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) Recursive 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 generalised to the operators $\langle a!x\rangle F$, $\langle a?\rangle \exists x.F$, $\langle a?\rangle \forall x.F$ and $[a!x]F$, $[a?]\exists x.F$, $[a?]\forall x.F$ respectively, all of which bind occurrences of x in F. The use of this property language leads naturally to judgements of the form

$$T : F$$

where T is an *open* process term, i.e. does not in itself represent a process; it may contain free occurrences of *data variables* which need to be instantiated in order to represent a process. For example consider the (value-passing) process

$$P \Leftarrow a?x.b!(x+1).P$$

which can input any value x on the channel a and output its successor on channel b. The process P obviously satisfies the property

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 procedure will be expressed in terms of a proof system whose judgements are of the form

$$B \vdash T : F$$

where B is a boolean expression and T is an open process term. Such a proof system has been given in [6] for a first-order modal property language and in this paper we extend it by allowing the property formulae F to be defined using parameterised maximal and minimal fixpoints and by using the tag set method of [13], outlined above, to design associated proof rules.

A typical example of a parameterised maximal fixpoint is given by

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

Here z is used as a *recursion parameter* and in general these formulae are interpreted as abstractions over values; formulae are obtained by applying them to expressions denoting values. So if A refers to this abstraction then $A.(0/z)$ is a formula representing the

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 *generalisation* 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 *relativised* 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 expect any general completeness theorems. For example, checking if $a!e_1.P$ is equivalent to $a!e_2.P$ involves checking whether the data-expressions e_1, e_2, which may have free occurrences of data-variables, are semantically equivalent and for sufficiently rich data languages this would be undecidable. However our proof rules already work modulo an arbitrary proof system, or oracle, for data-expressions, with deductions such as

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

where $B_2 \models B_1$ indicates that B_2 semantically implies B_1. So a completeness result for our proof method would in effect be a relative completeness result, relative to an oracle for data-expressions. Also the completeness results in [13], for *pure processes*, is with respect to *finite* labelled transition systems rather than process terms from some process language; the model checking procedure in [13], is complete for finite state transition systems. Here we investigate the extent to which (relative) completeness can be obtained for finite symbolic transition systems.

Our first result is a negative one. By using a simple example we can show that our system is not even relative complete for minimal fixpoint formulae. Consider the process

$$P \Leftarrow c?x.c!x.P$$

and the formula F, which intuitively says that P can output some integer on the channel c, having first input an integer:

$$\langle c?\rangle \forall x((\mu X(\langle c!y\rangle y = z) \vee X.(z + 1/z)).(0/z))$$

Semantically P satisfies the property F but we will show that

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

is not derivable in our system. The argument will be quite simple: for any Q, G, if $\mathbf{true} \vdash Q : G$ is derivable then so is $\mathbf{true} \vdash Q : G^n$, for some n, where G^n is obtained by *unwinding* the recursive definitions in G, n times. However (assuming the soundness of our rules) $\mathbf{true} \vdash P : F^n$ can not be derivable for any n since it is not semantically true. The details may be seen at the end of Section 3.

For the sub-language containing only maximal fixpoints, the language of *liveness* properties, we do have a completeness result. In Section 4 we show that if T is a term over a finite symbolic transition system which satisfies a liveness property F in every interpretation which satisfies the boolean B then we can derive the judgement

$$B \vdash T : F.$$

Moreover the proof is constructive; we give an effective method for deriving the judgement using our proof rules. Recall from the above exposition that the difficulties in elaborating proofs with our system of rules arise because

- tags contain pairs (B, T) and so even if the symbolic transition system is finite there is an infinite number of potential tags,
- suitable data expressions must be conjured up in order to use the *generalisation* rule App.

Our completeness theorem

- constructs proofs in which every term from the finite symbolic system occurs at most once in a tag set,
- uses a general method for generating the required boolean expressions for use with the *generalisation* rule App.

