t \xrightarrow{\alpha} t} \qquad \frac{t_1 \xrightarrow{\alpha} t_1'}{t_1 + t_2 \xrightarrow{\alpha} t_1'} \qquad \frac{t_2 \xrightarrow{\alpha} t_2'}{t_1 + t_2 \xrightarrow{\alpha} t_2'} \qquad \frac{}{\mu X.t \xrightarrow{\tau} t\{(\mu X.t)/X\}}$$

The last rule states that a test of the form $\mu X.t$ can always perform a τ-action before evolving in the test $t\{\mu X.t/X\}$. Further, since the transition relation is the smallest relation defined by the inference rule above, it is also the case that this is the only action that a recursive test can perform.
This treatment of recursive processes will allow us to prove properties of paths of recursive tests and experiments by performing an induction on their length.
Further, the following properties hold for a test t in grammar (2.1):

Proposition 2.3.1. *Let $\mathcal{T} = \langle T, Act_\tau, \longrightarrow \rangle$ be the LTS generated by a state t in grammar* (2.1)*: then*

(i) $\mathcal{T}$ is finite branching.

(ii) $\mathcal{T}$ is finite state.

Proof. We prove the two statements separately.

(i) First, notice that every time a test t in grammar (2.1) performs a transition $t \xrightarrow{\alpha} t'$, then t' is itself a closed term of such a grammar.

Further, each closed term of grammar 2.1 can be represented as

$$\sum_{i \in I} t_i$$

where I is finite and each t_i is either in the form 0, $\alpha.t'$ or $\mu X.t'$. Then for each $i \in I$ the number of outgoing transitions $n(t_i)$ of t_i is at most one: we have therefore

$$n(t) \leq \sum_{i \in I} n(t_i) \leq |I|$$

The above argument applies to all states of the generated LTS: hence $\mathcal{T}$ is finite branching.

(ii) A standard proof of this Proposition can be obtained by converting each test into a **Nondeterministic Finite state Tree Automata** [RS].

□

[1] The rules use an abuse of notation, by considering α as an action from $Act_\tau \cup \omega$ rather than from Act_τ.

Henceforth we will always make the assumption that the LTS of tests we consider is branching finite. Further, if also the LTS of processes is also assumed to contain only branching finite states, then the induced LTS of experiments is branching finite as well. It is also ensured that, given an experiment $s \mid t$ in such a LTS and such that *s must satisfy t*, then the maximal length of a successful computation is well defined. To prove this result we will need the following Lemma, which is a variation of Konig's Lemma [BJ89] for directed graphs.

Lemma 2.3.2 (Konig's Lemma for directed graphs)**.** *Let G be a directed graph whose set of vertices is countable. Let a root of G be any node with no incoming edge. Also, assume that G satisfies the following hypothesis:*

- *G has finitely many roots,*
- *each node of G has finite degree,*
- *each node in G is reachable from some root in G.*

Then there is an infinite path in G starting from some root.

Proof. See [KLSV06], Lemma 2.3. □

Theorem 2.3.3. *Let S, T be finite branching LTSs of processes and tests respectively. Let s, t be two states in such LTSs, respectively. Then if s* must satisfy *t the maximal length of a successful computation $|s, t|$ is well defined.*

Proof. Let $\mathcal{E} = \langle E, \{\tau\}, \rightarrow \rangle$ be a finite branching LTS of experiments. For each $e \in E$ we define its *Computation Tree* T_e as the smallest tree whose nodes are (not necessarily all the) elements of $E*$, and whose edges of a node $e_1 \cdots e_n$ are defined as follows: follows:

- if e_n has the form $s \mid t$, with $t \xrightarrow{\omega}$, then node $e_1 \cdots e_n$ has no children,
- otherwise, for each e_{n+1} such that $e_n \xrightarrow{\tau} e_{n+1}$, there is an edge from $e_1 \cdots e_n$ to $e_1 \cdots e_n \cdot e_{n+1}$.

Intuitively speaking, each path of T_e rooted in represents a computation of the experiment e. A more formal definition of T_e can be given as a function of recursive type $\mathcal{T} : N \rightarrow \mathcal{T}$ (see [Cou83] for details).

Suppose now s, t are chosen in finite branching LTSs of processes and tests, respectively. Suppose also *s must satisfy t*. It is straightforward to prove that the LTS of experiments generated by $s \mid t$ is also finite branching. Since *s must satisfy t*, it is the case that all leaves in $T_{s \mid t}$ represent successful computations. In order to prove that the maximal length of a successful computation $|s, t|$ is well defined, we distinguish two different cases:

(i) the number of nodes in $T_{s \mid t}$ is finite. In this case each path between $s \mid t$ and a leaf in $T_{s \mid t}$ has finite length, bounded by the number of nodes in the tree itself; since every path is associated with a successful computation, it follows that $|s, t|$ is bounded by the number of nodes in $T_{s \mid t}$ and therefore is well defined,

(ii) $T_{s \mid t}$ has infinite nodes. Since the LTS generated by $s \mid t$ is finite branching, we have that the degree of each node in the computation tree above is finite. Thus,

by an application of Lemma 2.3.2, we have that $T_{s \mid t}$ contains an infinite path starting from the unique root $s \mid t$ of such a tree; such a path represents an infinite, unsuccessful computation, contradicting the hypothesis *s must satisfy t*.

□

Chapter 3

Testing formulae

Relative to a process LTS $\langle S, Act_\tau, \longrightarrow_S \rangle$ and a test LTS $\langle T, Act_\tau \cup \{\omega\}, \longrightarrow_T \rangle$, we now explore the relationship between tests from our default LTS of tests and formulae of *recHML*. Specifically, given a test t, our goal is to infer a formula ϕ such that the set of processes which *may satisfy/must satisfy* such test is completely characterised by the interpretation $[\![\phi]\!]$. Moreover, we aim to establish exactly the subsets of *recHML* for which each formula can be checked by some test, both in the *may* and *must* case.
For this purpose some definitions are necessary:

Definition 3.0.3. *Let ϕ be a recHML formula and t a test. We say that:*

- ϕ may-*represents*/must-*represents the test t, if for all* $p \in S$, p may satisfy t/p must satisfy t *if and only if* $p \models \phi$.
- ϕ *is* may-*testable*/must-*testable whenever there exists a test which* ϕ may-*represents*/must-*represents.*
- t *is* may-*representable*/must-*representable, if there exists some* $\phi \in$ *recHML which* may-*represents*/must-*represents it respectively.* □

First we present both formulae which are *may*-testable (*must*-testable) and formulae which are not.

Example 3.0.4 (Testable formulae)**.** *In this example we will use tests defined from grammar* (2.1)*. All the examples are handled in an informal manner, as formal details will be covered in a more general way in the remaining of the report.*

(a) Formula $min(X, \langle a \rangle tt \vee \langle b \rangle X)$ *is* may-*testable. A state satisfies such a formula if and only if there exists a finite index* $n \geq 0$ *such that* $s = s_0 \overset{b}{\Longrightarrow} s_1 \overset{b}{\Longrightarrow} \cdots \overset{b}{\Longrightarrow} s_n$ *for some* $s_0, \cdots s_n$ *with* $s_n \overset{a}{\Longrightarrow}$. *We can therefore consider the test* $t \equiv fix(X = \tau.a.\omega.0 + \tau.b.X)$ *If a state s satisfies the above property, then it can synchronise (after a sequence of internal actions performed both by the state itself and by the process) with the test through a b-action; that is, the experiment* $s \mid t$ *can evolve in* $s_1 \mid t$ *after a finite sequence of internal actions. This procedure can be repeated until the configuration* $s_n \mid t$ *is reached. In this case,* s_n *can now synchronise with test t (again after both of them performed some internal steps) through an a-action, thus reaching a successful configuration.*
On the other hand, consider now a state s which not satisfies such a property. That

is, as long as it synchronises with the test through the execution of a b action in a computation of the induced experiment, the resulting state component will never be able to synchronise with the test through the execution of an a action; however this is mandatory for the experiment to reach a successful configuration. Therefore, in this case the experiment $s \mid t$ has no successful computation, and therefore s does not may satisfy *t.*

(b) Formula $min(X, [a]ff \wedge [b]X)$ is must-testable. A process s satisfies this formula if and only if whenever $s = s_0 \stackrel{b}{\Longrightarrow} s_1 \cdots \stackrel{b}{\Longrightarrow} s_n$ for some $n \geq 0$ and states $s_0 \cdots s_n$ with $s_n \stackrel{b}{\not\Longrightarrow}$, it holds that

- *$s_i \Downarrow$ for all $i : 0 \leq i \leq n$,*
- *$s_i \stackrel{a}{\not\Longrightarrow}$ for all $i : 0 \leq i \leq n$,*

