Local model checking for a value-based modal μ-calculus.

J. Rathke M. Hennessy

June 26, 1996

Abstract

We first present a first-order modal μ-calculus which uses parameterised maximal and minimal fixpoints to describe safety and liveness properties of processes which communicate values along ports. Then, using a class of models based on symbolic graphs, we give a local model checking proof system for this logic. The system is a natural extension of existing proof systems and although it is sound it is incomplete in general.

We prove two different completeness results for certain sub-logics. The first is for safety properties alone, where only parameterised maximal fixpoints are used while the second allows both kinds of fixpoints but restricts the use of parameterisation.

1 Introduction

Model checking refers to the verification that a system, represented as a point t of some model, satisfies some property expressed in a particular property logic. The modal μ-calculus is one, very expressive, logic which is often used to express properties of systems; in addition to the usual boolean connectives it contains modal operators for describing the possible actions of a system together with maximal and minimal fixpoints for describing safety and liveness properties. In papers such as [7, 11, 12, 13] sound and complete proof methods are developed for checking that formulae from this calculus are satisfied by terms from a pure process calculus, such as CCS [10], represented as points from a finite labelled transition system. The object of the this paper is to generalise these methods to *value-passing* process calculi, which are used to describe distributed systems which manipulate and interchange data along communication channels.

The approach we take is to generalise Winskel's tag set proof method [13] for pure processes. We first give a brief outline of this method: Consider the process

$$P \Leftarrow a.a.P$$

which can perform the abstract action a twice and then return to its original state. Evidently this can perform an infinite sequence of a actions and this property can be expressed in the modal μ-calculus with the formula

$$\nu X.\langle a\rangle X.$$

Let us try to formally establish that P satisfies this formula, which we write as

$$P : \nu X.\langle a \rangle X. \tag{1}$$

Because we can unwind recursive definitions this can be reduced to establishing

$$a.a.P : \langle a \rangle \nu X.\langle a \rangle X.$$

But this reduction does not lead very far; the only applicable rule now is that associated with the modality $\langle a \rangle$ which reduces this judgement to

$$a.P : \nu X.\langle a \rangle X.$$

A further unwinding and application of the modality rule will only reduce this to the original judgement (1).

The tag set method records the points that have already been checked so that verifying

$$P : \nu X.\langle a \rangle X$$

is reduced, by the *(tagged) Recur ive Unwinding rule*, to checking the judgement

$$P : \langle a \rangle \nu X.[P]\langle a \rangle X.$$

The tag $[P]$ indicates that there is no further need to check the process P against this formula. Now by an application of the modality rule and a further application of the tagged recursive unwinding rule we reduce our proof obligation to establishing

$$P : \nu X.[P, a.P]\langle a \rangle X.$$

We can now terminate the checking routine since P is in the tag set, i.e. P has already been visited.

We generalise this approach to model checking by

- replacing pure process terms and their associated labelled transition systems with value-passing process terms and their symbolic transition graphs, [5], and
- by generalising the property logic to a first-order modal μ-calculus.

Let us first give an indication of the nature of this generalisation and why the various extensions to the proof system are required. The first-order modal μ-calculus allows the use of boolean expressions from some unspecified boolean language, which includes first order quantification, and, as in [6], the propositional modal operators $\langle a \rangle F$, $[a]F$ are

> whenever a value is input on the channel a a larger value can subsequently be output the channel b.

This property can be formalised by the modal formula

$$[a?]\,\forall x\langle b!y\rangle(y > x)$$

and the only way to establish the judgement

$$P : [a?]\,\forall x\langle b!y\rangle(y > x)$$

is to reduce it, using rules associated with the modality $[a?]$ and universal quantification, to establishing

$$b!(x+1).P : \langle b!y\rangle(y > x),$$

a judgement involving open process terms.

The presence of these open terms engenders further complications in the allowed judgements. For example consider the judgement

$$Q : [a?]\,\forall x\langle b!y\rangle(even\,y)$$

where Q is the process defined by

$$Q \Leftarrow a?x.\,even(x) \mapsto b!x.Q,\ b!(x+1).Q.$$

Using the natural proof rule for the modality $[a?]$ this can be reduced to the judgement

$$(even(x) \mapsto b!x.Q,\ b!(x+1).Q) : \langle b!y\rangle(even\,y)$$

which can only be reduced to the *relativised* judgements

$$\textit{on the assumption that } x \textit{ is even},\ b!x.Q : \langle b!y\rangle(even\,y)$$

and

$$\textit{on the assumption that } x \textit{ is odd},\ b!(x+1).Q : \langle b!y\rangle(even\,y).$$

In short our model checking procedur0.24Tf9.oeTd(b)Tj/R1260.24Tf5.039840Td(!(60.24r.4999.6(j4.56016 ability of a process to output an infinite sequence of values along the channel a which are alternatively even and odd.

