Process Behaviour: Formulae versus Tests (Extended Abstract)

Andrea Cerone
Trinity College Dublin
Dublin, Ireland
School of Computer Science and Statistics*
ceronea@cs.tcd.ie

Matthew Hennessy
Trinity College Dublin
Dublin, Ireland
School of Computer Science and Statistics
Matthew.Hennessy@cs.tcd.ie

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 this property logic which can be captured by tests. Then we show that those subsets of the property logic capture precisely the power of tests.

1 Introduction

One central concern of concurrency theory is to determine whether two processes exhibit the same behaviour; to this end, many notions of behavioural equivalence have been investigated [Gla93]. One approach, proposed in [DH84], is based on tests. 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.

In the framework set up in [DH84] observers have very limited ability to manipulate the processes under test; informally processes are conceived as completely independent entities who 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 denotational models based on *Failures* used in the theory of CSP, [Hoa85, Old87, DN83].

Another approach to behavioural equivalence is to say that two processes are equivalent unless there is a property which one enjoys and the other does not. Here again much depends on the chosen set of properties, and what it means for a process to enjoy a property. *Hennessy Milner Logic* [HM85] is a modal logic often used for expressing process properties in term of the actions they are able to perform. It is well-known that it can be used, via differing interpretations, to determine numerous variations on *bisimulation equivalence*, [Mil89, AILS07]. What has received very little attention in the literature however is the relationship between these properties and tests. This is the subject of the current paper.

More specifically, we address the question of determining which formulae of a recursive version of the Hennessy Milner Logic, which we will refer to as *recHML*, can be used to characterise tests. This problem has already been solved in [AI99] for a non-standard notion of testing; this is discussed

*The financial support of Science Foundation Ireland is gratefully appreciated.

S. Fröschle, F.D. Valencia (Eds.): Workshop on
Expressiveness in Concurrency 2010 (EXPRESS'10).
EPTCS 41, 2010, pp. 31–45, doi:10.4204/EPTCS.41.3

© A, Cerone & M. Hennessy
This work is licensed under the
Creative Commons Attribution License.

more fully later in the paper. But we will focus on 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, 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 5.2

(ii) There is a formula $\phi_{\text{must}}(t)$ which *must*-represents t; see Theorem 4.18

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 5.1

(b) Every $\phi \in$ *mustHML* *must*-represents some test $t_{\text{must}}(\phi)$; see Theorem 4.14

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 test t with respect to *must* testing; 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 the next section we recall the formal definitions required to state our results precisely. Our results in the may case will only hold when the set of tests we consider come from a finite state finite branching LTS. Further, we also require for the LTS of processes to be finite branching when dealing with the *must* testing relation. The reader should also be warned that we use a slightly non-standard interpretation of *recHML*.

We then explain both *may* and *must* testing, where we take as processes the set of states from an arbitrary LTS, and give an explicit syntax for tests. In Section 3 we give a precise statement of our results, including definitions of the sub-languages *mayHML* and *mustHML*, together with some illuminating examples. The proofs of these results for the *must* case are given in Section 4, while those for the *may* case are outlined in Section 5. We end with a brief comparison with related work.

2 Background

One formal model for describing the behaviour of a concurrent system is given by **Labelled Transition Systems (LTSs)**:

Definition 2.1. *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.*

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 □.

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 $s\xrightarrow{\tau}s'\xrightarrow{\tau}\cdots$, while it converges, $s\Downarrow$, otherwise.

For a given LTS, each action of the form $\xrightarrow{a}$ can be interpreted as an observable activity; informally speaking, this means that each component which is external to the modeled system can detect that such an action has been performed. On the other hand, the action τ is meant to represent internal unobservable activity; this gives rise to the standard notation for weak actions. $s\stackrel{\tau}{\Longrightarrow}s'$ Is used to denote reflexive transitive closure of $\xrightarrow{\tau}$, while $s\stackrel{a}{\Longrightarrow}s'$ denotes $s\stackrel{\tau}{\Longrightarrow}s''\xrightarrow{a}s'''\stackrel{\tau}{\Longrightarrow}s'$. When $s\stackrel{\alpha}{\Longrightarrow}s'$ we say that s' is an α-derivative of s. The associated notation $s\stackrel{\alpha}{\Longrightarrow}$, $s\Longrightarrow$, $s\stackrel{\alpha}{\not\Longrightarrow}$ and $s\not\Longrightarrow$ have the obvious definitions.