Consider the test $t \equiv fix(X = \tau.(a.0 + \tau.\omega.0) + \tau.(b.X + \tau.\omega.0))$, and suppose s satisfies the property above. Consider an arbitrary computation of $s \mid t$; in this case either the test component will perform a series of τ actions, thus reaching a successful computation, or a synchronisation with the test occurs through the execution of a b actions, thus deriving $s \mid t \stackrel{\tau}{\Longrightarrow} s_1 \mid t$. This procedure can be repeated until reaching configuration $s_n \mid t$. As in this case we also have $s_n \stackrel{b}{\not\Longrightarrow}$, the only possibility is to make the test component of the experiment to perform a series of internal actions, thus reaching a successful configuration. In other words, each computation of $s \mid t$ is doomed to reach a configuration where the test component can perform a ω action, and therefore s must satisfy *t. Conversely, suppose s is a process which does not satisfy the property above. That is, either one of the following occurs:*

- *there exists a finite index $n \geq 0$ such that $s = s_0 \stackrel{b}{\Longrightarrow} s_1 \stackrel{b}{\Longrightarrow} \cdot \stackrel{b}{\Longrightarrow} s_n$ with $s_n \Uparrow$,*
- *there exists a finite index $n \geq 0$ such that $s = s_0 \stackrel{b}{\Longrightarrow} s_1 \stackrel{b}{\Longrightarrow} \cdot \stackrel{b}{\Longrightarrow} s_n$ with $s_n \stackrel{a}{\Longrightarrow}$,*
- *s has an infinite path $s = s_0 \stackrel{b}{\Longrightarrow} s_1 \stackrel{b}{\Longrightarrow} \cdots$.*

In the first case we can build an unsuccessful computation by letting the state component of the experiment synchronise with the test through the execution of a b action until configuration $s_n \mid t$ is reached. Then we can obtain an unsuccessful infinite computation by making evolve only the state component of the experiment. In the second case, we can build a computation where the process component synchronise with the test through the execution of a b action until reaching configuration $s_n \mid t$, then, through a series of internal steps and a synchronisation through an a action, we obtain a configuration in which the test component can no longer proceed. This computation is also unsuccessful. Finally, in the third case we can provide an infinite computation in which the state component of the experiment always synchronise with the test component through the execution of a b action; even this computation is not successful. It holds therefore that s does not must satisfy *t.* □

Example 3.0.5 (Negative results)**.**

(a) $\phi = [a]ff$ is not may*-testable.*
Let $s \in [\![[a]ff]\!]$; a new process p can be built starting from s by letting $p \stackrel{\tau}{\longrightarrow} p$,

whenever $s \xrightarrow{\alpha} s'$ *then* $p \xrightarrow{\alpha} s'$.
Processes p and s may satisfy *the same set of tests. However,* $p \notin [\![[a]ff]\!]$, *as* $p \Uparrow$.
Therefore
no test may-*represents* $[a]ff$.

(b) $\phi = \langle a \rangle tt$ *is not* must-*testable.*
We show by contradiction that there exists no test t that must-*represents* ϕ. *To this end, we perform a case analysis on the structure of t.*

- $t \xrightarrow{\omega}$: *Consider the process* 0 *with no transitions. Then* 0 must satisfy *t, whereas* $0 \notin [\![\phi]\!]$.
- $t \not\xrightarrow{\omega}$: *Let* $s \in [\![\phi]\!]$ *and consider the process p built up from s according to the rules of the example above; we have* $p \in [\![\phi]\!]$. *On the other hand, p* must satisfy *t is not true; indeed the experiment* $p \mid t$ *leads to the unsuccessful computation* $p \mid t \rightarrow p \mid t \rightarrow \cdots$.

Therefore there is no test t which must-*represents* ϕ.

(c) $\phi = \langle a \rangle tt \wedge \langle b \rangle tt$ *is not* may-*testable.*
Let s be the process whose only transitions are $s \xrightarrow{a} 0$, $s \xrightarrow{b} 0$. *Let also* p, p' *be the processes whose only transitions are* $p \xrightarrow{a} 0$, $p' \xrightarrow{b} 0$. *We have* $s \in [\![\phi]\!]$, *whereas* $p, p' \notin [\![\phi]\!]$. *We show that whenever s* may satisfy *a test t, then either p* may satisfy *t or* p' may satisfy *t. Thus there exists no test which is* may-*satisfied by exactly those processes in* $[\![\phi]\!]$, *and therefore* ϕ *is not* may-*representable. First, notice that if s* may satisfy *t, then at least one of the following holds:*

(i) $t \xRightarrow{\omega}$,
(ii) $t \xRightarrow{a} t' \xRightarrow{\omega}$,
(iii) $t \xRightarrow{b} t' \xRightarrow{\omega}$.

If $t \xRightarrow{\omega}$, *then trivially both p and* p' may satisfy *t. On the other hand, if* $t \xRightarrow{a} t' \xRightarrow{\omega}$, *then there exist* t'', t_ω *such that* $t \xRightarrow{\tau} t'' \xrightarrow{a} t' \xRightarrow{\tau} t_\omega \xrightarrow{\omega}$. *We can build the computation fragment for* $p \mid t$ *such that*

$$p \mid t \rightarrow \cdots \rightarrow p \mid t'' \rightarrow 0 \mid t' \rightarrow \cdots \rightarrow 0 \mid t_\omega$$

which is successful. Hence p may satisfy *t. Finally, The case* $t \xRightarrow{b} t' \xRightarrow{\omega}$ *is similar.*

(d) In an analogous way of (c) *it can be shown that* $[a]ff \vee [b]ff$ *is not* must-*testable.*
□

We now investigate precisely which formulae in *recHML* can be represented by tests. To this end, we define two sub-languages, namely *mayHML* and *mustHML*.

Definition 3.0.6. *(Representable formulae)*

- *The language mayHML is defined to be the set of closed formulae generated by the following recHML grammar fragment:*

$$\phi \ ::= \ tt \mid ff \mid X \mid \langle \alpha \rangle \phi \mid \phi_1 \vee \phi_2 \mid min(X, \phi) \tag{3.1}$$

- *The language mustHML is defined to be the set of closed formulae generated by the following recHML grammar fragment:*

$$\phi \ ::= \ tt \ \mid \ ff \ \mid \ Acc(A) \ \mid \ X \ \mid \ [\alpha]\phi \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ min(X, \phi) \tag{3.2}$$

□

Note that both sub-languages use the minimal fixpoint operator only; this is not surprising, as informally at least testing is an inductive rather than a co-inductive property. The modality $[\cdot]$ and the conjunction operator $\wedge$ are not allowed in *mayHML*; the above examples show in fact that there exist formulae of the form $[\alpha]\phi$ which are not *may*-testable, and that conjunction of two formulae is not always *may*-testable. The same argument applies to the modality $\langle\cdot\rangle$ and the disjunction operator $\vee$ in the must case, which are therefore not included in *mustHML*.

We have now completed the set of definitions setting up our framework of properties and tests. In the remainder of the paper we prove the results announced, informally, in the Introduction.

3.1 The must case

We will now develop the mathematical basis needed to relate *mustHML* formulae and the *must* testing relation; in this section we will assume that the LTS of processes is branching finite.

First, we prove the following result:

Lemma 3.1.1. *Let $\phi \in$ mustHML, and let $p \in [\![\phi]\!]$, where $p \Uparrow$: then $[\![\phi]\!]$ is the entire process space, i.e. $[\![\phi]\!] = S$.*

Proof. Let p be a process such that $p \Uparrow$, let $\phi \in$ *mustHML* such that $p \in [\![\phi]\!]$. Then ϕ cannot be $Acc(A)$, ff , $[\alpha]\phi$ nor a conjunction of formulae containing one of such terms. We now show that ϕ cannot be a formula of the form $min(X, \psi)$, where ψ contains either free occurrences of the variable X or the operators $Acc(A)$, $[\alpha]$. To this end, we perform a case analysis on the formula ψ:

(i) ψ contains an occurrence of the operator $[\alpha]$. Here we can apply Corollary 2.2.4 to obtain a formula of the form $[\alpha]\phi' \wedge \phi''$ which is logically equivalent to ϕ. Thus, if $p \Uparrow$ then $p \notin [\![\phi]\!]$,

(ii) ψ contains the operator $Acc(A)$. We can proceed as in Case (i),

(iii) ψ contains at least a free occurrence of variable X. If such an occurrence is guarded by a $[\alpha]$ operator, then we can proceed as in Case i. Otherwise we can obtain a formula of the form $min(X, X \wedge \psi')$ which is equivalent to $\phi = min(X, \psi)$. Again, this is done by a repeated application of Corollary 2.2.4. Now it is trivial to notice that $\emptyset$ is a solution to the equation $X = X \wedge \psi$, and therefore it is its least solution. Hence $[\![\phi]\!] = \emptyset$, so that $p \notin [\![\phi]\!]$.

The only possible case left for $p \Uparrow$, $p \in [\![\phi]\!]$ to hold is therefore given by ϕ being generated by the Grammar below:

$$\phi \ ::= \ \mathrm{tt} \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ min(X, \phi). \tag{3.3}$$

It is trivial now to show $[\![\phi]\!] = S$. □

This Lemma has important consequences; it means formulae in *mustHML* either have the trivial interpretation as the full set of states S, or they are only satisfied by convergent states.

Definition 3.1.1. *Let C be the collection of subsets of S determined by:*

- $S \in C$,
- $X \in C, s \in X$ *implies* $s \Downarrow$. □

Proposition 3.1.2. *C ordered by set inclusion is a* continuous partial order, cpo.