The latter point is tackled indirectly as for each term in a finite symbolic transition system T and formula F we define a boolean expression $(T \,\mathrm{sat}\, F)$ such that

- in every interpretation which satisfies the boolean expression $(T \,\mathrm{sat}\, F)$ the process term T satisfies the modal formula F
- there is a systematic derivation of $(T \,\mathrm{sat}\, F) \vdash T : F$ using our proof rules.

In order to describe these characteristic boolean expressions $(T \,\mathrm{sat}\, F)$ we need a very expressive language of boolean expressions. Essentially we introduce a *property variable* for each term in the symbolic transition system and the characteristic expression uses maximal fixpoints over these variables. Thus these may be very complicated but in many cases these fixpoints can be eliminated, or solved, to obtain propositional or first order boolean expressions. This completeness result is described in detail in Section 4.

The following Section 5 contains another completeness result for a different sub-language of the property language. Here both maximal and minimal fixpoints are allowed

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

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

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

Figure 1: Grammar for the logic

but abstractions which use them may only be applied to *restricted parameters*; 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 a short conclusion, outlining what we have achieved, future directions for research and related work.

2 The Logic and its Models

The present work is parameterised with respect to data domains; we do not give a precise description of any such domains but we will demand that certain properties hold. We assume that we have a non-empty set *Val* of values, a set *DVar* of data variables and a language *ValExp* of expressions built from these. Similarly we assume a language of boolean expressions called *BoolExp* which contains at least the two truth values {**tt**,**ff**} and the expression $e = e'$ for each $e, e' \in ValExp$. A *substitution* $\sigma : DVar \to ValExp$ is a total function. We write $[e/x]$ for the substitution

$$[e/x](y) = \begin{cases} e & \text{if } y = x \\ y & \text{otherwise} \end{cases}$$

We assume that substitutions behave in a reasonable manner when extended to *ValExp* and *BoolExp*. There are two particular types of substitution that we will make use of. Firstly, *evaluations* $\delta : DVar \to Val$ where the range of the substitution is contained entirely in the value set. Secondly we will refer to a substitution, $\sigma : DVar \to DVar$ whose range consists of variables alone, as a *simple* substitution. When the expression $e \in ValExp$ is *closed*, that is, contains no occurrences of variables, then the value of the expression is independent of evaluations and we denote it by $[\![e]\!]$. Similarly we denote the value of a closed boolean expression, b by $[\![b]\!]$. Given an evaluation δ and a boolean expression b, we write $\delta \models b$ if $[\![b\delta]\!] = \mathbf{tt}$; similarly $b \models b'$ will mean $\delta \models b$ implies $\delta \models b'$ and we will use the slightly non-standard notation $\delta[e/x]$ to denote the evaluation δ' which coincides with δ except at x where we have $\delta'(x) = [\![e\delta]\!]$.

We give a three-level grammar to describe the logic in Figure 1, where B ranges over $BoolExp$, c ranges over a set of channel names $Chan$ and X ranges over a set of *recursion*

variables 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, which are used in conjunction with the modal operators $\langle c!x \rangle$ and $[c!x]$. Note that these act as binders for the variable x as do the quantifiers $\exists x$ and $\forall x$. Finally A ranges over *recursion abstractions* but these may only be used to define modal formulae by the construction $A.(\tilde{e}/\tilde{z})$. This denotes the application of the abstraction A to the evaluation $[\tilde{e}/\tilde{z}]$. This binds the vector of variables $\tilde{z}$ in A and in this situation whenever A is of the form $\nu X[\mathcal{A}]F$ or $\mu X[\mathcal{A}]F$ we assume that the names $\tilde{z}$ comprise all of the free data variables present in F. These variables $\tilde{z}$ are called the *recursion parameters* of A and we use $RPar$ to refer to the subset of $DVar$ of all such variables. In these formulae we say that the set $\mathcal{A}$ is the tag set for the variable X.