In order to give an outline of the rules associated with these formulae consider the process

$$P \Leftarrow a!0.a!1.P$$

and the judgement

$$\mathbf{true} \vdash P : A.(0/z) \tag{2}$$

The (tagged) Recursive Unwinding rule can not be applied directly here as the formula is a recursively defined formula applied to a specific value 0. The main new rule in the proof system is a *generali ation* rule which enables us to generalise a judgement about specific value, such as (2), to a more general statement about arbitrary values, to which the recursive unwinding rule can be applied:

$$\text{App} \quad \frac{B \vdash t : A}{B[\tilde{e}/\tilde{z}] \vdash t : A.(\tilde{e}/\tilde{z})}$$

Using this rule, with B instantiated to $even(z)$, the judgement (2) can be reduced to establishing

$$even(z) \quad \vdash \quad P : A, \tag{3}$$

since $even(0)$ implies **true**. Here the Recursive Unwinding rule can be applied. But because the judgements are *relativi ed* the tags will now have to consist of process terms together with boolean expressions. Thus the judgement (3) can be reduced to the judgement

$$even(z) \quad \vdash \quad a!1.P : \langle a!x\rangle(x = z\, mod\, 2) \;\wedge\; C.(z+1/z) \tag{4}$$

where C is the abstraction $\nu X[\mathcal{A}]\langle a!x\rangle(x = z\, mod\, 2) \wedge X.(z+1/z)$ and $\mathcal{A}$ is the tag set $\{(even(z), P)\}$. Using standard rules this in turn leads to two judgements

$$even(z) \vdash a!1.P : (0 = z\, mod\, 2) \quad \text{and} \quad even(z) \vdash a!1.P : C.(z+1/z).$$

The first is easily established as it is simply a property of the data space but the second requires a further generalisation, a use of the rule App, in order to once more unwind the recursively defined property. This time the appropriate instantiation of the boolean B is $odd(z+1)$ as $odd(z+1)$ implies $even(z)$ and therefore (4) can be reduced to the judgement

$$odd(z) \vdash a!1.P : C$$

After another application of the Recursive Unwinding rule, followed by an application of standard rules we obtain the two judgements

$$odd(z) \vdash P : (1 = z\, mod\, 2) \quad \text{and} \quad odd(z) \vdash P : D.(z+1/z)$$

where D is the abstraction $\nu X[\mathcal{B}]\langle a!x\rangle(x = z\, mod\, 2) \wedge X.(z+1/z)$ and $\mathcal{B}$ is the tag set $\{(even(z), P),\ (odd(z), a!1.P)\}$. The first is straightforward to establish while a final use of generalisation reduces the latter to $even(z) \vdash P : D$. Here the deduction terminates as $(even(z), P)$ is in the tag set $\mathcal{B}$.

The use of the rule App, and in particular the choice of the appropriate instantiation of the Boolean B, is essential in the development of such proofs and it is easy to see that a bad choice of instantiation will hinder the successful termination of a proof. The use of boolean expressions in tag sets can also lead to difficulties in generating successfully terminating proofs but their use is essential; if we only used process terms in the tag sets the resulting proof system would be unsound. For example it would be easy to establish the judgement

$$\mathbf{true} \vdash P : (\nu X \langle a!x \rangle (x = z) \wedge X.(z + 1/z))(0/z)$$

where P is the process

$$P \Leftarrow a!0.P,$$

and this is obviously untrue as the property states that P can output an infinite sequence of increasing values.

This completes our overview of the proof system. It is formally defined in Section 3 which also contains a soundness theorem. However in order to abstract away from the details of a specific value-passing language we represent processes using terms over *symbolic transition systems*, [5]. These graphs are described in Section 2 where they are also used as models for interpreting formulae from our first order modal μ-calculus.

We now give an overview of the remaining sections of the paper, which address completeness issues. One can not

Semantically P satisfies the property F but we will show that

$$\textbf{true} \vdash P : F$$

is not derivable in our system. The argument will be quite simple: for any

$$F ::= B \mid F \vee F \mid F \wedge F \mid \langle \tau \rangle F \mid [\tau]F | \langle c!x \rangle F \mid [c!x]F \mid \langle c? \rangle G \mid [c?]G \mid A.(\tilde{e}/\tilde{z})$$

$$G ::= \exists x.F \mid \forall x.F$$

$$A ::= X \mid \nu X[\mathcal{A}]F \mid \mu X[\mathcal{A}]F$$

Figure 1: Grammar for the logic

but abstractions which use them may only be applied to *re tricted parameter* ; parameters which are either recursion variables or do not use any recursion variables. Thus $A.(z+1/z)$ is forbidden but formulae such as $\langle a!x \rangle A(x+1/z)$ are allowed. The proof of this completeness theorem also uses the method of characteristic boolean expressions although the details are somewhat more complicated.

This sub-language of property formulae may seem overly restricted but we also give some examples to show that it is quite expressive. In particular in Section 6 we give an example of a process which inputs an infinite sequence of integers and continually outputs the greatest one received so far. This property can be expressed using restricted parameters and we also outline a proof that the process enjoys this property.