Proof. The empty set is obviously the least element in C. So it is sufficient to show that if $X_0 \subseteq X_1 \subseteq \cdots$ is a chain of elements in C then $\bigcup_n X_n$ is also in C. □

We can now take advantage of the fact that *mustHML* actually has a continuous interpretation in $(C, \subseteq)$. The only non trivial case here is the continuity of the operator $[\cdot\alpha\cdot]$:

Proposition 3.1.3. *Suppose the LTS of processes is finite-branching: If $X_0 \subseteq X_1 \subseteq \cdots$ is a chain of elements in C then*

$$\bigcup_n [\cdot\alpha\cdot]X_n = [\cdot\alpha\cdot]\bigcup_n X_n.$$

Proof. It is trivial to show that

$$\bigcup_n [\cdot\alpha\cdot]X_n \subseteq [\cdot\alpha\cdot]\bigcup_n X_n.$$

Thus we only need to show that the opposite implication holds.
First, notice that it $X_i = S$ for some i, then

$$\bigcup_n [\cdot\alpha\cdot]X_n = \{s : \ s \Downarrow\} = [\cdot\alpha\cdot]\bigcup_n X_n$$

Suppose then that $X_i \neq S$ for all $i \geq 0$. Then we have $\bigcup_n X_n \neq S$. By definition the set $[\cdot\alpha\cdot]\bigcup_n X_n$ can be written as

$$\{s \ : \ s \Downarrow, \mathrm{Succ}(\alpha, s) \subseteq \bigcup_n X_n\}.$$

We will prove that for each state s in such a set $\mathrm{Succ}(\alpha, s)$ is finite, therefore there exists an X_n such that $\mathrm{Succ}(\alpha, s) \subseteq X_n$. As a direct consequence, $s \in [\cdot\alpha\cdot]X_n$, which is included in $\bigcup_n [\cdot\alpha\cdot]X_n$.
Let $s \in [\cdot\alpha\cdot]\bigcup_n X_n$ and let s' be one of its α derivative. By definition we have $s' \in \bigcup_n X_n$. Thus there exists $n \geq 0$ such that $s' \in X_n$. Since $X_n \in C$, $X_n \neq S$, it holds $s' \Downarrow$. Since we are assuming that the LTS of processes is finite, as a consequence of Konig's lemma we obtain that if the set $\mathrm{Succ}(\alpha, s)$ is infinite then the τ-computation tree of either s or one of its α-derivative s' has an infinite path. The former contradicts the statement $s \Downarrow$, while the latter contradicts the property $s' \Downarrow$ we just proved. Thus $\mathrm{Succ}(\alpha, s)$ is finite. □

This continuous interpretation of *mustHML* allows us to use chains of finite approximations for these formulae of *mustHML*. That is given $\phi \in$ *mustHML* and $k \geq 0$, recursion free formulae ϕ^k will be defined such that $[\![\phi^k]\!] \subseteq [\![\phi^{(k+1)}]\!]$ and $\bigcup_{k\geq 0} = [\![\phi]\!]$. We can therefore reason inductively on approximations in order to prove properties of recursive formulae.

Definition 3.1.2 (Formulae approximations)**.** *For each formula ϕ in mustHML define*

$$\begin{aligned}
\phi^0 &\triangleq ff \\
\phi^{(k+1)} &\triangleq \phi && \text{if } \phi = tt, ff \text{ or } Acc(A) \\
([\alpha]\phi)^{(k+1)} &\triangleq [\alpha](\phi)^{(k+1)} \\
(\phi_1 \wedge \phi_2)^{(k+1)} &\triangleq \phi_1^{(k+1)} \wedge \phi_2^{(k+1)} \\
(min(X,\phi))^{(k+1)} &\triangleq (\phi\{min(X,\phi)/X\})^k
\end{aligned}$$

□

It is obvious that for every $\phi \in$ *mustHML*, $[\![\phi^k]\!] \subseteq [\![\phi^{(k+1)}]\!]$ for every $k \geq 0$; The fact that the union of the approximations of ϕ converges to ϕ itself depends on the continuity of the interpretation:

Proposition 3.1.4.

$$\bigcup_{k\geq 0} [\![\phi^k]\!] = [\![\phi]\!]$$

Proof. This is true in the initial continuous interpretation of the language, and therefore also in our interpretation. For details see [CN78]. □

Having established these properties of the interpretation of formulae in *mustHML*, we now show that they are all *must*-testable. The required tests are defined by induction on the structure of the formulae.

Definition 3.1.3. *For each (possibly open) formula ϕ in Grammar* (3.2) *define $t_{must}(\phi)$ as follows:*

$$t_{must}(tt) = \omega.0 \tag{3.4}$$

$$t_{must}(ff) = 0 \tag{3.5}$$

$$t_{must}(Acc(A)) = \sum_{a\in A} a.\omega.0 \tag{3.6}$$

$$t_{must}(X) = X \tag{3.7}$$

$$t_{must}([\tau]\phi) = \tau.\, t_{must}(\phi) \tag{3.8}$$

$$t_{must}([a]\phi) = a.\, t_{must}(\phi) + \tau.\omega.0 \tag{3.9}$$

$$t_{must}(\phi_1 \wedge \phi_2) = \begin{cases} \omega.0, & \text{if } \phi_1 \wedge \phi_2 \text{ is closed and logically equivalent to } tt \\ \tau.\, t_{must}(\phi_1) + \tau.\, t_{must}(\phi_2), & \text{otherwise} \end{cases} \tag{3.10}$$

$$t_{must}(min(X,\phi)) = \begin{cases} t_{must}(\phi), & \text{if } \phi \text{ is closed} \\ \mu X.\, t_{must}(\phi), & \text{otherwise} \end{cases} \tag{3.11}$$

□

For each formula ϕ in *mustHML*, the test $t_{\text{must}}(\phi)$ is defined in a way such that the set of processes which *must satisfy* $t_{\text{must}}(\phi)$ is exactly $[\![\phi]\!]$. Before supplying the details of a formal proof of this statement, let us comment on the definition of $t_{\text{must}}(\phi)$.
Cases (3.4), (3.5) and (3.7) are straightforward. In the case of $Acc(A)$, the test allows only those action which are in A to be performed by a process, after which it reports

success.
For the box operator, a distinction has to be made between $[a]\phi$ and $[\tau]\phi$. In the former we have to take into account that a converging process which cannot perform a weak a-action satisfies such a property; thus, synchronisation through the execution of a a-action is allowed, but a possibility for the test to report success after the execution of an internal action is given. In the case of $[\tau]\phi$ no synchronization with any action is required; however, since we are adding a convergence requirement to formula ϕ, we have to avoid the possibility that the test $t_{\text{must}}([\tau]\phi)$ can immediately perform a ω action. This is done by requiring the test $t_{\text{must}}([\tau]\phi)$ to perform only an internal action.
Finally, (3.10) and (3.11) are defined by distinguishing between two cases; this is because a formula of the form $\phi_1 \wedge \phi_2$ or $min(X, \phi)$ can be logically equivalent to tt, whose interpretation is the entire state space. However, the second clause in the definition of $t_{\text{must}}(\phi)$ for such formulae require the test to perform a τ action before performing any other activity, thus at most converging processes *must satisfy* such a test.

In order to give a formal proof that $t_{\text{must}}(\phi)$ does indeed capture the formula ϕ we need to establish some preliminary properties. The first essentially says that that no formula of the form $min(X, \phi)$, with ϕ not closed, will be interpreted in the whole state space.

Lemma 3.1.5. *Let $\phi = min(X, \psi)$, with ψ not closed. Then $[\![\phi]\!] \neq S$.*

Proof. By contradiction. Suppose $[\![min(X, \psi)]\!] = S$; then $min(X, \psi)$ is a term of the grammar (3.3), as shown in the proof of Lemma 3.1.1. That is, formula ψ is necessarily closed. □

Next we state some simple properties about recursive tests.

Lemma 3.1.6.

- p must satisfy $\mu X.t$ *implies* p must satisfy $\mu X.t\{\mu X.t/X\}$.
- $p \Downarrow$, p must satisfy $t[\mu X.t/X]$ *implies* p must satisfy $\mu X.t$.

Proof.

- Suppose p *must satisfy* $\mu X.t$. Then all computations with prefix

  $$p \mid \mu X.t \to p \mid t\{\mu X.t/X\}$$

  are successful; hence p *must satisfy* $t\{\mu X.t/X\}$.

- Suppose $p \Downarrow$, p *must satisfy* $t\{\mu X.t/X\}$. Then for each computation of $p \mid \mu X.t$ with prefix

  $$p \mid \mu X.t \to \cdots \to p' \mid \mu X.t \to p' \mid t\{\mu X.t/X\}$$

  there exists a computation with prefix

  $$p \mid t\{\mu X.t/X\} \to \cdots \to p' \mid t\{\mu X.t/X\}$$

  which is successful. Hence p *must satisfy* $\mu X.t$.