The choice to bind the recursion parameters of the abstraction $A \equiv \nu X[\mathcal{A}]F$, say, at the application stage may appear rather unorthodox. For example one might expect that A is a closed formula of functional type. In our setting the free variables of A are the free variables of F; that is to say that A is in fact just an open formula. We exploit the fact that open formulae may be construed as functions from evaluations to closed terms and thus, beyond the syntactic domain, there is no real distinction between abstractions and open terms. This blurring of concepts enables us to give a cleaner presentation of both the semantics of the logic and the proof system of Section 3.

In the sequel, for convenience we will restrict the use of recursion variables: we assume that recursion variables are bound at most once in any given formula and that the parameter variables in each recursion abstraction are distinct from any variables used in the symbolic graphs.

2.1 Symbolic Graphs and models

The models that we employ are based on the symbolic graphs of [5]; a symbolic graph is a directed graph in which every node n has an associated set of variables called $fv(n)$, intended to capture the idea of free variables of a term. We have a set of action labels,

$$Act = \{c?x, c!e \mid c \in Chan, x \in Var, e \in ValExp\} \cup \{\tau\},$$

and we lift the definition of free and bound variables to actions by letting $fv(c!e) = fv(e), bv(c?x) = \{x\}$, otherwise $fv(\alpha)$ and $bv(\alpha)$ are both empty. Every branch is labelled by a pair b, α where $b \in BoolExp$ and $\alpha \in Act$ such that if the branch so labelled connects node n to n', written $n \xrightarrow{b,\alpha} n'$, then we require that $fv(b) \cup fv(\alpha) \subseteq fv(n)$ and $fv(n') \subseteq fv(n) \cup bv(\alpha)$. We refer the reader to [5] for examples of symbolic graphs and a description of how one obtains a symbolic graph from a value-passing process calculus.

Intuitively nodes in a symbolic graph correspond to open terms from some value-passing process calculus and we wish to give an operational semantics to objects correspoding to the closed terms of that hypothetical language. However it will be convenient later if we have a more general representation of open terms, one which supports the operation of substitution, or renaming of variables.

We call a pair (n, σ), where n is a node of a symbolic graph and σ a simple substitution, a *term*. Because we are considering a late semantics we will also have terms of the form $\lambda y.(n, \sigma[y/x])$, a function from $DVar$ to terms. These will represent data abstractions. We simply write n_σ for the term (n, σ), write $(x)n_\sigma$ for the term $\lambda y.(n, \sigma[y/x])$

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

Figure 2: Symbolic operational semantics for open terms.

$$t \xrightarrow{b,\tau} t' \quad \text{implies } t\delta \xrightarrow{\tau} t'\delta \quad \text{if } [\![b\delta]\!] = \mathbf{tt}$$
$$t \xrightarrow{b,c!e} t' \quad \text{implies } t\delta \xrightarrow{c!v} t'\delta \quad \text{if } [\![b\delta]\!] = \mathbf{tt} \text{ and } [\![e\delta]\!] = v$$
$$t \xrightarrow{b,c?} (x)t' \text{ implies } t\delta \xrightarrow{c?} (x)t'\delta \quad \text{if } [\![b\delta]\!] = \mathbf{tt}$$

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 *closed terms* whenever δ is an evaluation and for convenience it is written as $t\delta$. We let $p, q, \ldots$ range over closed terms. We can give an operational semantics to closed terms based on the edges of the symbolic graph. This is defined in in Figures 3 and uses a symbolic semantics for open terms, given in Figure 2.

Given a symbolic graph $\mathcal{G}$ we write

$$\mathcal{T}(\mathcal{G}) = \{n_\sigma, (x)n_\sigma \mid n \in \mathcal{G}, \sigma : DVar \to DVar\}$$

for the set of (open) terms of $\mathcal{G}$ and write

$$\mathcal{CT}(\mathcal{G}) = \{t\delta, (x)t\delta \mid t \in \mathcal{T}(\mathcal{G}), \delta : DVar \to Val\}$$

for the set of closed terms of $\mathcal{G}$.

