A Modal Logic for Message Passing Processes*

M. Hennessy and X. Liu

School of Cognitive and Computing Sciences

University of Sussex

Brighton BN1 9QH, England

January 1993

Abstract

A first–order modal logic is given for describing properties of processes which may send and receive values or messages along communication ports. We give two methods for proving that a process enjoys such a property. The first is the construction, for each process P and formula F, of a *characteristic formula* $P\ \mathtt{sat}\ F$ such that P enjoys the property F if and only if the formula $P\ \mathtt{sat}\ F$ is logically equivalent to $\mathtt{tt}$. The second is a sound and complete proof system whose judgements take the form $B \vdash P\colon F$, meaning: under the assumption B the process P enjoys the property F.

The notion of *symbolic operational semantics* plays a crucial role in the design of both the characteristic formulae and the proof system.

1 Introduction

Modal logics have long been used to represent properties of concurrent systems, [Sti87, Lar88, GS86, SI91], and verification tools have been developed to support their application [CPS88, GLZ89]. The fact that a process P satisfies a property F, where F is a property expressed in a modal logic, is represented as a satisfaction relation between processes and properties, $P \models F$. In papers such as [Sti87] proof systems are described for deducing judgements of the form $P \models F$ and a major concern is *compositionality*; the proof rules depend on the syntactic structure of the process P. In [Lar88] similar proof systems are described but in this case the proof rules do not use the structure of P but its *operational semantics*; the proof system is based on the behaviour of processes. Another approach, also justified behaviourally, is adopted in [GS86, SI91]. In this approach, for any process P a

*This work has been supported by the SERC grant GR/H16537 and the ESPRIT BRA CONCUR II

formula ϕ_P is defined with the property that for any formula F, $P \models F$ if and only if $\phi_P \rightarrow F$ is a valid formula. Thus deciding whether or not a process satisfies a property is reduced to testing for the validity of a formula in the underlying modal logic.

Much of this work is carried out within the framework of *pure* process algebras, where processes are determined by their ability to perform atomic or uninterpreted actions and the modal formula describe certain aspects of their behaviour vis. a vis. these actions. Thus satisfying the modal property $[a](\langle b\rangle \mathsf{tt} \vee [c]\mathsf{ff})$ means that every time a process performs the atomic action a either it can subsequently perform the action b or it can not perform the action c. The aim of this paper is to extend this work to what we call *message passing* process algebras, i.e. process descriptions in which these actions have a certain interpretation; the reception and transmission of data along conceptual channels. Thus, for example,

$$P \Longleftarrow c?x.\mathsf{if}\ x = 0\ \mathsf{then}\ c!x.P\ \mathsf{else}\ d!(x+1).P$$

describes a process which can input a value on the channel c and output its successor on the channel d unless it is zero in which case it is output on the original channel.

Properties of such processes depend, of course, on properties of the underlying message–space and thus proof systems for inferring such properties will also have to derive theorems valid in this message–space. Here we take the approach advocated in [Hen91]; we factor out as much as possible all this auxiliary reasoning. In the extended modal logic we can express assertions of the underlying message–space but in the accompanying proof systems theorems involving such assertions are obtained for "free"; i.e. we assume the existence of an oracle which will determine the truth or otherwise of these assertions. In reality, in any implementation, we would call on an auxiliary proof system to establish such assertions. In this paper we will allow any first order formula as an assertion about the underlying message–space.

Nevertheless we do have to generalise the modal operators $\langle a\rangle$ and $[a]$ to the setting of message–passing on channels. For output actions this generalisation is straightforward. For each channel name c and data–variable x we have two modalities $\langle c!x\rangle$ and $[c!x]$. Intuitively a process P satisfies $\langle c!x\rangle F$ if it is possible for it to output on the channel c a message v and in doing so evolve to a state P' such that P' satisfies the formula $F[v/x]$; the interpretation of $[c!x]$ is similar. However input actions are somewhat more complicated and the form of the modal logic depends on the operational semantics of the process language. There are two natural generalisations of the standard operational semantics of pure processes, called *early* and *late* in [HL92, MPW92]. Here we use the *late* operational semantics which expresses the ability of processes to perform abstract actions of the form $P \xrightarrow{c?} (x)A$ where $(x)A$ is an *abstraction* which describes what will happen when a value is received on the channel c. Thus we have two further simple modalities $\langle c?\rangle$ and $[c?]$ associated with input actions but now these must be followed by formulae expressing properties of abstractions; these take the form of quantified assertions, of the form $\exists x F$ and $\forall x F$. The end result is a very powerful language for describing properties of message–passing systems. For example

$$[c?]\forall x[d!y](y = x + 1)$$

asserts that whenever a value is input on the channel c and whenever a value is subsequently output on d the value output is the successor of the value input.

There are two main results in this paper. The first concerns *characteristic formulae*. For every message–passing process P and for every property formula F, a formula from the first-order modal language described above, we define the characteristic formula $P\,\mathtt{sat}\,F$ which has the property that P enjoys the property F if and only if $P\,\mathtt{sat}\,F$ is logically equivalent to $\mathtt{tt}$. Thus checking whether or not P satisfies F is reduced to checking the validity of $P\,\mathtt{sat}\,F$. Note that this characteristic formula depends on F as well as P, thus it does not correspond to ϕ_P as in [GS86, SI91]. Rather it corresponds to $\phi_P \Rightarrow F$. The second result is the soundness and completeness of a proof system for deriving assertions of the form: process P satisfies property F. The actual judgements of the proof system are a little more general:

$$B \vdash P : F$$

and may be read "if the formula B is true then the process P satisfies the property F".

The proof of the results relie heavily on the development of *symbolic operational semantics* as in [HL92]. Recall that the standard operational semantics of processes is only defined for *closed* terms, i.e. terms which contain no free occurrences of data-variables. But the proof system is necessarily concerned with open terms; whether or not the process $c?x.T$ enjoys a particular property depends on whether the open term T enjoys a related property. Thus we are inevitably led into investigations of open terms and we do so using symbolic operational semantics. One consequence of this approach is that our results, such as the completeness of the proof system, apply not only to closed terms but to arbitrary terms.

We end this introduction with a brief outline of the remainder of the paper. The next section contains a description of the message–passing process algebra we consider and the first–order modal language used to describe its properties. This is followed, in Section 3, by the first result on characteristic formulae; its definition is based on the symbolic operational semantics alluded to above. The proof system, together with its soundness and completeness is given in Section 4 and the paper ends with a short conclusion.

2 Message Passing Processes and Their Modal Properties

We first present a language for message passing process, define its concrete operational semantics and the notion of (late) bisimulation equivalence. We then present a modal logic for expressing properties of these processes.

The syntax and semantics of message passing processes presented here is a simplified version of that presented in [IT92]. It presupposes a language for message or value expressions ranged over by e. This contains at least a set of *values* V ranged over by $v, \ldots$, and a set of *value variables* X, ranged over by $x, y, \ldots$. We

assume that *substitutions*, mappings from X to expressions, act in the standard way on expressions; we use $e[\sigma]$ to denote the expression which results from applying the substitution σ to the expression e. An *evaluation* ρ is a particular type of substitution; it maps variables to values and we assume that for every expression e, $e[\rho]$ always evaluates to a value. Moreover we assume that this evaluation depends only on the variables occurring in e; if we let $fv(e)$ to denote the variables which occur in e then $e[\rho] = e[\rho']$ whenever ρ and ρ' agree on $fv(e)$. If e contains no variables then $e[\rho]$ is independent of ρ and we denote this value by $[\![e]\!]$. In a similar manner we assume a language for boolean expressions, ranged over by b, and substitutions and evaluations act on boolean expressions in a similar manner.