□

Note that the premise $p \Downarrow$ is essential in the second part of this lemma, as $\mu X.t$ cannot perform a ω action; therefore it can be *must*-satisfied only by processes which converge.

Proposition 3.1.7. *Suppose the LTS of processes is finitely branching. If p* must satisfy *$t_{must}(\phi)$ then $p \in [\![\phi]\!]$.*

Proof. Suppose *p must satisfy $t_{\text{must}}(\phi)$*; As both the LTS of processes (by assumption) and the LTS of tests (Proposition 2.3.1) are finite branching, then the LTS generated by $p \mid t$ is finite branching as well. By Theorem 2.3.3 we have that maximal length of a successful computation $|p, t_{\text{must}}(\phi)|$ is defined and finite. Thus it is possible to perform an induction over $|p, t_{\text{must}}(\phi)|$ to prove that $p \in [\![\phi^k]\!]$ for some $k \geq 0$. The result will then follow from Proposition 3.1.4.

- If $|p, t_{\text{must}}(\phi)| = 0$ then $t_{\text{must}}(\phi) \xrightarrow{\omega}$, and hence for each $p \in S$ *p must satisfy* $t_{\text{must}}(\phi)$. Further, by the definition of $t_{\text{must}}(\phi)$ we have that ϕ is logically equivalent to tt, hence $p \in [\![\phi]\!]$.
- If $|p, t_{\text{must}}(\phi)| = n+1$ then the validity of the Theorem follows from an application of an inner induction on ϕ. We show only the most interesting case, which is $\phi = min(X, \psi)$. There are two possible cases.
  (a) If X is not free in ψ then the result follows by the inner induction, as $min(X, \psi)$ is logically equivalent to ψ, and $t_{\text{must}}(min(X, \psi)) \equiv t_{\text{must}}(\psi)$ by definition.
  (b) If X is free in ψ then, by Lemma 3.1.6 *p must satisfy* $t_{\text{must}}(\psi)\{\mu X.\, t_{\text{must}}(\psi)\, /X\}$, which is syntactically equal to $t_{\text{must}}(\psi\{min(X, \psi)/X\})$.
  Since $|p, t_{\text{must}}(\psi\{min(X, \psi)/X\})| < |p, t_{\text{must}}(\phi)|$, by inductive hypothesis we have
  $p \in [\![\psi\{min(X, \psi)/X\}^k]\!]$ for some k, hence $p \in [\![\phi^{(k+1)}]\!]$.

□

To prove the converse of Proposition 3.1.7 we use the following concept:

Definition 3.1.4 (Satisfaction Relation)**.** *Let $R \subseteq S \times$ mustHML and for any ϕ let $(R\ \phi) = \{s \mid s\ R\ \phi\}$ Then R is a satisfaction relation if it satisfies*

$$
\begin{aligned}
(R\ tt) &= S \\
(R\ ff) &= \emptyset \\
(R\ Acc(A)) &= \{\, s \mid s \Downarrow, s \stackrel{\tau}{\Longrightarrow} s' \text{ implies } S(s') \cap A \neq \emptyset \,\} \\
(R\ [\alpha]\phi) &\subseteq [\cdot\alpha\cdot](R\ \phi) \\
(R\ \phi_1 \wedge \phi_2) &\subseteq (R\ \phi_1) \cap (R\ \phi_2) \\
(R\ \phi\{min(X, \phi)/X\}) &\subseteq (R\ min(X, \phi))
\end{aligned}
$$

□

Satisfaction relations are defined to agree with the interpretation $[\![\cdot]\!]$. Indeed, all implications required for satisfaction relations are satisfied by $\models$. Further, as $[\![min(X, \phi)]\!]$ is defined to be the least solution to the recursive equation $X = \phi$, we expect it to be the smallest satisfaction relation.

Proposition 3.1.8. *The relation $\models$ is a satisfaction relation. Further, it is the smallest satisfaction relation.*

Proof. The definition of $[\![\cdot]\!]$ ensures that $\models$ is a satisfaction relation; we have:

$$\begin{aligned}
(\models \mathrm{tt}) &= S \\
(\models \mathrm{ff}) &= \emptyset \\
(\models Acc(A)) &= \{\{\ s \mid s \Downarrow, s \stackrel{\tau}{\Longrightarrow} s' \text{ implies } S(s') \cap A \neq \emptyset\ \} \\
(\models [\alpha]\phi) &= [\cdot\alpha\cdot](\models \phi) \\
(\models \phi\{min(X,\phi)/X\}) &= (\models min(X,\phi))
\end{aligned}$$

where the last equality follows from Corollary 2.2.4.
It remains to show that $\models$ is in fact the smallest satisfaction relation.
Let R be a satisfaction relation, and suppose that $p \in [\![\phi]\!]$: we show that $p\ R\ \phi$.
By Proposition 3.1.4 there exists $k \geq 0$ such that $p \in [\![\phi^k]\!]$. We proceed by induction on k.
The case $k = 0$ is vacuous. Assume the result holds for a generic k; we will perform an inner induction on the structure of ϕ. Again, only the most interesting details are given.
Suppose $\phi = min(X,\psi)$: then $min(X,\psi)^{(k+1)} = (\psi\{\phi/X\})^k$, and by inductive hypothesis $p\ R\ \psi\{\phi/X\}$ follows, and so $p\ R\ \phi$ by the definition of satisfaction relation.
Finally, if ϕ has the form $[\alpha]\psi$ or $\phi_1 \wedge \phi_2$, it is not possible to use the inductive hypothesis directly. This is because $([\alpha]\phi)^{(k+1)} = [\alpha](\phi)^{(k+1)}$, $(\phi_1 \wedge \phi_2)^{(k+1)} = \phi_1(k+1) \wedge \phi_2^{(k+1)}$.
We define therefore the height of a formula $h(\phi)$ as

$$\begin{aligned}
h(\mathrm{tt}) &= 0 \\
h(\mathrm{ff}) &= 0 \\
h(Acc(A)) &= 0 \\
h(min(X,\psi)) &= 0 \\
h([\alpha]\psi) &= h(\psi) + 1 \\
h(\phi_1 \wedge \phi_2) &= \max(h(\phi_1), h(\phi_2)) + 1
\end{aligned}$$

and we perform another induction of $h(\phi)$. The case $h(\phi) = 0$ has already been handled.
Suppose then $h(\phi) = n + 1$; then either $\phi = [\alpha]\psi$ or $\phi = \phi_1 \wedge \phi_2$. We will consider only the first case.Here $h(\psi) = n$, so that by inductive hypothesis we have $p' \models \psi$ implies $p'\ R\ \psi$.
If $p \models [\alpha]\psi$ then $p \Downarrow$; further, whenever $p \stackrel{\alpha}{\Longrightarrow} p'$, we have $p' \models \psi$ and therefore $p'\ R\ \psi$.
Thus $p \in [\cdot\alpha\cdot](R\phi)$. □

This Proposition can be exploited to prove properties for couples (p,ϕ) such that $p \models \phi$, for $\phi \in mustHML$.
Let π be a property over $S \times mustHML$, and suppose the relation $R = \{(s,\phi) \mid \pi(s,\phi)\}$ is a satisfaction relation. We obtain, by Proposition 3.1.8, that $p \models \phi$ implies $\pi(p,\phi)$.
Next we consider the relation R_{must} such that $p\ R_{must}\ \phi$ whenever p *must satisfy* $t_{\mathrm{must}}(\phi)$, and show that it is a satisfaction relation.

Proposition 3.1.9. *The relation R_{must} is a satisfaction relation.*

Proof. We proceed by induction on formula ϕ. Again, we only check the most interesting case.
Suppose $\phi = min(X,\psi)$. We have to show p *must satisfy* $t_{\mathrm{must}}(\psi\{\phi/X\})$ implies p *must satisfy* $t_{\mathrm{must}}(\phi)$.
We distinguish two cases:

(a) X does not appear free in ψ. then $t_{\text{must}}(\phi) = t_{\text{must}}(\psi)$, and $\psi\{\phi/X\} = \psi$. This case is trivial.

(b) X does appear free in ϕ: in this case $t_{\text{must}}(\phi) = \mu X.\, t_{\text{must}}(\psi)$, and $t_{\text{must}}(\psi\{\phi/X\})$ has the form $t_{\text{must}}(\psi)\{\mu X.\, t_{\text{must}}(\psi)\ /X\}$.
By Lemma 3.1.5 $[\![\, \phi \,]\!] \neq S$; therefore Lemma 3.1.1 ensures that $p \Downarrow$, and hence by Lemma 3.1.6 it follows *p must satisfy* $t_{\text{must}}(\phi)$.

□

Combining all these results we now obtain our result on the testability of *mustHML*.

Theorem 3.1.10. *Suppose the LTS of processes is finite-branching. Then for every $\phi \in$ mustHML, there exists a test $t_{must}(\phi)$ such that ϕ* must*-represents the test $t_{must}(\phi)$.*

Proof. We have to show that for any process p, *p must satisfy* $t_{\text{must}}(\phi)$ if and only if $p \in [\![\, \phi \,]\!]$. One direction follows from Proposition 3.1.7. Conversely suppose $p \in [\![\, \phi \,]\!]$. By Proposition 3.1.8 it follows that for all satisfaction relations R it holds $p\ R\ \phi$; hence, by Proposition 3.1.9, $p\ R_{\text{must}}\ \phi$, or equivalently *p must satisfy* $t_{\text{must}}(\phi)$. □

We now turn our attention to the second result, namely that every test t is *must*-representable by some formula in *mustHML*. Let us for the moment assume a branching finite LTS of tests in which the state space T is finite.