The paper ends with

variable called $RVar$; note that we generally use lower case $x, y, \ldots$ to denote variables from $DVar$ and upper case $X, Y, \ldots$ to denote recursion variables. F ranges over the main syntactic category of modal formulae while G ranges over quantified formulae,

$$
\begin{array}{l}
n \xrightarrow{b,\tau} n' \text{ implies } n_\sigma \xrightarrow{b\sigma,\tau} n'_\sigma \\
n \xrightarrow{b,c!e} n' \text{ implies } n_\sigma \xrightarrow{b\sigma,c!e\sigma} n'_\sigma \\
n \xrightarrow{b,c?x} n' \text{ implies } n_\sigma \xrightarrow{b\sigma,c?} (x)n'_\sigma
\end{array}
$$

Figure 2: Symbolic operational semantics for open terms.

$$
\begin{array}{lll}
t \xrightarrow{b,\tau} t' & \text{implies } t\delta \xrightarrow{\tau} t'\delta & \text{if } \llbracket b\delta \rrbracket = \mathbf{tt} \\
t \xrightarrow{b,c!e} t' & \text{implies } t\delta \xrightarrow{c!v} t'\delta & \text{if } \llbracket b\delta \rrbracket = \mathbf{tt} \text{ and } \llbracket e\delta \rrbracket = v \\
t \xrightarrow{b,c?} (x)t' & \text{implies } t\delta \xrightarrow{c?} (x)t'\delta & \text{if } \llbracket b\delta \rrbracket = \mathbf{tt}
\end{array}
$$

Figure 3: Concrete operational semantics for closed terms.

and we use $t, u, \ldots$ to range over terms. These terms are the expressions that we will be using in the tag sets. In actual fact a tag set $\mathcal{A}$ will contain pairs (B, t) where B is some boolean expression.

We define pairs (t, δ) to be *clo ed term* whenever δ is an evaluation and for convenience it is written as $t\delta$. We let $p, q, \ldots$

$$
\begin{array}{lcl}
[\![B]\!]\rho\delta & = & \begin{cases} \mathcal{CT}(\mathcal{G}) & \text{If } \delta \models B \\ \emptyset & \text{Otherwise} \end{cases} \\
[\![F \wedge F']\!]\rho & = & [\![F]\!]\rho \cap [\![F']\!]\rho \\
[\![F \vee F']\!]\rho & = & [\![F]\!]\rho \cup [\![F']\!]\rho \\
[\![\langle\tau\rangle F]\!]\rho\delta & = & \left\{p \mid \exists p' \cdot p \xrightarrow{\tau} p' \text{ and } p' \in [\![F]\!]\rho\delta\right\} \\
[\![[\tau]F]\!]\rho\delta & = & \left\{p \mid \forall p' \cdot p \xrightarrow{\tau} p' \text{ implies } p' \in [\![F]\!]\rho\delta\right\} \\
[\![\langle c!x\rangle F]\!]\rho\delta & = & \left\{p \mid \exists p', v \cdot p \xrightarrow{c!v} p' \text{ and } p' \in [\![F[v/x]]\!]\rho\delta\right\} \\
[\![[c!x]F]\!]\rho\delta & = & \left\{p \mid \forall p', v \cdot p \xrightarrow{c!v} p' \text{ implies } p' \in [\![F[v/x]]\!]\rho\delta\right\} \\
[\![\langle c?\rangle G]\!]\rho\delta & = & \left\{p \mid \exists (x)p' \cdot p \xrightarrow{c?} (x)p' \text{ and } (x)p' \in [\![G]\!]\rho\delta\right\} \\
[\![[c?]G]\!]\rho\delta & = & \left\{p \mid \forall (x)p' \cdot p \xrightarrow{c?} (x)p' \text{ implies } (x)p' \in [\![G]\!]\rho\delta\right\} \\
[\![\exists x.F]\!]\rho\delta & = & \{(y)p \mid \exists v \in Val \cdot ((y)p)v \in [\![F[v/x]]\!]\rho\delta\} \\
[\![\forall x.F]\!]\rho\delta & = & \{(y)p \mid \forall v \in Val \cdot ((y)p)v \in [\![F[v/x]]\!]\rho\delta\} \\
[\![A.(\tilde{e}/\tilde{z})]\!]\rho\delta & = & ([\![A]\!]\rho)\delta[\tilde{e}/\tilde{z}] \\
[\![X]\!]\rho & = & \rho(X) \\
[\![\mu X[\mathcal{A}]F]\!]\rho & = & \mu f.(\lambda\delta.[\![F]\!]\rho[f/X]\delta \setminus \lambda\mathcal{A}) \\
[\![\nu X[\mathcal{A}]F]\!]\rho & = & \nu f.(\lambda\delta.[\![F]\!]\rho[f/X]\delta \cup \lambda\mathcal{A})
\end{array}
$$

where $\lambda\mathcal{A}(\delta) = \{t\delta \mid (B,t) \in \mathcal{A} \text{ and } \delta \models B\}$ and $\setminus$, $\cup$ and $\cap$ denote pointwise set difference, union and intersection respectively.

Figure 4: Interpretation of logic in a model $\mathcal{T}(\mathcal{G})$.

Lemma 1 *(Reduction Lemma)*

Let $L = V \rightarrow \mathcal{P}T$ be a complete lattice and let $\varphi : L \rightarrow L$ be a monotone functional. Let $B \subseteq V$ and write $f \leq_B g$ to mean $\forall v \in B.f(v) \subseteq g(v)$. Then for any $f \in L$,

$$f \leq_B \nu x.\varphi(x) \text{ iff } f \leq_B \varphi(\nu x.(\varphi(x) \cup \lambda[B]f))$$

where $\lambda[B]f(v) = f(v)$ if $v \in B$ and i empty otherwi e.