The syntax of process expression is defined by the following BN-form:

$$T ::= \mathbf{0} \mid \tau.T \mid c?x.T \mid c!e.T \mid T + T' \mid \mathsf{if}\ b\ \mathsf{then}\ T\ \mathsf{else}\ T' \mid T|T' \mid T\backslash L \mid T[f] \mid C(e_1, \ldots, e_n)$$

We will first briefly explain these process constructions. Later a formal semantics will be given. The term $\mathbf{0}$ represents a process which does nothing while $\tau.P$ will do an internal action τ and then behave like the process P. The process $c?x.T$ will receive a value v on channel c and then behaves like the process $T[v/x]$, where we use the standard notation $T[v/x]$ to describe the substitution of v for all free occurrences of the variable x in T. Thus in $c?x.T$ the prefix $c?x$ is a binding operator for the value variable x and this leads to the standard definition of *free* and *bound* occurrences of variables in terms. The application of a substitution or evaluation, defined above on value expressions, is generalised to terms; thus for example $T[\sigma]$ denotes the result of substituting in T all free occurrences of x by $\sigma(x)$. We will also take for granted the definition of α-conversion, $\equiv_\alpha$, on terms. Returning to our informal explanation of the language $c!e.P$ will send the value of the expression e on channel c and subsequently behave like P. The operator $+$ represent choice: $P + P'$ will either proceed like P or P'. The process $\mathsf{if}\ b\ \mathsf{then}\ P\ \mathsf{else}\ P'$ will behave like P if the value of the boolean expression b is true and otherwise will behave like P'. Parallel composition is represented by $P|P'$ which also allows values to be exchanged between P and P'. We also have standard restriction and relabelling operators; if L is a set of channel names then in $P\backslash L$ external communication by the process P is restricted to those channels not in L while $P[f]$ acts exactly like the process P except that the channels are relabelled by f, a renaming function over the set of channel names. Finally, $C(x_1, \ldots, x_n)$ is a process constant of arity n. Each process constant $C(x_1, \ldots, x_n)$ has associated with it a *definition*, written as $C(x_1, \ldots, x_n) \Leftarrow T$, where all free variables of T are in $\{x_1, \ldots, x_n\}$. Thus $C(x_1, \ldots, x_n)$ is a process which behaves as T; the term T is referred to as the *body* of the definition.

The above informal description of process behaviour can be made precise by a formal operational semantics. This describes what actions a process can perform and the consequence of performing such actions. It is only defined over *closed*

terms, i.e. terms which contain no occurrences of free variables; we use $P, Q, \ldots$ to range over these processes. Such an operational semantics is given in Figure 1. It presupposes that each process constant has associated with it a definition as explained above and moreover we assume that the body of each such definition is *guarded*, i.e. every occurrence of a process constant is within a prefix subterm, [Mil89].

There are three kinds of actions: internal action τ, input action on a channel $c?$, and output action on a channel $c!$. The result of an internal action is another process. The result of an input action on a channel c is an *abstraction* which is a function from values to processes while the result of an output action on a channel c is a pair consisting of the output value and the residual process after the output action. Abstractions and pairs have no transitions of their own. Their interaction is seen in a communication event between two processes P and Q (rule Com). If process P can output v on channel c then behave as process P', that is $P \xrightarrow{c!} (v, P')$ in terms of the transition relation, and process Q can input on channel c and its resulting behaviour is represented by the abstraction $(x)Q'$, that is $Q \xrightarrow{c?} (x)Q'$ in terms of transition relation, then $P|Q$ can do an internal communication in which v is transmitted to Q'; this is described by $P|Q \xrightarrow{\tau} P'|Q'[v/x]$.

The rules given in Figure 1 are straightforward; they correspond to the late (concrete) operational semantics of [HL92, MPW92]. The conventions used in the rules Rest, Ren should be obvious ; for any action α $name(\alpha)$ refers to the channel name used in the action while $f(\alpha)$ renames the channel according to the renaming function f.

The notion of (late) bisimulation can be defined based on the operational semantics.

Definition 2.1 *A bisimulation relation $\mathcal{B}$ is a symmetric binary relation on processes such that whenever $(P, Q) \in \mathcal{B}$ then:*

1. *Whenever $P \xrightarrow{\tau} P'$ then $Q \xrightarrow{\tau} Q'$ for some Q' such that $(P', Q') \in \mathcal{B}$,*
2. *whenever $P \xrightarrow{c!} (v, P')$ then $Q \xrightarrow{c!} (v, Q')$ for some Q' such that $(P', Q') \in \mathcal{B}$,*
3. *whenever $P \xrightarrow{c?} (x)T$ then $Q \xrightarrow{c?} (y)U$ for some $(y)U$ such that for all $v \in V$, $(T[v/x], U[v/y]) \in \mathcal{B}$.*

Two processes P and Q are said to be bisimulation equivalent, written as $P \sim Q$, if (P, Q) is included in some bisimulation relation $\mathcal{B}$.

To describe properties of these processes, we present a modal logic shown in Figure 2. This logic resembles the one in [MPW92] for the π–calculus in many ways. In particular, our logic here characterizes the late bisimulation equivalence between message passing processes just like the modal logic in [MPW92] characterizes the late bisimulation equivalence between processes of the π–calculus.

This definition presupposes a class of first-order formulae ranged over by B. Just as the process language is parameterised on a language for expressions this

Act $\quad \dfrac{}{\tau.P \xrightarrow{\tau} P} \qquad \dfrac{}{c!e.P \xrightarrow{c!} (\llbracket e \rrbracket, P)} \qquad \dfrac{}{c?x.T \xrightarrow{c?} (x)T}$

Sum $\quad \dfrac{P \xrightarrow{\alpha} A}{P + Q \xrightarrow{\alpha} A} \qquad \dfrac{Q \xrightarrow{\alpha} A}{P + Q \xrightarrow{\alpha} A}$

Cond $\quad \dfrac{P \xrightarrow{\alpha} A}{\mathsf{if}\ b\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\alpha} A} \quad \llbracket b \rrbracket = \mathsf{tt}$

$\dfrac{Q \xrightarrow{\alpha} A}{\mathsf{if}\ b\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\alpha} A} \quad \llbracket b \rrbracket = \mathsf{ff}$

Par $\quad \dfrac{P \xrightarrow{\tau} P'}{P|Q \xrightarrow{\tau} P'|Q} \qquad \dfrac{P \xrightarrow{c!} (v, P')}{P|Q \xrightarrow{c!} (v, P'|Q)}$

$\dfrac{P \xrightarrow{c?} (x)T}{P|Q \xrightarrow{c?} (x)(T|Q)}$

Com $\quad \dfrac{P \xrightarrow{c!} (v, P') \quad Q \xrightarrow{c?} (x)T}{P|Q \xrightarrow{\tau} P'|T[v/x]}$

Rest $\quad \dfrac{P \xrightarrow{\alpha} A}{P\backslash L \xrightarrow{\alpha} A\backslash L} \quad name(\alpha) \cap L = \emptyset$

Ren $\quad \dfrac{P \xrightarrow{\alpha} A}{P[f] \xrightarrow{f(\alpha)} A[f]}$

Rec $\quad \dfrac{T[v_1/x_1, \ldots, v_n/x_n] \xrightarrow{\alpha} A}{C(v_1, \ldots, v_n) \xrightarrow{\alpha} A} \quad C(x_1, \ldots, x_n) \Longleftarrow T$

(Symmetric cases for Par and Com are omitted)

Figure 1: The Transition Rules for Processes

$$
\begin{array}{rcl}
F & ::= & B \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \langle\tau\rangle F \mid [\tau]F \mid \\
 & & \langle c!x\rangle F \mid [c!x]F \mid \langle c?\rangle G \mid [c?]G \\
G & ::= & \exists x F \mid \forall x F
\end{array}
$$