Definition 3.1.5. *Assume we have a test-indexed set of variables $\{X_t\}$. For each test $t \in T$ define φ_t as below:*

$$\varphi_t \triangleq tt \qquad \text{if } t \xrightarrow{\omega} \tag{3.12}$$

$$\varphi_t \triangleq ff \qquad \text{if } t \not\xrightarrow{} \tag{3.13}$$

$$\varphi_t \triangleq \Big(\bigwedge_{a,t' : t \xrightarrow{a} t'} [a]X_{t'}\Big) \wedge Acc(\{a | t \xrightarrow{a}\}) \qquad \text{if } t \not\xrightarrow{\omega}, t \not\xrightarrow{\tau}, t \longrightarrow \tag{3.14}$$

$$\varphi_t \triangleq \Big(\bigwedge_{t' : t \xrightarrow{\tau} t'} [\tau]X_{t'}\Big) \wedge \Big(\bigwedge_{a,t' : t \xrightarrow{a} t'} [a]X_{t'}\Big) \qquad \text{if } t \not\xrightarrow{\omega}, t \xrightarrow{\tau} \tag{3.15}$$

Take ϕ_t to be the extended formula $min_t(\overline{X_T}, \overline{\varphi_T})$, using the simultaneous least fixed points introduced in Section 2.2.

Notice that we have a finite set of variables $\{X_t\}$ and that the conjunctions in Definition 3.1.5 are finite, as the LTS of tests is finite state and finite branching. These two conditions are needed therefore for ϕ_t to be well defined.

Formula ϕ_t captures the properties required by a process to *must satisfy* test t. The first two clauses of the definition are straightforward. If t cannot make an internal action or cannot report a success, but can perform a visible action a to evolve in t', then a process should be able to perform a $\stackrel{a}{\Longrightarrow}$ transition and evolve in a process p' such that p' *must satisfy* t'. The requirement $Acc(\{a \mid t \xrightarrow{a}\})$ is needed because a synchronisation between the process p and the test t is required for *p must satisfy t* to be true.
In the last clause, the test t is able to perform at least a τ-action. In this case there is no need for a synchronisation between a process and the test, so there is no term of the form $Acc(\{a \mid t \xrightarrow{a}\})$ in the definition of ϕ_t. However, it is possible that a process p will never synchronise with such test, instead t will perform a transition $t \xrightarrow{\tau} t'$ after

p has executed an arbitrary number of internal actions. Thus, we require that for each transition $p \stackrel{\tau}{\Longrightarrow} p'$, p' *must satisfy* t'.

We now supply the formal details which lead to state that formula ϕ_t characterises the test t. Our immediate aim is to show that the two environments, defined by

$$\begin{aligned} \rho_{min}(X_t) &\triangleq [\![\, \phi_t \,]\!] \\ \rho_{must}(X_t) &\triangleq \{p \mid p \text{ must satisfy } t\} \end{aligned}$$

are identical. This is achieved in the following two propositions.

Proposition 3.1.11. *For all $t \in T$ it holds that $\rho_{min}(X_t) \subseteq \rho_{must}(X_t)$.*

Proof. We just need to show that $[\![\, \varphi_t \,]\!]\rho_{must} \subseteq \rho_{must}(X_t)$: then we can apply the *minimal fixpoint property*, Theorem 2.2.3 (i), to conclude

$$\rho_{min}(X_t) = [\![\, min_t(\overline{X_T}, \overline{\phi_T}) \,]\!] \subseteq \rho_{must}(X_T).$$

The proof is carried out by performing a case analysis on t. We will only consider Case (3.14), as cases (3.12) and (3.13) are trivial and Case (3.15) is handled similarly. Assume $p \in [\![\, \varphi_t \,]\!]\rho_{must}$. We have

(a) $p \Downarrow$,

(b) whenever $p \stackrel{\tau}{\Longrightarrow} p'$ there exists an action $a \in Act$ such that $t \stackrel{a}{\longrightarrow}$ and $p' \stackrel{a}{\Longrightarrow}$,

(c) whenever $p \stackrel{a}{\Longrightarrow} p'$ and $t \stackrel{a}{\longrightarrow} t'$, $p' \in \rho_{must}(X_{t'})$, i.e. p' *must satisfy* t'.

Conditions (a) and (b) are met since $p \in [\![\, Acc(\{a \mid t \stackrel{a}{\longrightarrow}) \,]\!]$ and $t \stackrel{a}{\longrightarrow}$ for some $a \in Act$, while (c) is true because of $p \in [\![\, \bigwedge_{a,t':\, t \stackrel{a}{\longrightarrow} t'} [a] X_{t'} \,]\!]$.

To prove that $p \in \rho_{must}(X_t)$ we have to show that every computation of $p \mid t$ is successful. To this end, consider an arbitrary computation of $p \mid t$; condition (b) ensures that such a computation cannot have the finite form

$$p \mid t \to p_1 \mid t \to \cdots p_k \mid t \to p_{k+1} \mid t \to \cdots \to p_n \mid t \tag{3.16}$$

For such a computation we have that $p_n \stackrel{\tau}{\Longrightarrow} p'$, and there exists p'' with $p' \stackrel{a}{\longrightarrow} p''$ for some action a and test t' such that $t \stackrel{a}{\longrightarrow} t'$. Therefore we have a computation prefix of the form

$$p \mid t \to p_1 \mid t \to \cdots p_n \mid t \to \cdots \to p' \mid t \to p'' \mid t',$$

hence the maximality of computation (3.16) does not hold.

Further, condition (a) ensures that a computation of $p \mid t$ cannot have the form

$$p \mid t \to p_1 \mid t \to \cdots \to p_k \mid t \to p_{k+1} \mid t \to \cdots$$

Therefore all computations of $p \mid t$ have the form

$$p \mid t \to p_1 \mid t \to \cdots \to p_n \mid t \to p' \mid t'$$

with p' *must satisfy* t' by condition (c); then for each computation of $p \mid t$ there exist p'', t'' such that

$$p \mid t \to \cdots \to p' \mid t' \to \cdots \to p'' \mid t'',$$

and $t'' \stackrel{\omega}{\longrightarrow}$. Hence, every computation from $p \mid t$ is successful. □

Proposition 3.1.12. *Assume the LTS of processes is branching finite. For every $t \in T$, $\rho_{must}(X_t) \subseteq \rho_{min}(X_t)$.*

Proof. We have to show *p must satisfy t* implies $p \in [\![\phi_t]\!]$.
Suppose *p must satisfy t*; since we are assuming that the set T, as well as the set S, contains only finite branching tests (processes), That is, the maximal length of a successful computation fragment $|p, t|$ is defined and finite by Theorem 2.3.3.
Recall that $\phi_t = min_t(\overline{X_T}, \overline{\varphi_T})$. We proceed by induction on $k = |p, t|$ to show that *p must satisfy t* implies $p \in [\![\varphi_t]\!]\rho_{min}$; then the result $p \in [\![\phi_t]\!]$ is obtained by applying the Fixpoint Property 2.2.3(ii).

- $k = 0$: In this case, $t \xrightarrow{\omega}$, and hence for all $p \in S$ we have *p must satisfy t*. Moreover, $\varphi_t = \mathrm{tt}$, and hence for all $p \in S$ $p \in [\![\phi_t]\!]\rho_{min}$,
- $k > 0$. There are several cases to consider, according to the structure of the test t:
  1. $t \overset{\omega}{\not\longrightarrow}, t \overset{\tau}{\not\longrightarrow}, t \longrightarrow$: we first show that $p \in [\![Acc(\{a | t \xrightarrow{a})]\!]\rho_{min}$.
     Since *p must satisfy t*, we have $p \Downarrow$. Consider a computation fragment of the form
     $$p \mid t \to \cdots \to p^n \mid t$$
     As $p \Downarrow$, we require that all computations rooted in $p^n \mid t$ will eventually contain a term of the form $p^k \mid t'$, where $t' \neq t$. Further, as $t \overset{\tau}{\not\longrightarrow}$, such a test should follow from a synchronisation between p^{k-1} and t. We have that then that, whenever $p \overset{\tau}{\Longrightarrow} p^n$, there exists an action a such that $t \xrightarrow{a} t'$ and $p^n \overset{a}{\Longrightarrow} p^k$, which combined with the constraint $p \Downarrow$ is equivalent to $p \in [\![Acc(\{a | t \xrightarrow{a})]\!]$.
     We also have to show that $p \in [\![[a]X_{t'}]\!]\rho_{min}$. Let $p \xrightarrow{a} p'$. Then *p must satisfy t* implies *p′ must satisfy t′*. Moreover, we have $|p', t'| < k$. By inductive hypothesis, we have that $p' \in [\![\phi_{t'}]\!]$, that is $p' \in \rho_{min}(X_{t'})$. Then the result $p \in [\![[a]X_{t'}]\!]\rho_{min}$ holds.
  2. $t \overset{\omega}{\not\longrightarrow}, t \xrightarrow{\tau}$: A similar analysis as in the case above can be carried out.