Proof. Straightforward generalisation of the proof in [13]. □

It is interesting to note at this point that the corresponding theorem for least fixpoints

$$f \leq_B \mu x.\varphi(x) \text{ iff } f \leq_B \varphi(\mu x.(\varphi(x) \setminus \lambda[B]f))$$

does not hold. To see how this fails consider the following example:

Using the sets $T, V, B = \{a, b\}$ and letting $\lambda\emptyset$ denote the constant empty function we define $\phi(\lambda\emptyset) = f$ where $f(a) = \{a\}, f(b) = \emptyset$ and $\phi(g) = d$ where $d(a) = d(b) = \{a\}$ whenever $g \neq \lambda\emptyset$. It is easy to see that $d = \mu x.\phi(x)$, but notice that $d \not\leq \phi(\mu x.(\phi(x) \setminus d)) = f$.

Lemma 2 *If $(B', t) \notin \mathcal{A}$ for all B' then*

$$t \models_B \nu X[\mathcal{A}]F \text{ iff } t \models_B F[\nu X[\mathcal{A}, (B,t)]F/X].$$

Proof. Uses Lemma 1 and simple properties about substitutions. □

3 The Proof System

We now present a proof system for verifying whether a formula of the logic holds at a particular point of the model. The system is similar in style to those of [6, 4] in that the proof rules carry side conditions which leave proof obligations of checking implication in some language of boolean conditions and of calculating

$$Id \quad \frac{}{B \vdash t : B}$$

$$Cut \quad \frac{B_1 \vdash t : F, \ldots, B_n \vdash t : F}{\bigvee_{1 \leq i \leq n} B_i \vdash t : F}$$

$$Cons \quad \frac{B_1 \vdash t : F}{B_2 \vdash t : F} \quad (B_2 \models B_1)$$

$$Ex \quad \frac{B \vdash t : F}{\exists x.B \vdash t : F} \quad (x \notin fv(t, F))$$

$$\alpha \quad \frac{B \vdash t' : F'}{B \vdash t : F} \quad (t' \equiv_\alpha t,\ F' \equiv_\alpha F)$$

$$\wedge \quad \frac{B \vdash t : F_1 \quad B \vdash t : F_2}{B \vdash t : F_1 \wedge F_2}$$

$$\vee_L \quad \frac{B \vdash t : F_1}{B \vdash t : F_1 \vee F_2}$$

$$\vee_R \quad \frac{B \vdash t : F_2}{B \vdash t : F_1 \vee F_2}$$

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

$$[\tau] \quad \frac{B \wedge b_1 \vdash t_1 : F, \ldots, B \wedge b_n \vdash t_n : F}{B \vdash t : [\tau]F}$$

$$\text{where } \{(b_1, t_1), \ldots, (b_n, t_n)\} = \{(b, t') \mid t \xrightarrow{b,\tau} t'\}$$

$$\langle c! \rangle \quad \frac{B \vdash t' : F[e/x]}{B \wedge b \vdash t : \langle c!x \rangle F} \quad t \xrightarrow{b,c!e} t'$$

$$[c!] \quad \frac{B \wedge b_1 \vdash t_1 : F[e_1/x], \ldots, B \wedge b_n \vdash t_n : F[e_n/x]}{B \vdash t : [c!x]F}$$

$$\text{where } \{(b_1, t_1, e_1), \ldots, (b_n, t_n, e_n)\} = \{(b, t', e) \mid t \xrightarrow{b,c!e} t'\}$$

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

$$[c?] \quad \frac{B \wedge b_1 \vdash (y_1)t_1 : F, \ldots, B \wedge b_n \vdash (y_n)t_n : G}{B \vdash t : [c?]G}$$

$$\text{where } \{(b_1, (y_n)t_1), \ldots, (b_n, (y_n)t_n)\} = \{(b, (y)t') \mid t \xrightarrow{b,c?} (y)t'\}$$

$$\forall \quad \frac{B \vdash t : F}{B \vdash (x)t : \forall x,i}$$

$$\text{App}\quad \frac{B \vdash t : A}{B[\tilde{e}/\tilde{z}] \vdash t : A.(\tilde{e}/\tilde{z})}$$

$$\nu_0 \quad \frac{}{B \vdash t : \nu X[\mathcal{A}]F} \quad (B,t) \in \mathcal{A} \qquad\qquad \nu_1 \quad \frac{B \vdash t : F[\nu X[\mathcal{A} \cup (B,t)]F/X]}{B \vdash t : \nu X[\mathcal{A}]F}$$

$$\mu \quad \frac{B \vdash t : F[\mu X[\mathcal{A} \cup (B,t)]F/X]}{B \vdash t : \mu X[\mathcal{A}]F} \quad \forall B'.(B',t) \in \mathcal{A} \text{ implies } B \wedge B' \models \mathbf{ff}$$

Figure 6: Fixpoint rules.