Figure 2: Syntax of the modal logic

$$
\begin{array}{lcl}
P \models B & : & [\![B]\!] = \mathsf{tt} \\
P \models F_1 \vee F_2 & : & P \models F_1 \text{ or } P \models F_2 \\
P \models F_1 \wedge F_2 & : & P \models F_1 \text{ and } P \models F_2 \\
P \models \langle\tau\rangle F & : & \text{for some } P',\ P \xrightarrow{\tau} P' \text{ and } P' \models F \\
P \models [\tau]F & : & \text{whenever } P \xrightarrow{\tau} P' \text{ then } P' \models F \\
P \models \langle c!x\rangle F & : & \text{for some } P' \text{ and } v,\ P \xrightarrow{c!} (v, P'),\ P' \models F[v/x] \\
P \models [c!x]F & : & \text{whenever } P \xrightarrow{c!} (v, P') \text{ then } P' \models F[v/x] \\
P \models \langle c?\rangle G & : & \text{for some } (y)T,\ P \xrightarrow{c?} (y)T,\ (y)T \models G \\
P \models [c?]G & : & \text{whenever } P \xrightarrow{c?} (y)T, \text{ then } (y)T \models G \\
(y)T \models \exists x F & : & \text{for some value } v \in V,\ T[v/y] \models F[v/x] \\
(y)T \models \forall x F & : & \text{for all value } v \in V,\ T[v/y] \models F[v/x]
\end{array}
$$

Figure 3: Semantic clauses

property language is also parameterised on this set of formulae; we only assume that such B are formula which can be interpreted over the same set of values V; if B is closed, i.e. contains no free occurrences of variables, then its interpretation will be denoted by $[\![B]\!]$ and this is always be either tt or ff. There are two syntactic classes of formulae. The first is ranged over by F and these are designed to describe properties of processes. while the second, ranged over by G, apply to abstractions. As usual $\forall x$ and $\exists x$ are binding operators but the two modal operators $\langle c!x\rangle$ and $[c!x]$ also bind variables. We take for granted the resulting definitions of substitution on formulae, α-conversion etc. .

The satisfaction relation $\models$, between processes and closed formulae, is defined by structural induction over closed formula in Figure 3. This determines when a process satisfies a process formula F and when an abstraction satisfies an abstraction formula G. The two modalities, $\langle\alpha\rangle$ and $[\alpha]$, correspond to existential

and universal quantification over transitions. That is, in order to satisfy a formula $\langle\alpha\rangle F$, a process must possess some α–transition leading to a configuration (a process or an abstraction) satisfying F. Dually, for a process to satisfy a formula $[\alpha]F$, any α–transition must lead to configurations satisfying F.

The logic characterizes the (late) bisimulation equivalence between message passing processes as the following theorem claims.

Theorem 2.2 (Modal characterization) *$P \sim Q$ if and only if for all (closed) modal formula F, $P \models F \Leftrightarrow Q \models F$.*

Proof: $\Rightarrow$: Suppose $P \sim Q$ and $P \models F$, we can show that $Q \models F$ by induction on the structure of F. It is easy when F has the forms $B, F_1 \wedge F_2, F_1 \vee F_2, \langle\tau\rangle F', [\tau]F'$. If F is $\langle c!x\rangle F'$, then there exists $P \xrightarrow{c!} (v, P')$ such that $P' \models F[v/x]$. Because $P \sim Q$, so $Q \xrightarrow{c!} (v, Q')$ for some Q' with $P' \sim Q'$, and by induction hypothesis $Q' \models F[v/x]$. Thus $Q \models \langle c!x\rangle F'$. Similarly we can prove the case when F is $[c!x]F'$. If F is $\langle c?\rangle G$, then there exists $P \xrightarrow{c?} (x)T$ such that $(x)T \models G$. Because $P \sim Q$, so $Q \xrightarrow{c?} (y)U$ for some $(y)U$ with $T[v/x] \sim U[v/y]$ for all $v \in V$. Now we prove that for this $(y)U$, $(y)U \models G$ and thus $Q \models \langle c?\rangle G$. Note that G is either $\forall z F'$ or $\exists z F'$, we will show that in both cases $(y)U \models G$. When G is $\forall z F'$, $(x)T \models G$ implies $T[v/x] \models F'[v/z]$ for all $v \in V$ and by induction hypothesis $U[v/y] \models F'[v/z]$ for all $v \in V$ hence $(y)U \models G$. When G is $\exists z F'$, $(x)T \models G$ implies $T[v/x] \models F'[v/z]$ for some v and, since $T[v/x] \sim U[v/y]$, by induction hypothesis $U[v/y] \models F'[v/z]$ hence also $(y)U \models G$. Similarly we can prove the case when F is $[c?]G$.
$\Leftarrow$: Let $\mathcal{B} = \{(P, Q) | \forall F. P \models F \Leftrightarrow Q \models F\}$. We will prove that $\mathcal{B}$ is a bisimulation. Suppose $(P, Q) \in \mathcal{B}$, we can show the following:

1. Whenever $P \xrightarrow{\tau} P'$ then $Q \xrightarrow{\tau} Q'$ for some Q' with $(P', Q') \in \mathcal{B}$,
2. Whenever $P \xrightarrow{c!} (v, P')$ then $Q \xrightarrow{c!} (v, Q')$ for some Q' with $(P', Q') \in \mathcal{B}$,
3. Whenever $P \xrightarrow{c?} (x)T$ then $Q \xrightarrow{c?} (y)U$ for some $(y)U$ such that for all $v \in V$, $(T[v/x], U[v/y]) \in \mathcal{B}$.

Here we only show the last case, the other two are simpler. Assume otherwise, that is there exists $P \xrightarrow{c?} (x)T$ such that whenever $Q \xrightarrow{c?} (y)U$ then $(T[v/x], U[v/y]) \notin \mathcal{B}$ for some v. More specifically and without loss of generality, there exists $P \xrightarrow{c?} (x)T$ such that whenever $Q \xrightarrow{c?} (y)U$ then there exists $v \in V$ and F' such that $T[v/x] \models F'$ and $U[v/y]) \not\models F'$. Because we assume that all definitions are guarded the operational semantics is *finite-branching*, so there are finitely these $(y)U$'s and we can list these (v, F')'s in $(v_1, F_1), \ldots, (v_n, F_n)$. Now

$$F \equiv \langle c?\rangle \forall x \bigwedge_{1 \leq i \leq n} (x = v_i \rightarrow F_i)$$

is a formula such that $P \models F$ but $Q \not\models F$, a controdiction. □

3 Proving Properties of Message Passing Processes

In this section, we will look at how to reason about whether a process satisfies a modal property. From the definition of the satisfaction relation $\models$ it seems quite natural to express whether $P \models F$ holds as a first order formula $R_{P,F}$, such that $[\![R_{P,F}]\!] = \mathtt{tt}$ just in case $P \models F$ holds. This idea is attractive because if we can do this for any process P and modal formula F, and assume that we have a first order theory about the domain V at our disposal, then the problem of whether $P \models F$ holds is reduced to the problem of whether we can establish $R_{P,F}$ in the first order theory. We will call such $R_{P,F}$ the characteristic formula of P satisfying F and write it as $P\ \mathtt{sat}\ F$ in the rest of the paper.

Now let us see how to construct $P\ \mathtt{sat}\ F$ for a given process P and formula F. Naturally the construction should be done inductively on the structure of F. Take the case $F_1 \wedge F_2$ as an example; assuming we have constructed $P\ \mathtt{sat}\ F_1$ and $P\ \mathtt{sat}\ F_2$, according to the definition of $\models$, it is reasonable to define $P\ \mathtt{sat}\ F_1 \wedge F_2$ as $P\ \mathtt{sat}\ F_1 \wedge P\ \mathtt{sat}\ F_2$. Take $\langle\tau\rangle F$ as another example; it is not difficult to see that $P\ \mathtt{sat}\ \langle\tau\rangle F$ should be $\bigvee_{P \xrightarrow{\tau} P'} P'\ \mathtt{sat}\ F$ assuming that we have already constructed $P'\ \mathtt{sat}\ F$ for all P' such that $P \xrightarrow{\tau} P'$. This construction relies on the finite branching of the operational semantics. It also uses the operational semantics of P in constructing $P\ \mathtt{sat}\ \langle\tau\rangle F$, and this is fine as long as we are working with a process P, i.e. a closed term, and a closed formula F. This inductive approach works until we come to consider abstractions and abstraction formulae, $(y)P\ \mathtt{sat}\ \exists x F$. From the definition of $\models$ it seems that $(y)P\ \mathtt{sat}\ \exists x F$ should be $\exists z P[z/y]\ \mathtt{sat}\ F[z/x]$. The problem here is that $P[z/y], F[z/x]$ are not closed any more, and we cannot use the induction hypothesis to assume that $P[z/y]\ \mathtt{sat}\ F[z/x]$ has already been constructed. So we have to generalize the construction to suit open terms and open formulae even if we are only interested in the construction for closed terms. A major obstacle in such a generalization is that the operational semantics for processes is not adequate in the case of open terms. So, in order to generalize the construction, we first need a suitable operational semantics for open terms.