□

Combining these two propositions we get our second result. Let us say that a test t from a LTS of tests $\mathcal{T} = \langle T, Act_\tau^\omega, \to \rangle$ is finitary if the derived LTS consisting of all states in $\mathcal{T}$ accessible from t is finite state and finite branching.

Theorem 3.1.13. *Assuming the LTS of processes is finite branching, every finitary test t is* must*-representable.*

Proof. Consider any test t. We can apply Definition 3.1.5 to the finite LTS of tests reachable from t to obtain a formula ϕ_t which *must*-represents test t. Notice that this formula is not contained in *recHML*, as it uses simultaneous least fixpoints. However, by Theorem 2.2.2 there exists a formula $\phi_{\mathrm{must}}(t) \in$ *recHML* such that $[\![\phi_t]\!] = [\![\phi_{\mathrm{must}}(t)]\!]$, thus t is *must*-representable. Further, since each operator used in Definition 3.1.5 to define φ_t belongs to *mustHML*, it is assured that $\phi_{\mathrm{must}}(t) \in$ *mustHML*. □

As a Corollary we are able to show that *mustHML* is actually the largest language (up to logical equivalences) of *must*-testable formulae.

Corollary 3.1.14. *Suppose ϕ is a formula in recHML which is* must*-testable. Then there exists some ψ in mustHML which is logically equivalent to it.*

Proof. Suppose ϕ is *must*-testable. By Theorem 3.1.10 there exists a finite test $t = t_{\text{must}}(\phi)$ which *must*-represents ϕ. Further, by theorem 3.1.13 there exists a formula $\psi = \phi_{\text{must}}(t) \in$ *mustHML* which *must*-tests for t. Therefore

$$p \in [\![\phi]\!] \Leftrightarrow p \textit{ must satisfy } t_{\text{must}}(\phi) \Leftrightarrow p \in [\![\psi]\!]$$

□

3.2 The may case

We now turn to the characterisation of the *may satisfy* testing relation in terms of *recHML* formulae.
Notice that the nature of the *may satisfy* testing relation is different from that of the *must satisfy* one; here an experiment composed of a process s and a state t is required to have only one successful computation to ensure that s *may satisfy* t holds. As a consequence, when considering the *may satisfy* testing relation, we will not need to reason about all the computations generated by an experiment; in other words, it will be no longer necessary to reason on the maximal length of a successful computation, therefore the assumption that the LTS of processes to be tested contains only finitely branching states can be dropped. However, we still need to assume that the LTS of tests to be considered is finitely branching; informally speaking this is because a test is *may*-represented by a disjunction of formulae, one for each of its branches. Therefore, as we do not allow infinite disjunction in our version of *recHML*, we need to focus only to LTS of finitely branching tests.

First we will prove that each formula in *mayHML may*-represents some test t in grammar (2.1); then we show that if the LTS generated by a test t is finitely branching and finite state, then there exists a formula ϕ which *may*-represents t. In this case we do not require for the LTS of processes to be branching finite.

To prove that the power of tests defined in grammar 2.1 can be captured (with respect to the *may satisfy* testing relation) by the language *mayHML*, we define the concept of *weak satisfaction relation*; this is obtained as the dual version of the weak satisfaction relation relation defined in [AI99].

Definition 3.2.1. *Let $R \subseteq S \times$ mayHML. Then R is a weak satisfaction relation if, and only if, it satisfies the following implications:*

$$\begin{aligned}
(R\ tt) &= S \\
(R\ ff) &= \emptyset \\
(R\ \langle\alpha\rangle\phi) &\supseteq \langle\cdot\alpha\cdot\rangle(R\ \phi) \\
(R\ \phi_1 \vee \phi_2) &\supseteq \langle\cdot\tau\cdot\rangle[(R\ \phi_1) \cup (R\ \phi_2)] \\
(R\ min(X,\phi)) &\supseteq \langle\cdot\tau\cdot\rangle(R\ \phi\{min(X,\phi)/X\})
\end{aligned}$$

□

Informally speaking, given a weak satisfaction relation R, it is possible to determine whether $s \in (R\ \phi)$ for some $s \in S$, $\phi \in$ *mayHML* by looking at the set of the τ-derivatives of s, rather than at the single state itself.

The satisfaction relation $\models$, when restricted to *mayHML*, is a weak satisfaction relation. This is because for any $\phi \in$ *mayHML* we have $[\![\phi]\!] = [\![\langle\tau\rangle\phi]\!]$.

Lemma 3.2.1. *Let $p \in S$, $\phi \in$ mayHML. Then $p \models \phi$ if and only if there exists $p' : p \stackrel{\tau}{\Longrightarrow} p'$ and $p' \models \phi$.*

Proof. For the only if implication notice that for all $p \in S$ it holds $p \stackrel{\tau}{\Longrightarrow} p$.
For the only if implication, notice that the semantics of *mayHML* is defined on weak actions, and that $[\![\langle\alpha\rangle\phi]\!] = [\![\langle\tau\rangle\langle\alpha\rangle\phi]\!]$. □

Proposition 3.2.2. *The relation $\models$ is a weak satisfaction relation.*

Proof. By Lemma 3.2.1 and the definition of $[\![\cdot]\!]$ we have the following implications:

$$\begin{aligned}
(\models \mathrm{tt}) &= S \\
(\models \mathrm{ff}) &= \emptyset \\
(\models \langle\alpha\rangle\phi) &= \langle\cdot\alpha\cdot\rangle(\models \phi) \\
(\models \phi_1 \vee \phi_2) &= (\models \phi_1) \cup \models (\phi_2) \\
&= \langle\cdot\tau\cdot\rangle[(\models \phi_1) \cup (\models \phi_2) \\
(\models min(X, \phi)) &= (\models \phi\{min(X, \phi)/X\}) \\
&= \langle\cdot\tau\cdot\rangle(\models \phi\{min(X, \phi)/X\})
\end{aligned}$$

Corollary 2.2.4 has been applied in the case of a least fixed point formula. □

Further, we have that $\models$ is the smallest weak satisfaction relation. To prove this statement we will use the same techniques used in Section 3.1; that is, first we will show that *mayHML* has a continuous interpretation in the complete lattice $(2^S, \subseteq)$. The only non trivial case here consists in proving the continuity of the $\langle\cdot\tau\cdot\rangle$ operator; this is a direct consequence of the following results, which states that such an operator is distributive over countable sets chosen in 2^S.

Proposition 3.2.3. *Let $P_i, i \in I$ be a countable set of elements in 2^S. Then*

$$\langle\cdot\alpha\cdot\rangle \bigcup_{i\in I} P_i = \bigcup_{i\in I} \langle\cdot\alpha\cdot\rangle P_i$$

Proof. It is trivial to show that

$$\bigcup_{i\in I} \langle\cdot\alpha\cdot\rangle P_i \subseteq \langle\cdot\alpha\cdot\rangle \bigcup_{i\in I} P_i.$$

For the opposite inclusion, suppose $s \in \langle\cdot\alpha\cdot\rangle \bigcup_{i\in I} P_i$; then there exists s' such that $s \stackrel{\alpha}{\Longrightarrow} s', s' \in \bigcup_{i\in I} P_i$. That is, $s' \in P_j$ for some $j \in I$; since $s \stackrel{\alpha}{\Longrightarrow} s'$, by definition $s \in \langle\cdot\alpha\cdot\rangle P_j$, and therefore $s \in \bigcup_{i\in I} \langle\cdot\alpha\cdot\rangle P_i$. □

Given a formula $\phi \in$ *mayHML*, it is possible to define a chain of recursion free formulae $\phi^0, \phi^1, \cdots$ which converge to ϕ itself. This definition is similar in style to that of Definition 3.1.2.

Definition 3.2.2 (Formulae approximations)**.** *For each formula ϕ in mayHML define*

$$
\begin{aligned}
\phi^0 &\triangleq ff \\
tt^{(k+1)} &\triangleq tt \\
ff^{(k+1)} &\triangleq ff \\
(\langle\alpha\rangle\phi)^{(k+1)} &\triangleq \langle\alpha\rangle(\phi)^{(k+1)} \\
(\phi_1 \vee \phi_2)^{(k+1)} &\triangleq \phi_1^{(k+1)} \vee \phi_2^{(k+1)} \\
(min(X,\phi))^{(k+1)} &\triangleq (\phi\{min(X,\phi)/X\})^k
\end{aligned}
$$

□

Proposition 3.2.4.

$$\bigcup_{k\geq 0} [\![\phi^k]\!] = [\![\phi]\!]$$

□

Chains of approximations of formulae in *mayHML* can be exploited to show that $\models$ is indeed the smallest weak satisfaction relation.