$z) \vee X.(z+1/z)$, states ‘there exists an output on channel a of a value at least as large as z’. We instantiate z at 0 to get the formula which simply reads “there exists an output, on channel a,ulae z

Induction gives $B \vdash t : F[F^{k'}/X]$ for all $k' \geq k-1$, but this is just $B \vdash t : F^{k'+1}$. Otherwise F' must be of the form $\mu Y[\mathcal{A}']F''$ and we have, writing A for $\mu X[\mathcal{A}]F$,

$$\mu \quad \frac{B \vdash t : (F''[A/X])[\mu Y[\mathcal{A}', (B,t)]F''[A/X]/Y]}{B \vdash t : \mu Y[\mathcal{A}']F''[A/X]}.$$

We can reorder the premis to read $B \vdash t : (F''[\mu Y[\mathcal{A}', (B,t)]F''/Y])[A/X]$ and hence by induction we get

$$B \vdash t : (F''[\mu Y[\mathcal{A}', (B,t)]F''/Y])[F^{k'}/X].$$

Again, by reordering and using the μ-rule on Y we get $B \vdash t : \mu Y.[\mathcal{A}']F''[$

$$\begin{array}{rcl}
[\![B]\!]\xi & = & B \\
[\![Q_1 \vee Q_2]\!]\xi & = & [\![Q_1]\!]\xi \vee [\![Q_2]\!]\xi
\end{array}$$

Proof. Similar to the proof in [6] though we use well-founded induction on tag restricted formulae. The only case of interest here is that for fixpoints. If t appears in the tag set then rule ν_0 gives the result. Otherwise, by induction we know that

$$[\![t \,\mathrm{sat}\, F[\nu X[\mathcal{A}']F/X]]\!]\eta \vdash t : F[\nu X[\mathcal{A}']F/X]$$

where $\mathcal{A}' = \mathcal{A} \cup (t \,\mathrm{sat}\, \nu X[\mathcal{A}]F, t)$. But $[\![t \,\mathrm{sat}\, F[\nu X[\mathcal{A}']F/X]]\!]\eta$ is easily seen to be $[\![t \,\mathrm{sat}\, \nu X[\mathcal{A}]F]\!]\eta$ so by rule ν_1

semantics. We write $t \models^{s}_{\rho,B\widehat{\varepsilon}} F$ iff $t \in [\![F]\!]_s \rho B\widehat{\varepsilon}$.

Pro osition 10 *If F (not nece arily recur ion clo ed) ha empty tag et then $t \models_{\rho,B\widehat{\varepsilon}}$ F iff $t \models^{s}_{\rho^*,B\widehat{\varepsilon}} F$ (where $t \in \rho^*$*

$$\begin{array}{lcl}
\varepsilon \triangleright t \,\mathsf{sat}\, B & = & B[\varepsilon(\tilde{z})/\tilde{z}] \\
\varepsilon \triangleright t \,\mathsf{sat}\, F_1 \wedge F_2 & = & \varepsilon \triangleright t \,\mathsf{sat}\, F_1 \wedge \varepsilon \triangleright t \,\mathsf{sat}\, F_2 \\
\varepsilon \triangleright t \,\mathsf{sat}\, F_1 \vee F_2 & = & \varepsilon \triangleright t \,\mathsf{sat}\, F_1 \vee \varepsilon \triangleright t \,\mathsf{sat}\, F_2 \\
\varepsilon \triangleright t \,\mathsf{sat}\, \langle \tau \rangle F & = & \bigvee_{t \xrightarrow{b',\tau} t'} b' \wedge \varepsilon \triangleright t' \,\mathsf{sat}\, F \\
\varepsilon \triangleright t \,\mathsf{sat}\, [\tau] F & = & \bigwedge_{t \xrightarrow{b',\tau} t'} b' \rightarrow \varepsilon \triangleright t' \,\mathsf{sat}\, F \\
\varepsilon \triangleright t \,\mathsf{sat}\, \langle c!x \rangle F & = & \bigvee_{t \xrightarrow{b',c!e} t'} b' \wedge \varepsilon \triangleright t' \,\mathsf{sat}\, F[e/x] \\
\varepsilon \triangleright t \,\mathsf{sat}\, [c!x] F & = & \bigwedge_{t \xrightarrow{b',c!e} t'} b' \rightarrow \varepsilon \triangleright t' \,\mathsf{sat}\, F[e/x] \\
\varepsilon \triangleright t \,\mathsf{sat}\, \langle c? \rangle G & = & \bigvee_{t \xrightarrow{b',c?} (x)t'} b' \wedge \varepsilon \triangleright (x)t' \,\mathsf{sat}\, G \\
\varepsilon \triangleright t \,\mathsf{sat}\, [c?] G & = & \bigwedge_{t \xrightarrow{b',c?} (x)t'} b' \rightarrow \varepsilon \triangleright (x)t' \,\mathsf{sat}\, G \\
\varepsilon \triangleright (y)t \,\mathsf{sat}\, \forall x.F & = & \forall w.(\varepsilon \triangleright t[w/y] \,\mathsf{sat}\, F[w/x]) \\
\varepsilon \triangleright (y)t \,\mathsf{sat}\, \exists x.F & = & \exists w.(\varepsilon \triangleright t[w/y] \,\mathsf{sat}\, F[w/x]) \\
\varepsilon \triangleright t \,\mathsf{sat}\, A.(\tilde{e}/\tilde{z}) & = & [\varepsilon(\tilde{e})/\tilde{z}] \triangleright t \,\mathsf{sat}\, A \\
\varepsilon \triangleright t \,\mathsf{sat}\, \nu X[\mathcal{A}]F & = & \begin{cases} \lfloor B \rfloor & \text{if } (B\widehat{\varepsilon}, t) \in \mathcal{A} \\ t \,\mathsf{sat}\, F[\nu X[\mathcal{A}']F/X] & \text{otherwise} \end{cases} \\
\varepsilon \triangleright t \,\mathsf{sat}\, \mu X[\mathcal{A}]F & = & \begin{cases} \mathbf{ff} & \text{if } (B\widehat{\varepsilon}, t) \in \mathcal{A} \\ (t \,\mathsf{sat}\, F[\mu X[\mathcal{A}'']F/X]) & \text{otherwise} \end{cases}
\end{array}$$