In [HL92], an abstract structure of symbolic transition graph is introduced to describe the operational semantics of open terms. Here we will describe directly on open terms a *symbolic operational semantics*, as a symbolic transition graph whose nodes are the open process terms themselves. This is in contrast to the *concrete* operational semantics defined for processes in Figure 1.

Such an operational semantics is given in Figure 4. Here transitions are labelled not only by the action which takes place, but also by a boolean expression which is the condition of that action. For example $\tau.T$ can perform τ in all circumstances, so the transition is labelled by the boolean $\mathtt{tt}$. If T can perform α under condition b, then $\mathsf{if}\ b'\ \mathsf{then}\ T\ \mathsf{else}\ U$ can perform α under condition $b \wedge b'$. If process T can output on channel c under condition b, that is $T \xrightarrow{b,c!} (e, T')$, and process U can input on channel c under condition b', that is $U \xrightarrow{b',c?} (x)U'$, then $T|U$ can do an internal communication under condition $b \wedge b'$, described by $T|U \xrightarrow{b \wedge b',\tau} T'|U'[e/x]$.

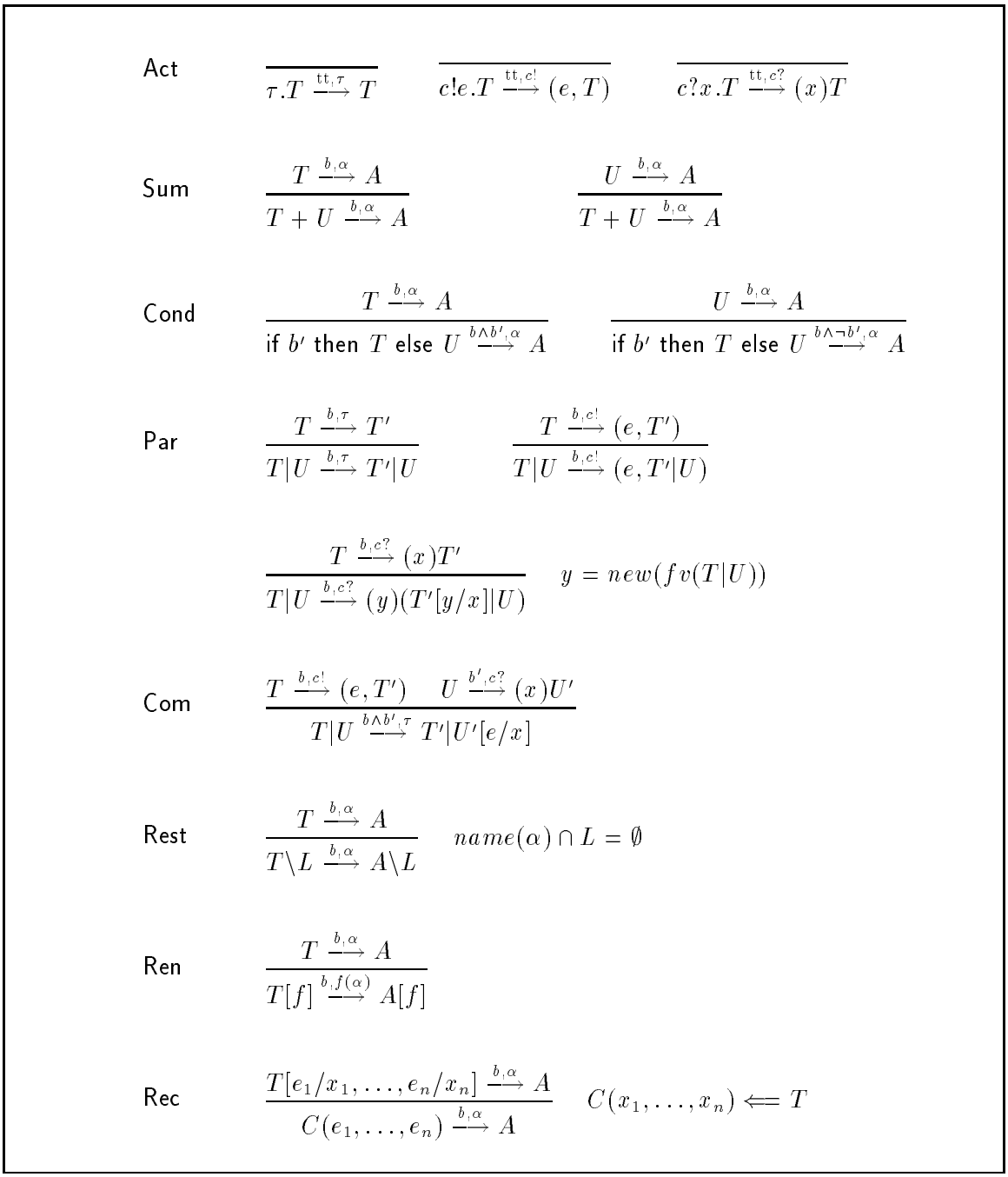

Figure 4: The Transition Rules for Terms

Now look at the third Par rule, it is possible that the bound variable x in $(x)T'$ also appears free in U. Obviously these two appearances of x should not be confused. In order to avoid such confusion, we assume a function new which takes a set of variables X as argument and produce a new variable which is not in X, and use new to change the bound variable.

The first lemma shows that this operational semantics for open terms is in fact a generalization of the operational semantics for processes in Figure 1.

Lemma 3.1 *For any process P, $P \xrightarrow{\alpha} A$ if and only if $P \xrightarrow{b,\alpha} A$ for some (closed) b with $[\![b]\!] = \mathtt{tt}$.*

Proof: The statement of the lemma assumes that a closed value expression e is identified with its value $[\![e]\!]$. The proof consists of two routine inductions, one on the rules in Figure 1, and the other on those of Figure 4. □

The next result states that in some sense the symbolic operational semantics is preserved under substitutions.

Lemma 3.2 *For any term T and substitution σ, $T[\sigma] \xrightarrow{b',\alpha} A'$ if and only if $T \xrightarrow{b,\alpha} A$ for some b, A such that $b' \equiv b[\sigma], A' \equiv_\alpha A[\sigma]$.*

Proof: Each direction is proved by induction on the rules in Figure 4.

□

With these symbolic actions we can now give the construction of $T\ \mathtt{sat}\ F$, the characteristic formulae of T satisfying F for any open term T and formula F. The construction is inductive on the structure of F and is shown in Figure 5.