These sets of closed terms will form the labelled transition systems which are the models of our logic subject to the following interpretations. We use *recursion environments* $\rho : RVar \to ((DVar \to Val) \to \mathcal{P}(\mathcal{CT}(\mathcal{G}))$. Interpretations are mappings from formulae in a recursion environment under an evaluation δ into sets of closed terms. Details are listed in Figure 4. We define a satisfaction relation using these semantics:

$$t \models_{\rho,B} F \text{ iff } t\delta \in [\![F]\!]\rho\delta \text{ for every } \delta \text{ such that } \delta \models B.$$

When we are using recursion closed formulae this relation is independent of ρ and we simply write $t \models_B F$. For the sub-logic without fixpoint expressions this satisfaction relation coincides with the one proposed in [6] where it is proved that this finite sub-logic characterises late bisimulation equivalence. There now follows a technical Lemma which is a generalisation of the so called Reduction Lemma of [13], the essence of the tag set method.

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

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 $L = V \to \mathcal{P}T$ be a complete lattice and let $\varphi : L \to L$ 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 is empty otherwise.

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 transitions in a graph. The judgements of the proof system are sequents of the form $B \vdash t : F$ where B is a boolean expression, t is a term and F is a recursion closed formula of the logic. The proof rules for the system are listed in Figures 5 and 6.

The former imports all of the rules from [6], the latter introduces the rules necessary for the treatment of the fixpoint operators. These new rules are the obvious adaptions of the unfolding rules of [13] to the current setting with the possible exception of rule *App*. This rule can be understood as a combination of two simpler rules: Firstly,

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

which says that the satisfaction relation is preserved by substitution. Note that the assumption that t does not contain any of the recursion parameters $\tilde{z}$ is vital for soundness here. Secondly,

$$\frac{B \vdash t : A}{B \vdash t : A.(\tilde{z}/\tilde{z})}$$

which is a form of λ-elimination. However we do not require that $\tilde{z}$ do not occur freely in B as one would expect from a λ-elimination rule proper. The formula A is a function from evaluations to closed terms and the formula B specifies the domain of evaluations which we are interested in.

The side condition on the μ rule ensures that a term t, say, appearing in a tag set more than once, does so in disjoint boolean worlds. This may seem overly restrictive however it is necessary for soundness. For example we could use the μ-rule to deduce

$$\mathbf{tt} \vdash t : \mu X[(B', t)]\mathbf{tt}$$

since $\vdash t : \mathbf{tt}$ is trivially derivable. However

$$[\![\mu X[(B', t)]\mathbf{tt}]\!]\rho\delta = \mathcal{CT}(\mathcal{G}) \setminus \{t\delta' \mid \delta' \models B'\}$$

which is different from $\mathcal{CT}(\mathcal{G})$ provided there is some δ such that $\delta \models B'$.

Theorem 3 *(Soundness) If $B \vdash t : F$ then $t \models_B F$.*

Proof. By induction on the length of the derivation $B \vdash t : F$. The soundness of the crucial rule ν_1 is guaranteed by Lemma 2 while that of ν_0 follows in a straightforward manner from the interpretation of tagged fixpoints. We leave the reader to verify the soundness of the μ rule and we show here the case for the application rule. Suppose $\delta \models B[\tilde{e}/\tilde{z}]$. Then we know that $\delta[\tilde{e}/\tilde{z}] \models B$ and thus by induction we have $t\delta[\tilde{e}/\tilde{z}] \in [\![A]\!]\rho\delta[\tilde{e}/\tilde{z}]$. Now $\tilde{z} \notin fv(t)$ so we have $t\delta \in [\![A]\!]\rho\delta[\tilde{e}/\tilde{z}]$ which is, by definition, $t\delta \in [\![A.(\tilde{e}/\tilde{z})]\!]\rho\delta$. Therefore $t \models_{B[\tilde{e}/\tilde{z}]} A.(\tilde{e}/\tilde{z})$. □