where $w = new(t, \varepsilon, \forall x.F)$, and tag sets $\mathcal{A}' = \mathcal{A} \cup ((\varepsilon \triangleright t \,\mathsf{sat}\, \nu X.[\mathcal{A}]F)\widehat{\varepsilon}, t)$ and $\mathcal{A}'' = \mathcal{A} \cup ((\varepsilon \triangleright t \,\mathsf{sat}\, \mu X.[\mathcal{A}]F)\widehat{\varepsilon}, t)$.

Figure 10: Sat construction for symbolic semantics

is a limit ordinal. Let α be the least such ordinal. M is a lattice of monotone functions so for all $\beta < \alpha$ and $b' \leq b$ we have that $t \notin \varphi^\beta(b')$ and so $\varphi^\beta = \varphi^\beta \setminus \lambda(b, t)$. The result now follows from the monotonicity of φ. □

Lemma 12 $t \models^s_{B\widehat{\varepsilon}} \mu X[\mathcal{A}]F$ *implie* $t \models^s_{B\widehat{\varepsilon}} F[\mu X[\mathcal{A} \cup (B\widehat{\varepsilon}, t)]F/X]$.

Proof. Follows from previous lemma taking T to be $\mathcal{T}(\mathcal{G})$ and $\mathcal{B}$ to be the boolean expressions (up to equivalence) ordered by $\models^{-1}$. □

The approach to proving completeness is the same as the proof of the previous section. That is we define a characteristic formula $t \,\mathsf{sat}\, F$ which is the solution a fixpoint formula over a first-order language of boolean expressions. We no longer require parameterised fixpoint formulae as we deal with the recursion parameters using the $B\widehat{\ }$

therefore any boolean information required to do this proof can be expressed without least fixpoints also. Again we note that the tag sets will contain more information than is strictly necessary to define sat; for fixpoints we only need to record the term t and the environment ε in the tag sets but for the sake of a cleaner presentation later on we include the extra syntax.

We define what it means for a formula F to be tag restricted in a similar manner to before; $\nu X.[\mathcal{A}]F$ (or $\mu X.[\mathcal{A}]F$) is tag restricted if either $\mathcal{A}$ is empty or if $\mathcal{A} = \mathcal{A}' \cup (B\widehat{\varepsilon}, t)$ with $t \notin \mathcal{A}'$then B is $\varepsilon \triangleright t \,\mathrm{sat}\, \nu X.[\mathcal{A}']F$ (or $\varepsilon \triangleright t \,\mathrm{sat}\, \mu X.[\mathcal{A}']F$) and $\nu X[\mathcal{A}']F$ is also tag restricted. A formula F is tag restricted if all of its abstraction subformulae are tag restricted. A term t can now appear more than once in the tag set of a tag restricted formula. However, any given term t along with a substitution ε may appear at most once.

This change will of course affect our ordering $\ll$. The relation $\ll$ given in the previous section

$$
\begin{array}{rclcl}
DApps(B) & = & DApps(X) & = & \emptyset \\
DApps(F_1 \wedge F_2) & = & DApps(F_1 \vee F_2) & = & DApps(F_1) \cup DApps(F_2) \\
DApps(\langle \alpha \rangle F) & = & DApps([\alpha] F) & = & DApps(F) \\
DApps(\forall x.F) & = & DApps(\exists x.F) & = & DApps(F) \\
DApps(\mu X[\mathcal{A}]F) & = & DApps(\nu X[\mathcal{A}]F) & = & DApps(F) \\
DApps(A.(e/z)) & = & \multicolumn{3}{l}{\left\{ \begin{array}{ll} DApps(A) \cup \{e\} & \text{If } e \cap RPar = \emptyset \\ DApps(A) & \text{If } e \in RPar \end{array} \right.}
\end{array}
$$

Figure 11: Definition of function *DApps* over formulae.

All environments ε' used when calculating us that $\delta[\tilde{e}/\tilde{z}] \models [\![\varepsilon' \triangleright t \,\mathrm{sat}\, A]\!]\eta$ and because $\tilde{z}$ does not appear free

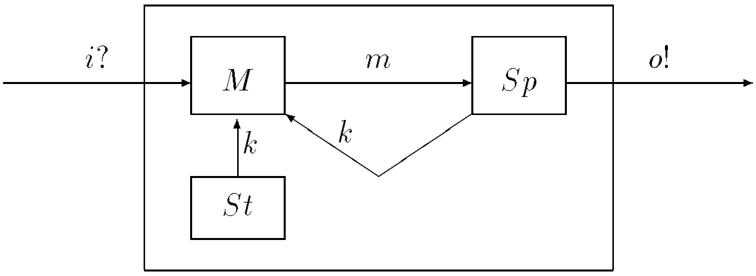

Figure 12: Flow diagram for process Max.