Because the construction now is for terms and formulae which may contain free variables, we need a more general interpretation than $[\![P\ \mathtt{sat}\ F]\!] = \mathtt{tt} \Leftrightarrow P \models F$. In the following we will prove that the correct interpretation here is $[\![(T\ \mathtt{sat}\ F)[\rho]]\!] = \mathtt{tt} \Leftrightarrow T[\rho] \models F[\rho]$ for any evaluation ρ. For this we need the following two results:

Proposition 3.3 *For any process P and closed process formula F, any closed abstraction term $(y)T$ and closed abstraction formula G, the following equivalences hold:*

$$[\![P\ \mathtt{sat}\ F]\!] = \mathtt{tt} \quad \Leftrightarrow \quad P \models F$$
$$[\![(y)T\ \mathtt{sat}\ G]\!] = \mathtt{tt} \quad \Leftrightarrow \quad (y)T \models G$$

Proof: They are proved by induction on the stucture of F and G with the help of Lemma 3.1. We give four example cases.

$$
\begin{array}{rcl}
P \,\mathtt{sat}\, B & \equiv & B \\
P \,\mathtt{sat}\, F_1 \vee F_2 & \equiv & P \,\mathtt{sat}\, F_1 \vee P \,\mathtt{sat}\, F_2 \\
P \,\mathtt{sat}\, F_1 \wedge F_2 & \equiv & P \,\mathtt{sat}\, F_1 \wedge P \,\mathtt{sat}\, F_2 \\
P \,\mathtt{sat}\, \langle\tau\rangle F & \equiv & \bigvee_{P \xrightarrow{b,\tau} P'} b \wedge P' \,\mathtt{sat}\, F \\
P \,\mathtt{sat}\, [\tau] F & \equiv & \bigwedge_{P \xrightarrow{b,\tau} P'} b \rightarrow P' \,\mathtt{sat}\, F \\
P \,\mathtt{sat}\, \langle c!x\rangle F & \equiv & \bigvee_{P \xrightarrow{b,c!} (e,P')} b \wedge P' \,\mathtt{sat}\, F[e/x] \\
P \,\mathtt{sat}\, [c!x] F & \equiv & \bigwedge_{P \xrightarrow{b,c!} (e,P')} b \rightarrow P' \,\mathtt{sat}\, F[e/x] \\
P \,\mathtt{sat}\, \langle c?\rangle G & \equiv & \bigvee_{P \xrightarrow{b,c?} (y)P'} b \wedge (y)P' \,\mathtt{sat}\, G \\
P \,\mathtt{sat}\, [c?] G & \equiv & \bigwedge_{P \xrightarrow{b,c?} (y)P'} b \rightarrow (y)P' \,\mathtt{sat}\, G \\
(y)P \,\mathtt{sat}\, \forall x F & \equiv & \forall z P[z/y] \,\mathtt{sat}\, F[z/x] \\
(y)P \,\mathtt{sat}\, \exists x F & \equiv & \exists z P[z/y] \,\mathtt{sat}\, F[z/x]
\end{array}
$$

where z does not occur free in $(y)P$ and $\exists x F (\forall x F)$

Figure 5: Construction of $P \,\mathtt{sat} F$

The case $\langle\tau\rangle F$:

$$
\begin{array}{lll}
 & [\![P \,\mathtt{sat}\, \langle\tau\rangle F]\!] = \mathsf{tt} & \\
\Leftrightarrow & [\![\bigvee_{P \xrightarrow{b,\tau} P'} b \wedge P' \,\mathtt{sat}\, F]\!] = \mathsf{tt} & \text{def. of } \mathtt{sat} \\
\Leftrightarrow & \text{for some } P', P \xrightarrow{b,\tau} P' \text{ with } [\![b]\!] = \mathsf{tt} & \\
 & \text{and } [\![P' \,\mathtt{sat}\, F]\!] = \mathsf{tt} & \\
\Leftrightarrow & \text{for some } P', P \xrightarrow{\tau} P' \text{ with } [\![P' \,\mathtt{sat}\, F]\!] = \mathsf{tt} & \text{Lemma 3.1} \\
\Leftrightarrow & \text{for some } P', P \xrightarrow{\tau} P' \text{ with } P' \models F & \text{ind. hyp.} \\
\Leftrightarrow & P \models \langle\tau\rangle F & \text{def. of } \models
\end{array}
$$

The case $\langle c!x\rangle F$:

$$
\begin{array}{lll}
 & [\![P \,\mathtt{sat}\, \langle c!x\rangle F]\!] = \mathtt{tt} & \\
\Leftrightarrow & [\![\bigvee_{P \xrightarrow{b,c!} (e,P')} b \wedge P' \,\mathtt{sat}\, F[e/x]]\!] = \mathtt{tt} & \text{def. of } \mathtt{sat} \\
\Leftrightarrow & \text{for some } e, P', P \xrightarrow{b,c!} (e,P') \text{ with } [\![b]\!] = \mathtt{tt} & \\
 & \text{and } [\![P' \,\mathtt{sat}\, F[e/x]]\!] = \mathtt{tt} & \\
\Leftrightarrow & \text{for some } e, P', P \xrightarrow{c!} ([\![e]\!], P') \text{ and } [\![P' \,\mathtt{sat}\, F[[\![e]\!]/x]]\!] = \mathtt{tt} & \text{Lemma 3.1} \\
\Leftrightarrow & \text{for some } e, P', P \xrightarrow{c!} ([\![e]\!], P') \text{ with } P' \models F[[\![e]\!]/x] & \text{ind. hyp.} \\
\Leftrightarrow & P \models \langle c!x\rangle F & \text{def. of } \models
\end{array}
$$

The case $[c?]G$:

$$
\begin{array}{lll}
 & [\![P \,\mathtt{sat}\, [c?]G]\!] = \mathtt{tt} & \\
\Leftrightarrow & [\![\bigwedge_{P \xrightarrow{b,c!} (y)T} b \rightarrow (y)T \,\mathtt{sat}\, F]\!] = \mathtt{tt} & \text{def. of } \mathtt{sat} \\
\Leftrightarrow & \text{whenever } P \xrightarrow{b,c?} (y)T \text{ with } [\![b]\!] = \mathtt{tt} & \\
 & \text{then } [\![(y)T \,\mathtt{sat}\, G]\!] = \mathtt{tt} & \\
\Leftrightarrow & \text{whenever } P \xrightarrow{c?} (y)T \text{ then } [\![(y)T \,\mathtt{sat}\, G]\!] = \mathtt{tt} & \text{Lemma 3.1} \\
\Leftrightarrow & \text{whenever } P \xrightarrow{c?} (y)T \text{ then } (y)T \models G & \text{ind. hyp.} \\
\Leftrightarrow & P \models [c?]G & \text{def. of } \models
\end{array}
$$

The case $\forall x F$:

$$
\begin{array}{lll}
 & [\![(y)T \,\mathtt{sat}\, \forall x F]\!] = \mathtt{tt} & \\
\Leftrightarrow & [\![\forall z T[z/y] \,\mathtt{sat}\, F[z/x]]\!] = \mathtt{tt} & \text{def. of } \mathtt{sat} \\
\Leftrightarrow & \text{for all } v \in V, [\![T[v/y] \,\mathtt{sat}\, F[v/x]]\!] = \mathtt{tt} & \\
\Leftrightarrow & \text{for all } v \in V, T[v/y] \models F[v/x] & \text{ind. hyp.} \\
\Leftrightarrow & (y)T \models \forall x F & \text{def. of } \models
\end{array}
$$

□

Proposition 3.4 *For any substitution σ, the following logical equivalences hold:*

$$
\begin{array}{lll}
(T \,\mathtt{sat}\, F)[\sigma] & \equiv & T[\sigma] \,\mathtt{sat}\, F[\sigma] \\
((y)T \,\mathtt{sat}\, G)[\sigma] & \equiv & ((y)T)[\sigma] \,\mathtt{sat}\, G[\sigma]
\end{array}
$$

Proof: By induction on the stucture of F, using Lemma 3.2. We give four example cases.