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

$$Cons \quad \frac{B_1 \vdash t : F}{B_2 \vdash t : F} \quad (B_2 \models B_1) \qquad Ex \quad \frac{B \vdash t : F}{\exists x.B \vdash t : F} \quad (x \notin fv(t, F))$$

$$\alpha \quad \frac{B \vdash t' : F'}{B \vdash t : F} \quad (t' \equiv_\alpha t,\ F' \equiv_\alpha F) \qquad \wedge \quad \frac{B \vdash t : F_1 \quad B \vdash t : F_2}{B \vdash t : F_1 \wedge F_2}$$

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

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

$$[\tau] \quad \frac{B \wedge b_1 \vdash t_1 : F, \ldots, B \wedge b_n \vdash t_n : F}{B \vdash t : [\tau]F}$$
$$\text{where } \{(b_1, t_1), \ldots, (b_n, t_n)\} = \{(b, t') \mid t \xrightarrow{b,\tau} t'\}$$

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

$$[c!] \quad \frac{B \wedge b_1 \vdash t_1 : F[e_1/x], \ldots, B \wedge b_n \vdash t_n : F[e_n/x]}{B \vdash t : [c!x]F}$$
$$\text{where } \{(b_1, t_1, e_1), \ldots, (b_n, t_n, e_n)\} = \{(b, t', e) \mid t \xrightarrow{b,c!e} t'\}$$

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

$$[c?] \quad \frac{B \wedge b_1 \vdash (y_1)t_1 : F, \ldots, B \wedge b_n \vdash (y_n)t_n : G}{B \vdash t : [c?]G}$$
$$\text{where } \{(b_1, (y_n)t_1), \ldots, (b_n, (y_n)t_n)\} = \{(b, (y)t') \mid t \xrightarrow{b,c?} (y)t'\}$$

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

Figure 5: Local model checking rules.

Having proved the soundness of our proof system we turn to the question of completeness. Unfortunately we have to report that the system is incomplete for the full logic. The problem lies with the least fixpoint formulae. Consider the following example:

Let Val be the set of natural numbers and let the graph $\mathcal{G}$ have three terms t_1, t_2, t_3 with branches $t_1 \xrightarrow{c?x} t_2$ and $t_2 \xrightarrow{a!x} t_3$. The abstraction $\mu X[\emptyset]F$, where F is $(\langle a!y \rangle y =$

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

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

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

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, of some value." Therefore it should be clear that

$$t_1 \models_{\mathbf{tt}} \langle c? \rangle \forall x((\mu X[\emptyset]F).(0/z))$$

We now argue that $\mathbf{tt} \vdash t_1 : \langle c? \rangle \forall x(\mu X[\emptyset]F.(0/z))$ is not derivable. First consider the sequence of formulae $F^0 = \mathbf{ff}$ and $F^{n+1} = F[F^n[e/z]/X.(e/z)]$. To see that $t_2 \not\models F^k[0/z]$ for all k we suppose the contrary, that is $t_2 \models F^k[0/z]$ for some k. Hence for every δ we get that $t_2\delta \in [\![F^k[0/z]]\!]\rho\delta$. Now $[\![F^k[0/z]]\!]\rho\delta = \bigcup_{n<k} \{t_2\delta' \mid \delta'(x) = n\}$. So choose δ_0 such that $\delta_0(x) = k+1$ then by assumption $t_2\delta_0 \in \bigcup_{n<k} \{t\delta' \mid \delta'(x) = n\}$. This is a contradiction. From this we can conclude that $t_1 \models \langle c? \rangle \forall x F^k[0/z]$ for no k.

We now prove that if a term provably satisfies a least fixpoint then it provably satisfies a finite unwinding of this fixpoint. Having done this we easily see, by soundness, that $\mathbf{tt} \vdash t_1 : \langle c? \rangle \forall x(\mu X[\emptyset]F.(0/z))$ is not derivable.