Given a syntactic description of a process it is a simple matter to compile it down to a symbolic graph. This treatment can be given to Max and we see in Figure 13 that the resulting graph is in fact finite. We should point out that at node t_1 in this graph the τ transition leaving this node is guarded by the boolean $y \geq 0$. In light of the fact that y is an non-negative integer we elide this guard. Also, as a companion to this τ transition is another τ move with false guard $y < 0$. We prune this branch of the graph for the sake of clarity.

The property that we wish to prove Max to satisfy is that for every input on channel i there is an output on channel o of the maximum value received so far. Naturally there are internal actions to be accounted for so we will consider weak modalities, $\langle\langle\alpha\rangle\rangle$ and $[[\alpha]]$. A term will satisy $\langle\langle\alpha\rangle\rangle F$ if it can do *finitely* many τ transitions followed by an α transition to a term which satisfies F. Because we demand only finitely many τ transitions we use least fixpoints to define these modalities:

$$\langle\langle\alpha\rangle\rangle F \equiv (\mu X.\langle\tau\rangle X.(\tilde{z}/\tilde{z}) \vee \langle\alpha\rangle F).(\tilde{x}/\tilde{z})$$

where $\tilde{x} = fv(F)$. Similarly for box modalities

$$[[\alpha]]F \equiv (\mu X.[\tau]X.(\tilde{z}/\tilde{z}) \wedge [\alpha]F).(\tilde{x}/\tilde{z}).$$

We write the specification as a greatest fixpoint formula,

$$F_{Max} \equiv [[i?]]\forall y.A.(y,0/z,z')$$

where z is a parameter which represents the last value input, z' is a parameter which represents the maximum value received so far and A is defined to be $\nu X.(F_1 \wedge F_2)$. We use two formulae F_1 and F_2 to reflect the fact that, in addition to immediately outputting after an input, the process is able to receive (at most) the next input to be compared before any output transition occurs. These formulae can be written.

$$F_1 \equiv \langle\langle o!x\rangle\rangle[[i?]]\forall y'.F_3 \quad \text{and} \quad F_2 \equiv [[i?]]\forall y'.\langle\langle o!x\rangle\rangle F_3$$

where

$$F_3 \equiv ([x = z' \wedge z' > z] \vee [x = z \wedge z \geq z']) \wedge X.(y',x/z,z').$$

It is possible to give a proof that $\mathbf{tt} \vdash t_0 : F_{Max}$, however, purely in order to make the proof concise, we use more specific formulae to replace F_1 and F_2. We actually will use

$$F_1 \equiv \langle\langle o!x\rangle\rangle[[i?]]\forall y'.(\langle\tau\rangle)F_3 \vee F_3)$$

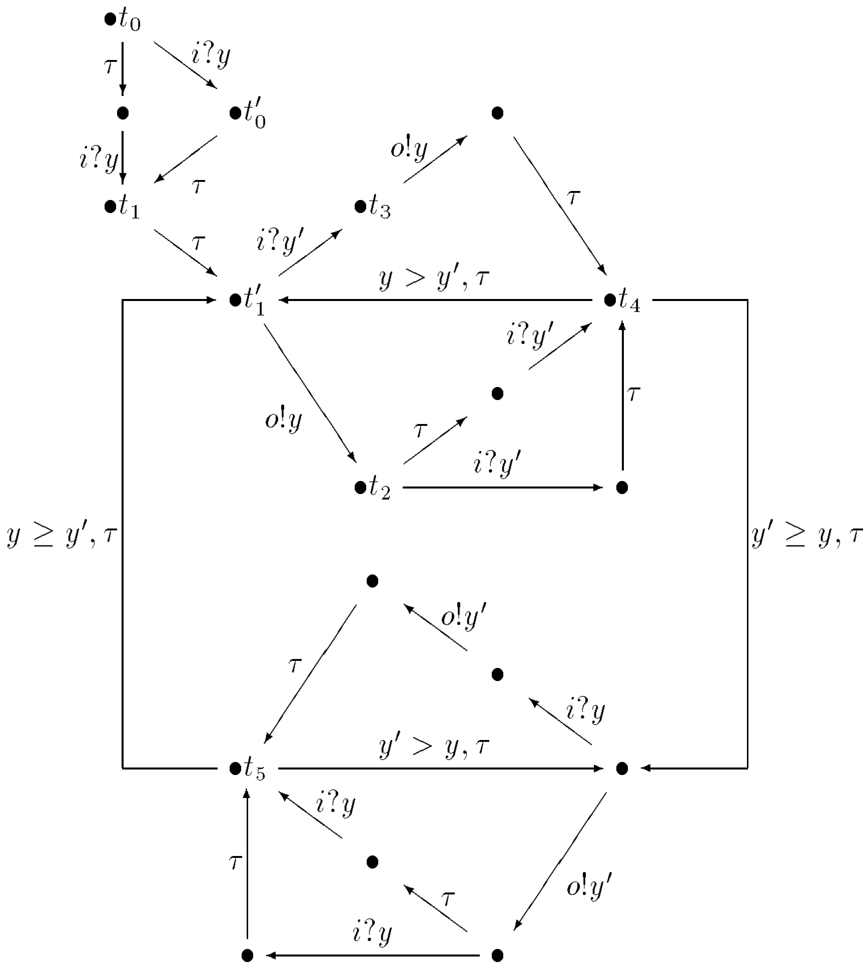

Figure 13: Symbolic graph for Max.

and

$$F_2 \equiv [[i?]]\forall y'.\langle o!x\rangle\langle\tau\rangle F_3.$$

These formulae differ from the former two only in their τ modalities.