The case $[\tau]F$:

$$\begin{array}{lll}
 & (T\ \mathtt{sat}\ [\tau]F)[\sigma] & \\
\equiv & (\bigwedge_{T \xrightarrow{b,\tau} T'} b \to T'\ \mathtt{sat}\ F)[\sigma] & \text{def. of } \mathtt{sat} \\
\equiv & \bigwedge_{T \xrightarrow{b,\tau} T'} b[\sigma] \to (T'\ \mathtt{sat}\ F)[\sigma] & \\
\equiv & \bigwedge_{T[\sigma] \xrightarrow{b[\sigma],\tau} T'[\sigma]} b[\sigma] \to (T'\ \mathtt{sat}\ F)[\sigma] & \text{Lemma 3.2} \\
\equiv & \bigwedge_{T[\sigma] \xrightarrow{b[\sigma],\tau} T'[\sigma]} b[\sigma] \to T'[\sigma]\ \mathtt{sat}\ F[\sigma] & \text{ind. hyp.} \\
\equiv & T[\sigma]\ \mathtt{sat}\ [\tau](F[\sigma]) & \text{def. of } \mathtt{sat} \\
\equiv & T[\sigma]\ \mathtt{sat}\ ([\tau]F)[\sigma] &
\end{array}$$

The case $[c!x]F$:
Because x is a bound variable in $[c!x]F$, we can use α-conversion and assume that it is not changed under σ.

$$\begin{array}{lll}
 & (T\ \mathtt{sat}\ [c!x]F)[\sigma] & \\
\equiv & (\bigwedge_{T \xrightarrow{b,c!} (e,T')} b \to T'\ \mathtt{sat}\ F[e/x])[\sigma] & \text{def. of } \mathtt{sat} \\
\equiv & \bigwedge_{T \xrightarrow{b,c!} (e,T')} b[\sigma] \to (T'\ \mathtt{sat}\ F[e/x])[\sigma] & \\
\equiv & \bigwedge_{T[\sigma] \xrightarrow{b[\sigma],c!} (e[\sigma],T'[\sigma])} b[\sigma] \to (T'\ \mathtt{sat}\ F[e/x])[\sigma] & \text{Lemma 3.2} \\
\equiv & \bigwedge_{T[\sigma] \xrightarrow{b[\sigma],c!} (e[\sigma],T'[\sigma])} b[\sigma] \to T'[\sigma]\ \mathtt{sat}\ (F[e/x])[\sigma] & \text{ind. hyp.} \\
\equiv & \bigwedge_{T[\sigma] \xrightarrow{b[\sigma],c!} (e[\sigma],T'[\sigma])} b[\sigma] \to T'[\sigma]\ \mathtt{sat}\ F[\sigma][e[\sigma]/x] & \\
\equiv & T[\sigma]\ \mathtt{sat}\ [c!x](F[\sigma]) & \text{def. of } \mathtt{sat} \\
\equiv & T[\sigma]\ \mathtt{sat}\ ([c!x]F)[\sigma] &
\end{array}$$

The case $\langle c?\rangle G$:

$$\begin{array}{lll}
 & (T\ \mathtt{sat}\ \langle c?\rangle G)[\sigma] & \\
\equiv & (\bigvee_{T \xrightarrow{b,c?} (y)T'} b \wedge (y)T'\ \mathtt{sat}\ G)[\sigma] & \text{def. of } \mathtt{sat} \\
\equiv & \bigvee_{T \xrightarrow{b,c?} (y)T'} b[\sigma] \wedge ((y)T'\ \mathtt{sat}\ G)[\sigma] & \\
\equiv & \bigvee_{T[\sigma] \xrightarrow{b[\sigma],c?} (y)T'[\sigma]} b[\sigma] \wedge ((y)T'\ \mathtt{sat}\ G)[\sigma] & \text{Lemma 3.2} \\
\equiv & \bigvee_{T[\sigma] \xrightarrow{b[\sigma],c?} (y)T'[\sigma]} b[\sigma] \wedge (y)T'[\sigma]\ \mathtt{sat}\ G[\sigma] & \text{ind. hyp.} \\
\equiv & T[\sigma]\ \mathtt{sat}\ \langle c?\rangle(G[\sigma]) & \text{def. of } \mathtt{sat} \\
\equiv & T[\sigma]\ \mathtt{sat}\ (\langle c?\rangle G)[\sigma] &
\end{array}$$

The case $\exists x F$:
Again we can use α-conversion to ensure that σ leaves unchanged all bound variables

in the process and formulae.

$$
\begin{array}{lll}
 & ((y)T\ \mathtt{sat}\ \exists x F)[\sigma] & \\
\equiv & (\exists z T[z/y]\ \mathtt{sat}\ F[z/x])[\sigma] & \text{def. of } \mathtt{sat} \\
\equiv & \exists z((T[z/y]\ \mathtt{sat}\ F[z/x])[\sigma]) & \\
\equiv & \exists z(T[z/y][\sigma]\ \mathtt{sat}\ F[z/x][\sigma]) & \text{ind. hyp.} \\
\equiv & \exists z(T[\sigma][z/y]\ \mathtt{sat}\ F[\sigma][z/x]) & \\
\equiv & (y)(T[\sigma])\ \mathtt{sat}\ \exists x(F[\sigma]) & \text{def. of } \mathtt{sat} \\
\equiv & ((y)T)[\sigma]\ \mathtt{sat}\ (\exists x F)[\sigma] &
\end{array}
$$

□

Now we can prove the main result about characteristic formulae:

Theorem 3.5 (Characteristic formulae) *For any process term T and formula F, $T\ \mathtt{sat}\ F$ characterizes the satisfaction relation in the sense that for any evaluation ρ*

$$[\![(T\ \mathtt{sat}\ F)[\rho]]\!] = \mathsf{tt} \ \Leftrightarrow\ T[\rho] \models F[\rho]$$

Proof: Follows directly from the above two results.

□

Note that $P[\rho]$ is simply P for any process P and similarly for closed property formulae. So as a corollary to this theorem we have that for any process P and any closed formula F $[\![(P\ \mathtt{sat}\ F)]\!] = \mathsf{tt} \ \Leftrightarrow\ P \models F$.

4 A Modal Proof System

The result of characteristic formulae provides a way of verifying whether a process T satisfies certain modal property F. In this section, we will present a proof system which provides an alternative way of doing such verification. This proof system is very similar to that in [Lar88] for proving properties of "pure processes" but we replace the use of the concrete operational semantics with the symbolic version.

The judgements of the proof system have the form $B \vdash T\!:\!F$, where B is a first order formula over V, which is to be read as "if B is true then T satisfies F". More precisely, it is to be interpreted as for any evaluation ρ if $[\![B[\rho]]\!] = \mathsf{tt}$ then $T[\rho] \models F[\rho]$.

Figure 6 shows the set of proof rules which should be self–explanatory. The only new notation occurs in rule Cons where $B_2 \Rightarrow B_1$ is used to mean: for every evaluation ρ $[\![B_2[\rho]]\!] = \mathsf{tt}$ implies $[\![B_1[\rho]]\!] = \mathsf{tt}$. We could also have used logical implication $B_2 \rightarrow B_1$ except for completeness we need to ensure that all such implications are derivable. We also should point out the special case of rule Cut when $n = 0$, which is $\mathsf{ff} \vdash T\!:\!F$ for any T and F.

Id $\quad \dfrac{}{B \vdash T : B}$ $\qquad\qquad$ Cut $\quad \dfrac{B_1 \vdash T : F \ldots\ldots B_n \vdash T : F}{\bigvee_{1 \le i \le n} B_i \vdash T : F}$ $\quad n \geq 0$

Cons $\dfrac{B_1 \vdash T : F}{B_2 \vdash T : F}$ $\quad B_2 \Rightarrow B_1$ $\qquad\qquad$ Ex $\quad \dfrac{B \vdash T : F}{\exists x B \vdash T : F}$ $\quad x$ does not occur free in T, F

$\vee$ $\quad \dfrac{B \vdash T : F_1}{B \vdash T : F_1 \vee F_2}$ $\qquad\qquad$ $\dfrac{B \vdash T : F_2}{B \vdash T : F_1 \vee F_2}$

$\wedge$ $\quad \dfrac{B \vdash T : F_1 \quad B \vdash T : F_2}{B \vdash T : F_1 \wedge F_2}$

$\langle \tau \rangle$ $\quad \dfrac{B \vdash T' : F}{B \wedge b \vdash T : \langle \tau \rangle F}$ $\quad T \xrightarrow{b, \tau} T'$

$[\tau]$ $\quad \dfrac{B \wedge b_1 \vdash T_1 : F, \ldots\ldots, B \wedge b_n \vdash T_n : F}{B \vdash T : [\tau] F}$ $\quad \{(b_1, T_1), \ldots, (b_n, T_n)\} = \{(b, T') \mid T \xrightarrow{b, \tau} T'\}$

$\langle c! \rangle$ $\quad \dfrac{B \vdash T' : F[e/x]}{B \wedge b \vdash T : \langle c!x \rangle F}$ $\quad T \xrightarrow{b, c!} (e, T')$