It is common to define many operators on LTSs for interpreting process algebras. In this paper we will use only one, a parallel operator designed with *testing* in mind.

Definition 2.2 (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:

$$\frac{s\xrightarrow{\tau}s'}{s|t\xrightarrow{\tau}s'|t}\qquad\frac{t\xrightarrow{\tau}t'}{s|t\xrightarrow{\tau}s|t'}\qquad\frac{s\xrightarrow{a}s'\quad t\xrightarrow{a}t'}{s|t\xrightarrow{\tau}s'|t'}$$

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

The first two rules express the possibility for each component of a LTS to perform independently an internal activity, which cannot be detected by the other component. The last rule models the synchronization of two processes executing the same action; this will result in unobservable activity.

2.1 Recursive HML

Hennessy Milner Logic (HML), [HM85] has proven to be a very expressive property language for states in an LTS. It is 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 an LTS.

Definition 2.3 (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:*

$$\phi ::= tt \mid ff \mid X \mid Acc(A) \mid \phi_1\vee\phi_2 \mid \phi_1\wedge\phi_2 \mid \langle\alpha\rangle\phi \mid [\alpha]\phi \mid min(X,\phi) \mid max(X,\phi)$$

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

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

Table 1: Interpretation of *recHML*

processes for which each τ-derivative has at least an a-derivative for $a \in Act$. $min(X,\phi)$ 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 $\rho : Var \to 2^S$ is called environment. Let Env be the set of environments. 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 2.1. 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 $p \in [\![\phi]\!]$.

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

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.

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 $1 \leq i \leq n$. This formula will be interpreted as the i-th projection of the simultaneous fixpoint formula.

Definition 2.4 (Interpretation of simultaneous fixpoints)**.** *Let $\overline{X}$ and $\overline{\phi}$ respectively be sequences of vari-*

ables 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 pointwise. □

Again we will omit the environment application if a formula of the form $min_i(\overline{X},\overline{\phi})$ is closed, that is the only variables that occur in $\overline{\phi}$ are those in $\overline{X}$. 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 given by $X_i = \phi_i$ for all $i = 1, \cdots, n$, while $[\![min_i(\overline{X},\overline{\phi})]\!]$ is the i-th projection of such a vector. Simultaneous fixpoints do not add any expressivity to *recHML*, as shown below:

Theorem 2.5 (Bekíc, [Win93])**.**
For each formula $\phi \in recHML^+$ there is a formula $\psi \in recHML$ such that $[\![\phi]\!] = [\![\psi]\!]$. □

Later we will need the following properties of simultaneous fixpoints:

Theorem 2.6 (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) Let ρ_{min} be an environments such that $\rho_{min}(X_i) = [\![min_i(\overline{X},\overline{\phi})]\!]$. Then $[\![min_i(\overline{X},\overline{\phi})]\!] = [\![\phi_i]\!]\rho_{min}$. □

2.2 Tests

Another way to analyse the behaviour of a process is given by testing. Testing a process can be thought of as an experiment in which another process, called test, detects the actions performed by the tested process, reacting to it by allowing or forbidding the execution of a subset of observables. After observing the behaviour of the process, the test could decree that it satisfies some property for which the test was designed for by 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 $p \mid t \xrightarrow{\tau} p_1 \mid t_1 \xrightarrow{\tau} \ldots\ldots \xrightarrow{\tau} p_k \mid t_k \xrightarrow{\tau} \ldots$ as a *computation* of $p \mid t$. It may be finite or infinite; it is successful if there exists some $n \geq 0$ such that $t_n \xrightarrow{\omega}$. As only τ-actions can be performed in an experiment, we will omit the symbol τ in computations and in computation prefixes. Successful computations lead to the definition of two well known *testing relations*, [DH84]:

Definition 2.7 (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 $s \mid t$ is successful.*

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

$$t ::= 0 \mid \alpha.t \mid \omega.0 \mid X \mid t_1 + t_2 \mid \mu X.t\,. \tag{1}$$

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 is 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 $t\{\mu X.t/X\}$. 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 (1):

Proposition 2.8. *Let $\mathcal{T} = \langle T, Act_\tau, \longrightarrow \rangle$ be the LTS generated by a state t in grammar* (1)*: then $\mathcal{T}$ is both branching finite and finite state.* □

3 Testing formulae

Relative to a process LTS $\langle S, Act_\tau, \longrightarrow \rangle$ and a test LTS $\langle T, Act_\tau^\omega, \longrightarrow \rangle$, we now explore the relationship between tests from our default LTS of tests and formulae of *recHML*. Given a test t, our goal is to find a formula ϕ such that the set of processes which *may satisfy*/*must satisfy* such a 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 3.1. *Let ϕ be a recHML formula and t a test. We say that:*

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

Similar definitions are given for may testing. □

First some examples.