Proposition 4 *If $B \vdash t : \mu X[\mathcal{A}]F$, then there exists $k \geq 0$ such that $B \vdash t : F^k$.*

Proof. We prove, by induction over depth of proofs, a more general statement: Let F' be an open formula such that $RVar(F') \subseteq \{X\}$. Suppose $B \vdash t : F'[\mu X[\mathcal{A}]F/X]$ and let k be the maximum number of occurrences of the μ-rule, using t and X, on any branch of this derivation. Then we have that $B \vdash t : F'[F^{k'}/X]$ for all $k' \geq k$.

This is reasonably straightforward except for the following illustrative cases:

Case: Conjunction.

$$\wedge\quad \frac{B \vdash t : F_1[\mu X[\mathcal{A}]F/X] \qquad B \vdash t : F_2[\mu[\mathcal{A}]F/X]}{B \vdash t : (F_1 \wedge F_2)[\mu X[\mathcal{A}]F/X]}.$$

Suppose without loss of generality that the maximum number of occurrences of the μ-rule appear on the left branch. Then by induction we get $B \vdash t : F_1^{k'}$ for $k' \geq k$. Because we chose k to be maximum across all branches then we also know by induction that $B \vdash t : F_2^{k'}$ for all $K' \geq K_2$ where $K \geq K_2$; in particular, we have this derivation for all $k' \geq k$. Therefore, using rule $\wedge$ we get the result.

Case: Fixpoints. Suppose that the last rule used in the derivation of $B \vdash t : F'[\mu X[\mathcal{A}]F/X]$ was the μ-rule. There are two cases to consider. If F' is simply the formula X then we have

$$\mu\quad \frac{B \vdash t : F[\mu X[\mathcal{A}, (B,t)]F/X]}{B \vdash t : \mu X[\mathcal{A}]F}.$$

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

Case: Application.

If F' is a fixpoint formula then we simply apply induction and use the App rule. In the case where F' is $X.(\tilde{e}/\tilde{z})$ we need a further (easy) induction to show that if $B \vdash t : F^{k'}$ then $B[\tilde{e}/\tilde{z}] \vdash t : F^{k'}[\tilde{e}/\tilde{z}]$. □

It is clear from this that the proof system is incomplete for the full logic. Therefore in order to obtain any kind of completeness results we must either augment the proof system or consider sub-logics. We opt for the latter, in fact we consider two restricted versions of the logic. In the next section we will prove completeness for the sub-logic obtained by disallowing all μ-formula — a sub-logic of safety properties. The proof involves developing a language for expressing the boolean expressions required to establish $B \vdash t : F$ and in this sense the proof is constructive. In the Section 5 we reintroduce μ-formulae but we restrict the type of expressions that may be passed to them as parameters. A formula F will now, until Section 5, be assumed to mean a formula without any least fixpoint subformulae.

4 Completeness and the Sat construction

In [6] it was shown how it is possible, for finite F, to reduce the statement $t \models F$ to a first-order formula $(t \,\mathrm{sat}\, F)$, thereby reducing the question of a point satisfying a modal formula down to validity of a boolean expression. We adopt the same approach here. However, to cope with the added complexity of parameterised fixpoints we reduce the statement $t \models F$ to a greatest fixpoint formula over a first-order language. This involves introducing fixpoints into our boolean language but what we achieve by doing so is an effective procedure for finding booleans sufficient to establish the judgements required. If we refer back to the example proof given in the introduction then we see that the boolean conditions equivalent to $even(z)$ and $odd(z)$ would be generated automatically as solutions of suitable formulae.