In the following proof $B \vdash t, t' : F$ will be an abbreviation for the two sequents $B \vdash t : F$ and $B \vdash t' : F$ and B_{Max} will denote the boolean expression $[x = z' \wedge z' > z] \vee [x = z \wedge z \geq z']$ so that $F_3 \equiv B_{Max} \wedge X.(y', x/z, z')$.

The goal $\mathbf{tt} \vdash t_0 : F_{Max}$ follows from rules $\forall$, $[\tau]$, $[i?]$, $\wedge$, μ and *App* by using the rule $\wedge$. Taking each of these in turn we see that the former pair can be reduced, by using μ unfolding and $\langle\tau\rangle$ rules (twice for the t_0' case) and then a $\langle o!\rangle$ rule to get

$$\widehat{\varepsilon_0} \vdash t_2 : [[i?]]\forall y'(\langle\tau\rangle F_3 \vee F_3)[A_1/X][y/x]. \tag{5}$$

Secondly we reduce the latter pair of sequents to

$$\widehat{\varepsilon_0} \vdash t_3 : \langle o!x\rangle\langle\tau\rangle F_3[A_1/X] \tag{6}$$

again by using μ unfoldings and $[\tau]$ rules and a $[i?]$ rule. Now both (5) and (6) can be reduced to the single judgement $\widehat{\varepsilon_0} \vdash t_4 : F_3[A_1/X][y/x]$ by using the appropriate modality rules. We notice that $\widehat{\varepsilon_0} \models B_{Max}[y/x]$ so, using rules $\wedge$, Cons and Id, our proof obligation becomes $\widehat{\varepsilon_0} \vdash t_4 : A_1.(y', y/z, z')$. We strip the outer application with rule App to get

$$\widehat{\varepsilon_1} \vdash t_4 : A_1$$

where $\widehat{\varepsilon_1} \equiv z = y' \wedge z' = y$. This judgement is ready to be ν unfolded to become

$$\widehat{\varepsilon_1} \vdash t_4 : (F_1 \wedge F_2)[A_2/X]$$

where $A_2 = \nu X.[\mathcal{A}, (\widehat{\varepsilon_1}, t_4)]F_1 \wedge F_2$. At this point we do a case analysis on y and y'. This is done by using the Cut rule to the two sequents

$$y > y' \wedge \widehat{\varepsilon_1} \vdash t_4 : (F_1 \wedge F_2)[A_2/X] \tag{7}$$

and

$$y' \geq y \wedge \widehat{\varepsilon_1} \vdash t_4 : (F_1 \wedge F_2)[A_2/X]. \tag{8}$$

We deal with (7])

c

where $A_3 = \nu X[\mathcal{A}, (\widehat{\varepsilon_1}, t_4$

uses a very simple language of boolean expressions containing only the atoms **tt** and **ff**. Of greater interest though is the way in which their treatment of fixpoint abstractions differs from the present work. In [1] fixpoints are interpreted as functions from vectors of names (analagous to our evaluations) into sets of terms or agents but, at the level of the proof system, abstractions are dealt with *pointwi e*. On the other hand we, by means of the App rule, deal with fixpoint expressions as abstractions proper. The ν-unfolding rule of [1] is given by

$$\frac{p : \phi[\vec{b}/\vec{a}][\nu X(\vec{a})\phi/X]}{p : (\nu X(\vec{a})\phi)(\vec{b})}$$

where $\vec{a}$ and $\vec{b}$ are vectors of names and all tag set information has been elided. The fixpoint formula is unfolded at the point $\vec{b}$ and this point is substituted into the unfolding. We would do no such substitution as we have already abstracted away from the particular point $\vec{b}$ by using App. These two approaches are more or less equivalent for the π-calculus and indeed for our restricted parameter sublogic of Section 5, because the nature of the data domain is such that only finitely many points will be encountered in a proof tableaux for a fixpoint formula. However the limitations of the pointwise approach become apparent when we consider more general languages of data expressions. The example proof that a process P satisfies

$$(\nu X.\langle a!x\rangle(x = z\,mod 2) \wedge X.(z+1/z)).(0/z)$$

which we presented in the introduction would be infeasible using the pointwise approach; we would necessarily start at the point 0, then progress by unfolding and modality rules to checking at the point 1, similarly on to point 2 and so on. Therefore our proof system generalises the approach in [1] and Section 5 shows that we incorporate (modulo π-calculus technicalities) the proof system of [1] by emulating the pointwise approach using the $\widehat{\varepsilon}$ boolean forms.

References

[1] R. Amadio and M. Dam. Toward a modal theory of types for the π-calculus. 1996. To appear.

[2] A. Arnold and P. ab990T6T.24Tf4.:the

[6] M. Hennessy and X. Liu. A modal logic for message passing processes. Technical