Example 3.2 (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 $p \xrightarrow{\tau} p$, 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 $p \mid t \to p \mid t \to \cdots$.*

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

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) $t \stackrel{\omega}{\Longrightarrow}$,

(ii) $t \stackrel{a}{\Longrightarrow} t' \stackrel{\omega}{\Longrightarrow}$,

(iii) $t \stackrel{b}{\Longrightarrow} t' \stackrel{\omega}{\Longrightarrow}$.

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

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

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

(d) In an analogous way to (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 3.3. *(Representable formulae)*

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

$$\phi ::= tt \mid ff \mid X \mid \langle\alpha\rangle\phi \mid \phi_1 \vee \phi_2 \mid min(X, \phi) \tag{2}$$

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

$$\phi ::= tt \mid ff \mid Acc(A) \mid X \mid [\alpha]\phi \mid \phi_1 \wedge \phi_2 \mid min(X, \phi) \tag{3}$$

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 coinductive property. Since there exist formulae of the form $[\alpha]\phi$, $\phi_1 \wedge \phi_2$ which are not *may*-representable, the $[\cdot]$ modality and the conjunction operator, have not been included in *mayHML* 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*.

Note also that the modality $[\cdot]$ is only used in *mustHML*, which will be compared with *must-testing*. No diverging process must satisfy a non-trivial test *t*, i.e. such that $t \stackrel{\omega}{\not\longrightarrow}$. Hence, in this setting, the convergence restriction on this modality is natural.

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.

4 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.

Lemma 4.1. *Let $\phi \in$ mustHML, and let $p \in [\![\phi]\!]$, where $p \Uparrow$: then $[\![\phi]\!]$ is the entire process space, i.e. $[\![\phi]\!] = S$.* □

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 4.2. *Let C be the set of subsets of S determined by:*

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

Proposition 4.3. *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 4.4. *Suppose the LTS of processes is finite-branching: If $X_0 \subseteq X_1 \subseteq \cdots$ is a chain of elements in C then*

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

□

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

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

□

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 4.6.

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

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 4.7. *For each ϕ in mustHML define $t_{must}(\phi)$ as follows:*

$$t_{must}(tt) = \omega.0 \tag{4}$$

$$t_{must}(ff) = 0 \tag{5}$$

$$t_{must}(Acc(A)) = \sum_{a\in A} a.\omega.0 \tag{6}$$

$$t_{must}(X) = X \tag{7}$$

$$t_{must}([\tau]\phi) = \tau.t_{must}(\phi) \tag{8}$$

$$t_{must}([a]\phi) = a.t_{must}(\phi) + \tau.\omega.0 \tag{9}$$

$$t_{must}(\phi_1 \wedge \phi_2) = \begin{cases} \omega.0 & \text{if } \phi_1 \wedge \phi_2 \text{ is closed and logically equivalent to } tt \\ \tau.Tmust\phi_1 + \tau.t_{must}(\phi_2) & \text{otherwise} \end{cases} \tag{10}$$

$$t_{must}(min(X,\phi)) = \begin{cases} t_{must}(\phi) & \text{if } \phi \text{ is closed} \\ \mu X.t_{must}(\phi) & \text{otherwise} \end{cases} \tag{11}$$

□

For each formula ϕ in *mustHML*, the test $t_{\mathrm{must}}(\phi)$ is defined in a way such that the set of processes which *must satisfy* $t_{\mathrm{must}}(\phi)$ is exactly $[\![\phi]\!]$. Before supplying the details of a formal proof of this statement, let us comment on the definition of $t_{\mathrm{must}}(\phi)$.
Cases (4), (5) and (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 $t_{\mathrm{must}}([\tau]\phi)$ can immediately perform a ω action. This is done by requiring the test $t_{\mathrm{must}}([\tau]\phi)$ to perform only an internal action.
Finally, (10) and (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 $t_{\mathrm{must}}(\phi)$ 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 $t_{\mathrm{must}}(\phi)$ does indeed capture the formula ϕ we need to establish some preliminary properties. The first essentially says that no formula of the form $min(X,\phi)$, with ϕ not closed, will be interpreted in the whole state space.