Proposition 3.2.5. *Let R be a weak satisfaction relation. Then, for any $s \in S$ and $\phi \in$ mayHML, $s \models \phi$ implies $s\ R\ \phi$.*

Proof. The proof is similar in style to that of Proposition 3.1.8. If $s \models \phi$ then by Corollary 3.2.4 we have that $s \models \phi^k$ for some $k \geq 0$. By performing an induction on k, we show that $s\ R\ \phi$. For $k = 0$ the statement is vacuous; assume then that the statement is true for a generic k, and consider the formula ϕ^{k+1}; we will only check the case $\phi = min(X, \psi)$.
If $s \models (min(X,\psi))^{k+1}$ then by Definition $s \models (\psi\{min(X,\psi)/X\})^k$. By Lemma 3.2.1 $s \models \langle\tau\rangle(\psi\{min(X,\psi)/X\})^k$, which is equivalent to $s \models \langle(\rangle\tau\psi\{min(X,\psi)/X\})^k$. Now, by inductive hypothesis $s\ R\ \langle(\rangle\tau\psi\{min(X,\psi)/X\})$, or equivalently $s \stackrel{\tau}{\Longrightarrow} s'$ with $s'\ R\ (\tau\psi\{min(X,\psi)/X\})$; then by Definition 3.2.1 we have $s\ R\ min(X,\psi)$. □

We are now ready to show that each formula of *mayHML may*-represents some test t. For each formula ϕ in Grammar (3.1), the test $t_{\text{may}}(\phi)$ is defined as below:

$$
\begin{aligned}
t_{\text{may}}(\,\text{tt}\,) &= \omega.0 \\
t_{\text{may}}(\,\text{ff}\,) &= 0 \\
t_{\text{may}}(X) &= X \\
t_{\text{may}}(\phi_1 \vee \phi_2) &= \tau.\, t_{\text{may}}(\phi_1) + \tau.\, t_{\text{may}}(\phi_2) \\
t_{\text{may}}(\langle\alpha\rangle\phi) &= \alpha.\, t_{\text{may}}(\phi) \\
t_{\text{may}}(min(X,\phi)) &= \mu X.\, t_{\text{may}}(\phi)
\end{aligned}
$$

We will need the following property for tests:

Proposition 3.2.6. *Let ϕ, ψ be two formulae in Grammar* (3.1)*, and suppose ψ is a closed formula. Then*

$$t_{may}(\phi)\{t_{may}(\psi)\,/X\} = t_{may}(\phi\{\psi/X\})$$

Proof. By induction on the structure of ϕ. □

Proposition 3.2.7. *The relation* $R_{may} = \{ (s, \phi) \mid s \text{ may satisfy } t_{may}(\phi)\}$ *is a weak satisfaction relation.*

Proof. We prove that R_{may} satisfies the constraints of Definition 3.2.1.

- $t_{\text{may}}(\text{tt}) = \omega.0$. It is trivial to check that each process in S *may satisfy* such a test.
- $t_{\text{may}}(\text{ff}) = 0$. Again, it is straightforward to show that for no process $p \in S$ we have p *may satisfy* $t_{\text{may}}(\text{ff})$.
- Suppose $p \stackrel{a}{\Longrightarrow} p'$, and $p' \; R_{\text{may}} \; \phi$. Then, we have the computation prefix

  $$p \mid \alpha.t_{\text{may}}(\phi) \to \cdots \to p'' \mid \alpha.t_{\text{may}}(\phi) \to p' \mid t_{\text{may}}(\phi)\,[1].$$

  Since p' *may satisfy* $t_{\text{may}}(\phi)$ by the definition of R_{may}, the experiment $p \mid t_{\text{may}}(\langle a\rangle\phi)$ has a successful computation, hence $p \; R_{\text{may}} \; \langle\alpha\rangle\phi$.
- Suppose $p \stackrel{\tau}{\Longrightarrow} p'$, and $p' \; R_{\text{may}} \; \phi_1$. Given an arbitrary formula ϕ_2, consider the experiment
  $p \mid \tau.t_{\text{may}}(\phi_1) + \tau.t_{\text{may}}(\phi_2)$, which has the computation fragment

  $$p \mid \tau.t_{\text{may}}(\phi_1) + \tau.t_{\text{may}}(\phi_2) \to p \mid t_{\text{may}}(\phi_1) \to \cdots \to p' \mid t_{\text{may}}(\phi_1)$$

  As p' *may satisfy* $t_{\text{may}}(\phi_1)$, we have p *may satisfy* $t_{\text{may}}(\phi_1 \vee \phi_2)$.
- Suppose $p \stackrel{\tau}{\Longrightarrow} p'$, with $p' \; R_{\text{may}} \; \psi\{min(X,\psi)/X\}$; we have $t_{\text{may}}(min(X,\psi)) = \mu X.\, t_{\text{may}}(\psi)$. In this case we have the computation

  $$p \mid \mu X.\, t_{\text{may}}(\psi) \to \cdots \to p' \mid \mu X.\, t_{\text{may}}(\psi) \to p' \mid t_{\text{may}}(\psi)\{\mu X.\psi/X\},$$

  where $t_{\text{may}}(\psi)\{\mu X.\psi/X\} = t_{\text{may}}(\psi\{min(X,\psi)/X\})$ by Proposition 3.2.6, and hence $p \; R_{\text{may}} \; min(X,\psi)$.

□

Proposition 3.2.8. *Let* $p \in S$ *and let* $\phi \in$ *mayHML. If* p may satisfy $t_{may}(\phi)$ *then* $p \models \phi$.

Proof. Assume p *may satisfy* $t_{\text{may}}(\phi)$. We proceed by induction on the minimal length of a successful prefix of a computation, denoted $|p, t_{\text{may}}(\phi)|$ with an abuse of notation, to show that $p \models \phi$.

- $|p, t_{\text{may}}(\phi)| = 0$. Then we may infer $t_{\text{may}}(\phi) \stackrel{\omega}{\longrightarrow}$ hence $\phi \equiv \text{tt}$. In this case, for each $p \in S$ it holds. p *may satisfy* $t_{\text{may}}(\phi)$, and $\forall p \in S.p \models \text{tt}$.
- $|p, t_{\text{may}}(\phi)| = k + 1$. Assume the statement holds for k, and consider the prefix

  $$p|\, t_{\text{may}}(\phi) \to p'|t'$$

  of a minimal successful computation.
  We distinguish several cases:

[1] where $p'' = p'$ in the case $\alpha = \tau$

(a) $p \xrightarrow{\tau} p', t' \equiv t_{\text{may}}(\phi)$. Then by inductive hypothesis $p' \models \phi$, and by Lemma 3.2.1 we have $p \models \phi$.

(b) $p = p', t_{\text{may}}(\phi) \xrightarrow{\tau} t'$: in this case there are tree possibilities.

- $\phi = min(X, \psi)$ for some ψ. Hence $t' \equiv t_{\text{may}}(\psi)\{t_{\text{may}}(\phi)\,/X\}$, which is $t' \equiv t_{\text{may}}(\psi\{\phi/X\})$. Again, by induction we have $p \models \psi\{\phi/X\}$, and hence $p \models \phi$.
- $\phi = \phi_1 \vee \phi_2$. Without loss of generality we may infer $t' \equiv t_{\text{may}}(\phi_1)$. By Inductive hypothesis we have $p \models \phi_1$, hence $p \models \phi_1 \vee \phi_2$.
- $\phi = \langle\tau\rangle\psi$ for some ψ. In this case we have $t' = t_{\text{may}}(\psi)$; by the inductive hypothesis it holds $p \models \psi$. Therefore, by Lemma 3.2.1 $p \models \langle\tau\rangle\psi$.

(c) $p \xrightarrow{a} p', t_{\text{may}}(\phi) \xrightarrow{a} t'$. In this case we have $\phi = \langle\alpha\rangle\psi$, and hence $t' \equiv t_{\text{may}}(\psi)$. Then, by using the inductive hypothesis again, we have $p \models \langle a\rangle\phi$.

□

Theorem 3.2.9. *Let $\phi \in$ mayHML, $p \in S$. Then $p \models \phi$ if and only if p* may satisfy *$t_{may}(\phi)$.*

Proof. Analogous to the proof of Theorem 3.1.10 □

Next, we show that if the LTS of tests generated by a test is finite state, then each test t is *may*-represented by a *mayHML* formula $\phi_{\text{may}}(t)$.
First, assume to have a test indexed set of test variables $\{X_t\}$. Then, for each test t define the formula ϕ_t as

$$\begin{array}{lcll} \varphi_t & = & \text{tt} & \text{if } t \xrightarrow{\omega} \\ \varphi_t & = & \text{ff} & \text{if } t \not\xrightarrow{} \\ \varphi_t & = & \bigvee_{\alpha, t' : t \xrightarrow{\alpha} t'} \langle\alpha\rangle X_{t'} & \text{if } t \not\xrightarrow{\omega}, t \longrightarrow \end{array}$$

and take $\phi_{\text{may}}(t)$ to be the *recHML*$^+$ formula $\min_t(\overline{X_T}, \overline{\varphi_T})$.
Next we define the following environments:

$$\begin{array}{lcl} \rho_{min}(X_t) & = & [\![\, \phi_{\text{may}}(t) \,]\!] \\ \rho_{may}(X_t) & = & \{p \mid p \textit{ may satisfy } t\} \end{array}$$

In the same style as Section 3.1, we will prove that the two environments above coincide.