$[c!]$ $\quad \dfrac{B \wedge b_1 \vdash T_1 : F[e_1/x], \ldots\ldots, B \wedge b_n \vdash T_n : F[e_n/x]}{B \vdash T : [c!x] F}$

where $\{(b_1, T_1, e_1), \ldots, (b_n, T_n, e_n)\} = \{(b, T', e) \mid T \xrightarrow{b, c!} (e, T')\}$

$\langle c? \rangle$ $\quad \dfrac{B \vdash (y) T' : G}{B \wedge b \vdash T : \langle c? \rangle G}$ $\quad T \xrightarrow{b, c?} (y) T'$

$[c?]$ $\quad \dfrac{B \wedge b_1 \vdash (y_1) T_1 : G, \ldots\ldots, B \wedge b_n \vdash (y_n) T_n : G}{B \vdash T : [c?] G}$

where $\{(b_1, (y_1) T_1), \ldots, (b_n, (y_n) T_n)\} = \{(b, T') \mid T \xrightarrow{b, c?} (y) T'\}$

$\forall$ $\quad \dfrac{B \vdash T : F}{B \vdash (x) T : \forall x F}$ $\quad x$ does not occur free in B

$\exists$ $\quad \dfrac{B \vdash T[e/x] : F[e/x]}{B \vdash (x) T : \exists x F}$ $\qquad\qquad$ α $\quad \dfrac{B \vdash T' : F'}{B \vdash T : F}$ $\quad T' \equiv_\alpha T, F' \equiv_\alpha F$

Figure 6: The Proof Rules

Theorem 4.1 (Soundness) *If $B \vdash T{:}F$ then, for any evaluation ρ, $[\![B[\rho]]\!] = \mathtt{tt}$ implies $T[\rho] \models F[\rho]$.*

Proof: First note that $T[\rho] \models F[\rho]$, for any such substitution ρ, if and only if $[\![(T\ \mathtt{sat}\ F)[\rho]]\!] = \mathtt{tt}$ by Theorem 3.5. So it is sufficient to prove $B \vdash T{:}F$ implies $B \Rightarrow T\ \mathtt{sat}\ F$. Therefore we only need to show that the rules of the proof system preserve soundness in this sense. Each case follows from simple identities involving $\Rightarrow$ and the logical operators. For example the soundness of the $[c?]$ rule relies on the following facts:

1. $B \wedge b \Rightarrow F$ if and only if $B \Rightarrow (b \rightarrow F)$,
2. $B \Rightarrow \bigwedge_{i \in S} F_i$ if and only if $B \Rightarrow F_i$ for all $i \in S$.

To prove the soundness of $\forall$ rule, we have to show that if $B \Rightarrow T\ \mathtt{sat}\ F$ and x does not occur free in B then $B \Rightarrow (x)T\ \mathtt{sat}\ \forall xF$. Notice that $(x)T\ \mathtt{sat}\ \forall xF \equiv \forall x T\ \mathtt{sat}\ F$. The result now follows since if x does not occur free in B then $B \Rightarrow T\ \mathtt{sat}\ F$ implies $B \Rightarrow \forall x T\ \mathtt{sat}\ F$.

To prove the soundness of $\exists$ rule, we have to show that if $B \Rightarrow T[e/x]\ \mathtt{sat}\ F[e/x]$ then $B \Rightarrow (x)T\ \mathtt{sat}\ \exists xF$. Here $T[e/x]\ \mathtt{sat}\ F[e/x]$ is logically equivalent to $(T\ \mathtt{sat}\ F)[e/x]$ which in turn implies $\exists x T\ \mathtt{sat}\ F$.

□

The completeness of the system depends on the following result:

Lemma 4.2 *For any term T and formula F, $T\ \mathtt{sat}\ F \vdash T{:}F$.*

Proof: By induction on the size of F. We give four cases.

In the case $\langle c!x\rangle F$, because of the construction of $T\ \mathtt{sat}\ \langle c!x\rangle F$, we have to show that $\bigvee_{T \xrightarrow{b,c!} (e,T')} b \wedge T'\ \mathtt{sat}\ F[e/v] \vdash T{:}\langle c!x\rangle F$. By the induction hypothesis, for all $T \xrightarrow{b,c!} (e,T')$, $T'\ \mathtt{sat}\ F[e/x] \vdash T'{:}F[e/x]$, so $b \wedge T'\ \mathtt{sat}\ F[e/x] \vdash T{:}\langle c!x\rangle F$ by rule $\langle c!\rangle$. So $\bigvee_{T \xrightarrow{b,c!} (e,T')} b \wedge T'\ \mathtt{sat}\ F[e/x] \vdash T{:}\langle c!x\rangle F$ by rule Cut.

When it is $[c?]G$, by the definition of $T\ \mathtt{sat}\ [c?]G$, we have to show that $\bigwedge_{T \xrightarrow{b,c?} (y)T'} b \rightarrow (y)T'\ \mathtt{sat}\ G \vdash T{:}[c?]G$. By the induction hypothesis, for all R such that $T \xrightarrow{b_R,c?} (y)R$, $(y)R\ \mathtt{sat}\ G \vdash (y)R{:}G$, then $(\bigwedge_{T \xrightarrow{b,c?} (y)T'} b \rightarrow (y)T'\ \mathtt{sat}\ G) \wedge b_R \vdash (y)R{:}G$ by rule Cons because $(\bigwedge_{T \xrightarrow{b,c?} (y)T'} b \rightarrow (y)T'\ \mathtt{sat}\ G) \wedge b_R \Rightarrow (y)R\ \mathtt{sat}\ G$. Then by rule $[c?]$, $\bigwedge_{T \xrightarrow{b,c?} (y)T'} b \rightarrow (y)T'\ \mathtt{sat}\ G \vdash T{:}[c?]G$.

When it is $\forall xF$, by the definition of $(y)T\ \mathtt{sat}\ \forall xF$, we have to show that $\forall z.T[z/y]\ \mathtt{sat}\ F[z/x] \vdash (y)T{:}\forall xF$. By the induction hypothesis,

$$T[z/y]\ \mathtt{sat}\ F[z/x] \vdash T[z/y]{:}F[z/x]$$

Because $\forall z.T[z/y]\ \mathtt{sat}\ F[z/x] \Rightarrow T[z/y]\ \mathtt{sat}\ F[z/x]$, by rule Cons we have

$$\forall z.T[z/y]\ \mathtt{sat}\ F[z/x] \vdash T[z/y]{:}F[z/x]$$

Then apply rule $\forall$ we have $\forall z.T[z/y]\,\mathtt{sat}\,F[z/x] \vdash (z)T[z/y]{:}\forall z F[z/x]$. Now $(z)T[z/y]{:}\forall z F[z/x]$ can be α converted into $(y)T{:}\forall x F$.

When it is $\exists x F$, by the definition of $(y)T\,\mathtt{sat}\,\exists x F$, we have to show that $\exists z.T[z/y]\,\mathtt{sat}\,F[z/x] \vdash (y)T{:}\exists x F$. By the induction hypothesis,

$$T[z/y]\,\mathtt{sat}\,F[z/x] \vdash T[z/y]{:}F[z/x]$$

Applying rule $\exists$ with e and x equal to z we have $T[z/y]\,\mathtt{sat}\,F[z/x] \vdash (z)T[z/y]{:}\exists z F[z/x]$. Applying rule Ex we have $\exists z.T[z/y]\,\mathtt{sat}\,F[z/x] \vdash (z)T[z/y]{:}\exists z F[z/x]$, then by rule α $\exists z.T[z/y]\,\mathtt{sat}\,F[z/x] \vdash (y)T{:}\exists x F$.

□

The completeness of the proof system follows immediately from this lemma.

Theorem 4.3 (Completeness) *If* $[\![B[\rho]]\!] = \mathtt{tt}$ *implies* $T[\rho] \models F[\rho]$ *for any evaluation* ρ *then* $B \vdash T{:}F$.