Lemma 4.8. *Let $\phi = min(X,\psi)$, with ψ not closed. Then $[\![\phi]\!] \neq S$.* □

Then we state some simple properties about recursive tests.

Lemma 4.9.

- p must satisfy $\mu X.t$ *implies* p must satisfy $t\{\mu X.t/X\}$.
- $p \Downarrow, p$ must satisfy $t\{\mu X.t/X\}$ *implies* 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 4.10. *Suppose the LTS of processes is finitely branching. If* p must satisfy $t_{must}(\phi)$ *then* $p \in [\![\phi]\!]$.

Proof. Suppose p *must satisfy* $t_{\mathrm{must}}(\phi)$; As both the LTS of processes (by assumption) and the LTS of tests (Proposition 2.8) are finite branching, the maximal length of a successful computation $|p,t|$ is defined and finite. This is a direct consequence of Konig's Lemma [BJ89]. Thus it is possible to perform an induction over $|p, t_{\mathrm{must}}(\phi)|$ to prove that $p \in [\![\phi^k]\!]$. The result will then follow from Proposition 4.6.

- If $|p, t_{\mathrm{must}}(\phi)| = 0$ then $t_{\mathrm{must}}(\phi) \xrightarrow{\omega}$, and hence for each $p \in S$ p *must satisfy* $t_{\mathrm{must}}(\phi)$. Further it is not difficult to show that ϕ is logically equivalent to tt, hence $p \in [\![\phi]\!]$.
- If $|p, t_{\mathrm{must}}(\phi)| = n+1$ 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 $t_{\mathrm{must}}(min(X,\psi)) \equiv t_{\mathrm{must}}(\psi)$ by definition.
  (b) If X is free in ψ then, by Lemma 4.9 p *must satisfy* $t_{\mathrm{must}}(\psi)\{\mu X.\, t_{\mathrm{must}}(\psi)\, /X\}$, which is syntactically equal to $t_{\mathrm{must}}(\psi\{min(X,\psi)/X\})$.
  Since $|p, t_{\mathrm{must}}(\psi\{min(X,\psi)/X\})| < |p, t_{\mathrm{must}}(\phi)|$, 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 4.10 we use the following concept:

Definition 4.11 (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*

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

□

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 4.12. *The relation* $\models$ *is a satisfaction relation. Further, it is the smallest satisfaction relation.* □

Proposition 4.12 ensures that, for any satisfaction relation R, $\models$ is included in R; in other words, if $p \models \phi$ then $p\ R\ \phi$. 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 4.13. *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* $t_{\text{must}}(\psi\{\phi/X\})$ implies *p must satisfy* $t_{\text{must}}(\phi)$. 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 4.8 $[\![\,\phi\,]\!] \neq S$; therefore Lemma 4.1 ensures that $p \Downarrow$, and hence by Lemma 4.9 it follows *p must satisfy* $t_{\text{must}}(\phi)$. □

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

Theorem 4.14. *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 4.10. Conversely suppose $p \in [\![\,\phi\,]\!]$. By Proposition 4.12 it follows that for all satisfaction relations R it holds $p\,R\,\phi$; hence, by Proposition 4.13, $p\,R_{\text{must}}\,\phi$, 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 4.15. *Assume we have a test-indexed set of variables $\{X_t\}$. For each test $t \in T$ define φ_t as below:*

$$\varphi_t \triangleq tt \qquad \text{if } t \xrightarrow{\omega} \tag{12}$$

$$\varphi_t \triangleq ff \qquad \text{if } t \not\xrightarrow{} \tag{13}$$

$$\varphi_t \triangleq \Big(\bigwedge_{a,t':t\xrightarrow{a}t'} [a]X_{t'}\Big) \wedge Acc(\{a \mid t \xrightarrow{a}\}) \qquad \text{if } t \not\xrightarrow{\omega}, t \not\xrightarrow{\tau}, t \longrightarrow \tag{14}$$

$$\varphi_t \triangleq \Big(\bigwedge_{t':t\xrightarrow{\tau}t'} [\tau]X_{t'}\Big) \wedge \Big(\bigwedge_{a,t':t\xrightarrow{a}t'} [a]X_{t'}\Big) \qquad \text{if } t \not\xrightarrow{\omega}, t \xrightarrow{\tau} \tag{15}$$