Proposition 3.2.10. *For each test $t, \rho_{min}(X_t) \subseteq \rho_{may}(X_t)$.*

Proof. Suppose the LTS generated by a test t is finite state and finite branching. We just need to show that $[\![\, \varphi_t \,]\!]\rho_{must} \subseteq \rho_{must}(X_t)$: then we can apply the *minimal fixpoint property*, Theorem 2.2.3 (i), to conclude

$$\rho_{min}(X_t) = [\![\, min_t(\overline{X_T}, \overline{\phi_T}) \,]\!] \subseteq \rho_{must}(X_T).$$

The proof is carried out by performing a case analysis on t.

- $t \xrightarrow{\omega}$. In this case we have $\rho_{may}(X_t) = S$, so the statement trivially holds.

- $t \not\longrightarrow$. W have $\phi_t = \mathrm{ff}$, hence $[\![\phi_t]\!]\rho_{may} = \emptyset$. Again, the statement is trivial.

- $t \stackrel{\omega}{\not\longrightarrow}, t\longrightarrow$. Suppose $p \in [\![\phi]\!]\rho_{may}$. We have that there exists at least one action α such that $t \stackrel{\alpha}{\longrightarrow} t'$; thus there exists a process p' such that $p \stackrel{\alpha}{\Longrightarrow} p'$ and p' *may satisfy* t' (in the case $\alpha = \tau$ choose $p' = p$). Hence we have the computation fragment

$$p \mid t \to \cdots \to p'' \mid t \to p' \mid t',$$

  so that p *may satisfy* t.

□

Proposition 3.2.11. *For each test* $t, \rho_{may}(X_t) \subseteq \rho_{min}(X_t)$.

Proof. Again, assume the LTS generated by a test t is finite state. Let p be a process such that p *may satisfy* t. We proceed by induction on the minimal length of a successful computation prefix $|p, t|$ to show that p *may satisfy* t implies $p \in [\![\varphi_t]\!]\rho_{min}$; then the result $p \in [\![\phi_t]\!]$ is obtained by applying the Fixpoint Property 2.2.3(ii).

- $|p, t| = 0$. In this case we have $t\stackrel{\omega}{\longrightarrow}$. By definition, $\phi_t = \mathrm{tt}$, so that we have $[\![\phi_t]\!]\rho_{min} = S$. This case is trivial.

- $|p, t| > 0$. Let

$$p \mid t \to p' \mid t' \to \cdots \to p_n \mid t_n$$

  be a successful computation prefix of length $|p, t|$. We distinguish several cases according to the structure of the computation. Since p' *may satisfy* t' and $|p', t'| < |p, t|$, in each case we have $p' \in [\![\varphi_{t'}]\!]\rho_{min}$ by inductive hypothesis.

  - $p = p'$, $t \stackrel{\tau}{\longrightarrow} t'$; we have $p \in [\![X_t]\!]\rho_{min} = [\![\langle\tau\rangle\phi_{t'}]\!]\rho_{min}$. Then $p \in [\![\bigvee_{\alpha, t' : t \stackrel{\alpha}{\longrightarrow} t'} \langle\tau\rangle X_{t'}]\!]\rho_{min}$.
  - $p \stackrel{\tau}{\longrightarrow} p'$, $t = t'$; we have $p' \in [\![X_t]\!]\rho_{min}$, and therefore $p \in [\![X_t]\!]\rho_{min}$ by Lemma 3.2.1.
  - $p \stackrel{a}{\longrightarrow} p'$, $t \stackrel{a}{\longrightarrow} t'$; in this case $p \in [\![\langle a\rangle X_{t'}]\!]\rho_{min}$, and hence $p \in [\![\phi_t]\!]\rho_{min}$.

□

Propositions 3.2.10 and 3.2.11 can be combined to obtain the following result:

Theorem 3.2.12. *Every finitary test* t *is* may*-representable.* □

Corollary 3.2.13. *Suppose* ϕ *is a formula in recHML which is* may*-testable. Then there exists some* ψ *in mayHML which is logically equivalent to it.*

Proof. Suppose ϕ is *may*-testable. By theorem 3.2.9 there exists a finite test $t = t_{\mathrm{may}}(\phi)$ which *may*-represents ϕ. Further, by Theorem 3.2.12 there exists a formula $\psi = \phi_{\mathrm{may}}(t) \in$ *mayHML* which *may*-tests for t. Therefore

$$p \in [\![\phi]\!] \Leftrightarrow p \textit{ may satisfy } t_{\mathrm{may}}(\phi) \Leftrightarrow p \in [\![\psi]\!]$$

□

Chapter 4

Conclusions

We have investigated the relationship between properties of processes as expressed in a recursive version of Hennessy-Milner logic, *recHML*, and *extensional* tests as defined in [DH84]. In particular we have shown that both *may* and *must* tests can be captured in the logic, and we have isolated logically complete sub-languages of *recHML* which can be captured by *may* testing and *must* testing. One consequence of these results is that the *may* and *must* testing preorders of [DH84] are determined by the logical properties in these sub-languages *mayHML* and *mustHML* respectively.

However these results come at the price of modifying the satisfaction relation; to satisfy a box formula a process is required to converge. One consequence of this change is that the language *recHML* no longer characterises the standard notion of *weak bisimulation equivalence*, as this equivalence is insensitive to divergence. But there are variations on *bisimulation equivalence* which do take divergence into account; see for example [Wal88, HP80].

The research reported here was initiated after reading [AI99]; there a notion of testing was used which is different from both *may* and *must* testing. They define *s passes* the test *t* whenever no computation from $s \mid t$ can perform the success action ω, and give a sub-language which characterises this form of testing. It is easy to check that *s passes t* if and only if, in our terminology, *s may t* is not true. So their notion of testing is dual to *may* testing, and therefore, not surprisingly, our results on *may* testing are simply dual versions of theirs.

We have concentrated on properties associated with essentially two behavioural theories, *weak bisimulation equivalence* and *testing*. However there are a large number of other behavioural theories; see [Gla93] for an extensive survey, including their characterisation in terms of *observational* properties.

Bibliography

[Abr87] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987.

[AI99] Luca Aceto and Anna Ingólfsdóttir. Testing hennessy-milner logic with recursion. In Thomas [Tho99], pages 41–55.

[BJ89] George S. Boolos and Richard C. Jeffrey. *Computability and Logic*. Cambridge University Press, third edition, 1989.

[BRR87] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Applications and Relationships to Other Models of Concurrency*. Number 255 in Lecture Notes in Computer Science. Springer-Verlag, 1987.

[CN78] Bruno Courcelle and Maurice Nivat. The algebraic semantics of recursive program schemes. In Winkowski [Win78], pages 16–30.

[Cou83] Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25(2):95–169, March 1983. Fundamental study.

[DH84] R. DeNicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 24:83–113, 1984.

[DN83] Rocco De Nicola. A Complete Set of Axioms for a Theory of Communicating Sequential Processes. In *FCT*, pages 115–126, 1983.

[Gla93] Rob J. van Glabbeek. The linear time - branching time spectrum ii. In *CONCUR '93: Proceedings of the 4th International Conference on Concurrency Theory*, pages 66–81, London, UK, 1993. Springer-Verlag.

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

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

[HP80] Matthew C. B. Hennessy and Gordon D. Plotkin. A term model for CCS. In *Mathematical Foundations of Computer Science 1980, Proceedings of the 9th Symposium*, volume 88, pages 261–274, Rydzyna, Poland, 1–5 September 1980. Springer.

[KLSV06] Dilsun Kirli Kaynar, Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan & Claypool Publishers, 2006.

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

[NV07] Sumit Nain and Moshe Y. Vardi. Branching vs. linear time: Semantical perspective. In Namjoshi et al. [NYHO07], pages 19–34.

[NYHO07] Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors. *Automated Technology for Verification and Analysis, 5th International Symposium, ATVA 2007, Tokyo, Japan, October 22-25, 2007, Proceedings*, volume 4762 of *Lecture Notes in Computer Science*. Springer, 2007.

[Old87] E.-R. Olderog. Tcsp: Theory of communicating sequential processes. In Brauer et al. [BRR87], pages 441–465.

[RS] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 3.

[Tho99] Wolfgang Thomas, editor. *Foundations of Software Science and Computation Structure, Second International Conference, FoSSaCS'99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*, volume 1578 of *Lecture Notes in Computer Science*. Springer, 1999.

[Wal88] David Walker. Bisimulations and divergence. In *Proceedings of the Third Annual IEEE Symposium on Logic in Computer Science (LICS 1988)*, pages 186–192. IEEE Computer Society Press, July 1988.

[Win78] Józef Winkowski, editor. *Mathematical Foundations of Computer Science 1978, Proceedings, 7th Symposium, Zakopane, Poland, September 4-8, 1978*, volume 64 of *Lecture Notes in Computer Science*. Springer, 1978.

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages*. The MIT Press, Cambrige, Massachusetts, 1993.