Proof: From Theorem 3.5 we know that for any evaluation $T[\rho] \models F[\rho]$ if and only if $[\![(T\,\mathtt{sat}\,F)[\rho]]\!] = \mathtt{tt}$. So, if $[\![B[\rho]]\!] = \mathtt{tt}$ implies $T[\rho] \models F[\rho]$ for any evaluation ρ then $B \Rightarrow T\,\mathtt{sat}\,F$. By Lemma 4.2, we have $T\,\mathtt{sat}\,F \vdash T{:}F$ and then $B \vdash T{:}F$ follows from rule Cons. □

In [HL92], the notion of *symbolic bisimulation* equivalence is introduced for process terms with free value variables. This completeness result enables us to show that the modal logic together with the proof system gives a natural characterization of the symbolic bisimulation equivalence. For convenience we define this equivalence by its key characterization instead of giving the original definition; the reader is referred to [HL92] for motivation and details.

Definition 4.4 *For terms* T, U *and boolean expression* B, T *is said to be symbolic bisimulation equivalent to* U *under* B, *written as* $T \sim^B U$, *if for every evaluation* ρ, $[\![B[\rho]]\!] = \mathtt{tt}$ *implies* $T[\rho] \sim U[\rho]$.

Lemma 4.5 *Let* T, U *be two process terms,* B *be a boolean expression. If* $T \sim^B U$ *then for every model formulae* F, $B \vdash T{:}F \;\Leftrightarrow\; B \vdash U{:}F$.

Proof: We assume $T \sim^B U$ and $B \vdash T{:}F$ for some F, and we will show that $B \vdash U{:}F$. By the completeness of the proof system it is sufficient to show that for any evaluation if $[\![B[\rho]]\!] = \mathtt{tt}$ then $U[\rho] \models F[\rho]$ From $B \vdash T{:}F$ and the soundness of the proof system, we have $T[\rho] \models F[\rho]$. Because $T \sim^B U$ and $[\![B[\rho]]\!] = \mathtt{tt}$ it follows that $T[\rho] \sim U[\rho]$, and by Theorem 2.2 $T[\rho] \sim U[\rho]$ and $T[\rho] \models F[\rho]$ implies $U[\rho] \models F[\rho]$. □

The property that $B \vdash T{:}F \;\Leftrightarrow\; B \vdash U{:}F$, for every modal formula F, alone is not sufficient to characterize $T \sim^B U$. Let us define $T \equiv^B U$ to be for every model formula F $B \vdash T{:}F \;\Leftrightarrow\; B \vdash U{:}F$. Then if this characterisation were true then $\equiv^B$ should coinside with $\sim^B$. But in general it is not true that $\equiv^B \subseteq \equiv^{B'}$ for any $B' \Rightarrow B$. So $\equiv^B$ is not $\sim^B$ simply because it does not have an obvious property enjoyed by $\sim^B$. However a slight generalization of $\equiv^B$ does provide a characterisation:

Theorem 4.6 *Let T, U be any two process and B be a boolean expression. Then $T \sim^B U$ if and only if for every modal formula F, $B' \vdash T{:}F \Leftrightarrow B' \vdash U{:}F$ for any B' such that $B' \Rightarrow B$.*

Proof:
One direction follows immediately from the above Lemma since if $B' \Rightarrow B$ and $T \sim^B U$ then $T \sim^{B'} U$. To prove the converse suppose $B' \vdash T{:}F \Leftrightarrow B' \vdash U{:}F$ for any modal formula F for any B' such that $B' \Rightarrow B$. We will show that $T \sim^B U$. For that we only need to show that if $[\![B[\rho]]\!] = \mathtt{tt}$ for any evaluation ρ then $T[\rho] \sim U[\rho]$, and by Theorem 2.2 we only need to show that for any closed F, $T[\rho] \models F \Leftrightarrow U[\rho] \models F$.

So suppose $[\![B[\rho]]\!] = \mathtt{tt}$ $T[\rho] \models F$; we will show that $U[\rho] \models F$. Because $T \,\mathtt{sat}\, F \vdash T{:}F$ by Lemma 4.2 and $B \wedge T \,\mathtt{sat}\, F \Rightarrow B$, so $B \wedge T \,\mathtt{sat}\, F \vdash T{:}F$. Thus $B \wedge T \,\mathtt{sat}\, F \vdash U{:}F$. Now because $T[\rho] \models F[\rho]$ so $[\![(T \,\mathtt{sat}\, F)[\rho]]\!] = [\![T[\rho] \,\mathtt{sat}\, F[\rho]]\!] = \mathtt{tt}$, and moreover $[\![B]\!] = \mathtt{tt}$, so $[\![(B \wedge T \,\mathtt{sat}\, F)[\rho]]\!] = \mathtt{tt}$, so $U[\rho] \models F$ by the soundness of the proof system. □

5 Conclusion

In this paper we have suggested a first–order modal logic for defining properties of message passing processes and shown how at least some of the techniques which apply to the use of propositional modal logic and pure process algebras, [Lar88, Sti87] can be extended to this setting. The first result is the definition of a characteristic formula for each process and property a formula in the first–order modal logic, which is logically equivalent to $\mathtt{tt}$ if and only if the process enjoys the property. The second is a sound and complete proof system for proving that a specific process enjoys a specific property. Of course this does not make the task of proving such a statement trivial. Moreover the proof system is modulo the underlying first–order theory of the messages being transmitted and thus any implementation of our proof system will require access to a proof system for this underlying theory. However most interesting properties of message passing systems will depend on properties of the underlying message domain and thus reasoning about the messages can not be avoided. Our approach merely provides a convenient and powerful framework for organising the interaction between that part of the proof concerned with processes and that concerned with properties of the messages. But it is conceived so that any implementation of the proof system can take advantage of a range of different theorem provers for the message domain. Indeed it is reasonable to assume that different application areas will have a range of different message domains and each of these could have a specialised theorem prover associated with it. Our proof system could be tailored to all of these application areas simply by using the associated theorem prover for the underlying message domain.

Much work remains to be done. First our results should be extended to recursive formulae as in [Lar88]. We do not envisage any major problems but the details have yet to be worked out. Secondly we should develop a prototype implementation of the proof system. The proof system relies heavily on symbolic transitions but

luckily these have already been implemented for a language very similar to ours in [Lin93]. Consequently using this groundwork a prototype implementation should be straightforward.

References

[CPS88] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench. University of Edinburgh, Scotland, 1988.

[GLZ89] J.C. Godskesen, K.G. Larsen, and M. Zeeberg. Tav — tools for automatic verification — users manual. Technical Report R 89-19, Department of Mathematics and Computer Science, Aalborg University, 1989. Presented at workshop on Automatic Methods for Finite State Systems, Grenoble, France, Juni 1989.

[GS86] S. Graf and J. Sifakis. A logic for the description of non–deterministic programs and their properties. *Information and Control*, 68(1–3), 1986.

[Hen91] M. Hennessy. A proof system for communicating processes with value–passing. *Formal Aspects of Computer Science*, 3:346–366, 1991.

[HL92] M. Hennessy and H. Lin. Symbolic bisimulation. Technical Report Technical Report 1/92, School of Congnitive and Computing Sciences, University of Sussex, 1992.

[IT92] A. Ingolfsdottir and B. Thomsen. Semantic models for ccs with values. Technical Report 63, Programming Methodology Group, Chalmers University of Technology, 1992. In Proceedings of the Workshop on Concurrency.

[Lar88] K.G. Larsen. Proof systems for Hennessy–Milner logic with recursion. *Lecture Notes In Computer Science, Springer Verlag*, 299, 1988. in Proceedings of 13th Colloquium on Trees in Algebra and Programming 1988.

[Lin93] H. Lin. A verification tool for value–passing processes. Technical report, School of Congnitive and Computing Sciences, University of Sussex, 1993.

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

[MPW92] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 1992. to appear.

[SI91] B. Steffen and A. Ingolfsdottir. Characteristic formulae for processes with divergence. Technical Report Technical Report 1/91, School of Congnitive and Computing Sciences, University of Sussex, 1991.

[Sti87] C. Stirling. Modal logics for communicating systems. *Theoretical Computer Science*, (311-347), 1987.