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 2.1. □

Notice that we have a finite set of variables $\{X_t\}$ and that the conjunctions in Definition 4.15 are finite, as the LTS of tests is finite state and finite branching. These two conditions are needed 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

$$\rho_{min}(X_t) \triangleq [\![\phi_t]\!] \qquad\qquad \rho_{must}(X_t) \triangleq \{p \mid p \textit{ must satisfy } t\}$$

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

Proposition 4.16. *For all $t \in T$ it holds that $\rho_{min}(X_t) \subseteq \rho_{must}(X_t)$.*

Proof. We just need to show that $[\![\varphi_t]\!]\rho_{must} \subseteq \rho_{must}(X_t)$: the result follows from an application of the *minimal fixpoint property*, Theorem 2.6 (i). The proof is carried out by performing a case analysis on t. We will only consider Case (14), as cases (12) and (13) are trivial and Case (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 $t \xrightarrow{a}$ and $p' \stackrel{a}{\Longrightarrow}$,

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

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

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

$$p \mid t \to p_1 \mid t \to \cdots p_k \mid t \to p_{k+1} \mid t \to \cdots \to p_n \mid t \tag{16}$$

For such a computation we have that $p_n \stackrel{\tau}{\Longrightarrow} p'$, and there exists p'' with $p' \xrightarrow{a} p''$ for some action a and test t' such that $t \xrightarrow{a} t'$. 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 (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 $t'' \xrightarrow{\omega}$. Hence, every computation from $p \mid t$ is successful. □

Proposition 4.17. *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), the maximal length of a successful computation fragment $|p,t|$ is defined and finite.
Therefore we proceed by induction on $|p,t|$; the main technical property used is the Fixpoint Property 2.6(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. $t \not\xrightarrow{\omega}, t \not\xrightarrow{\tau}, t \longrightarrow$: 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 $t \not\xrightarrow{\tau}$, such a test should follow from a synchronisation between p^{k-1} and t. We have that then that, whenever $p \stackrel{\tau}{\Longrightarrow} p^n$, there exists an action a such that $t \xrightarrow{a} t'$ and $p^n \stackrel{a}{\Longrightarrow} p^k$, 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. $t \not\xrightarrow{\omega}, t \xrightarrow{\tau}$: 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.

Theorem 4.18. *Assuming the LTS of processes is finite branching, every finitary test t is* must*-representable.*

Proof. Consider any test t. We can apply Definition 4.15 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 2.5 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 4.15 to define φ_t belongs to *mustHML*, it is ensured 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 equivalence) of *must*-testable formulae.

Corollary 4.19. *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 4.14 there exists a finite test $t = t_{\mathrm{must}}(\phi)$ which *must*-represents ϕ. Further, by theorem 4.18 there exists a formula $\psi = \phi_{\mathrm{must}}(t) \in$ *mustHML* which *must*-tests for t. Therefore
$$p \in [\![\phi]\!] \Leftrightarrow p \textit{ must satisfy } t_{\mathrm{must}}(\phi) \Leftrightarrow p \in [\![\psi]\!]$$

□

5 The may case

In this paper we simply state the corresponding theorems for *may* testing:

Theorem 5.1. *Suppose the LTS of processes is finite branching. Then for every $\phi \in mayHML$, there exists a test $t_{may}(\phi)$ such that, ϕ* may*-represents the test $t_{may}(\phi)$.*

Theorem 5.2. *Assuming the LTS of processes is finite branching, every test t is* may*-representable.*

Corollary 5.3. *Suppose ϕ is a formula in recHML which is* may*-testable. Then there exist some ψ in mayHML which is logically equivalent to it.*

Proof. Similar to that of Corollary 4.19. □

Our proofs for Theorem 5.2 and Theorem 5.1 are similar in style to the corresponding results for *must* testing, namely, namely Theorem 4.18 and Theorem 4.14. Also , as we point out in the Conclusion, they can be recovered by dualising the proofs of the corresponding Theorems in [AI99].

6 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 this is already a well-known result, [Hen85].

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. However we believe our results on *must* testing, specifically Theorem 4.14 and Theorem 4.18, are new.

We have concentrated on properties associated essentialy with the behavioural theory based on extensional 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.

References

[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.

[AILS07] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, New York, NY, USA, 2007.

[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.

[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.

[Hen85] M. Hennessy. Acceptance trees. *Journal of the ACM*, 32(4):896–928, October 1985.

[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 of *Lecture Notes in Computer Science*, pages 261–274, Rydzyna, Poland, 1–5 September 1980. Springer.

[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.

[RS96] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 3. Springer Verlag, Berlin, Heidelberg, New York, October 1996.

[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.