We now give a presentation of the first-order logic with parameterised fixpoints in which we construct our sat formulae. The logic is described by the following grammar:

$Q ::= B \mid Q \vee Q \mid Q \wedge Q \mid B \rightarrow Q \mid \exists x.Q \mid \forall x.Q \mid Q_A.(\tilde{e}/\tilde{z})$

$Q_A ::= X \mid \nu X.Q$

We present the semantics of this logic by first translating into (possibly open) terms of a first-order logic with infinite disjunction and then interpreting the translations in a

$$
\begin{array}{rcl}
[\![B]\!]\xi & = & B \\
[\![Q_1 \vee Q_2]\!]\xi & = & [\![Q_1]\!]\xi \vee [\![Q_2]\!]\xi \\
[\![Q_1 \wedge Q_2]\!]\xi & = & [\![Q_1]\!]\xi \wedge [\![Q_2]\!]\xi \\
[\![B \rightarrow Q]\!]\xi & = & B \rightarrow [\![Q]\!]\xi \\
[\![\exists x.Q]\!]\xi & = & \exists x.[\![Q]\!]\xi \\
[\![\forall x.Q]\!]\xi & = & \forall x.[\![Q]\!]\xi \\
[\![Q_A.(\tilde{e}/\tilde{z})]\!]\xi & = & [\![Q_A]\!]\xi[\tilde{e}/\tilde{z}] \\
[\![X]\!]\xi & = & \xi(X) \\
[\![\nu X.Q]\!]\xi & = & \bigvee \{b \mid b = [\![Q]\!]\xi[b/X]\}
\end{array}
$$

Figure 7: Interpretation of first-order fixpoint logic.

completely standard manner. Given a recursion environment $\xi : RVar \rightarrow BoolExp$, we translate formulae $[\![Q]\!]\xi$ as described in Figure 7 and define $\delta \models [\![Q]\!]\xi$ to be the obvious satisfaction relation. If Q has no free occurrences of recursion variables then $[\![Q]\!]\xi$ is independent of ξ and we simply write $[\![Q]\!]$. In such a case we will write $\delta \models Q$ to mean $\delta \models [\![Q]\!]$ and $Q \vdash t : F$ to mean $[\![Q]\!] \vdash t : F$.

The construction of the characteristic sat formulae is given in Figure 8. Although the definition of sat appears to be non-well-founded in the case for fixpoints we assume that no evaluation of sat formulae occurs inside tag sets. More precisely, consider the tag $(t \,\mathrm{sat}\, \nu X[\mathcal{A}]F, t)$ which is included in the tag set as we unwind a formula against t. While the expression $t \,\mathrm{sat}\, \nu X[\mathcal{A}]F$ is referred to in the context of being in a tag set then we treat it as a symbol only, it is the 'name' of a formula rather than the formula itself. Including the expression $t \,\mathrm{sat}\, \nu X[\mathcal{A}]F$ in the tag sets is rather gratuitous as we only really need to remember that t has been encountered before. However, this extra information eases presentation of the following Theorems significantly. Given a formula F, for each recursion variable X in F and each term t, we create a formula $\nu X_t \ldots$ All of the variables X_t have the same arity as the variable X. To prove that $(t \,\mathrm{sat}\, F)$ is always well-defined we develop a well-founded relation $\ll$ between formulae. This will only be defined between to so-called *tag restricted* formulae. $\nu X[\mathcal{A}]F$ *tag restricted* if $\mathcal{A}$ is empty or if $\mathcal{A} = \mathcal{A}' \cup (B, t)$ with $t \notin \mathcal{A}'$ then B is $t \,\mathrm{sat}\, \nu X[\mathcal{A}']F$ and $t \,\mathrm{sat}\, \nu X[\mathcal{A}']F$ is tag restricted. More generally, call a formula F tag restricted if each of its abstraction subformulae are tag restricted. An immediate consequence of a formula being tag restricted is that a term t may appear at most once in each tag set. A formula with empty tag sets is always tag restricted. Note that in the definition of $(t \,\mathrm{sat}\, G)$, when G has the form $\nu X[\mathcal{A}]F$, requires the calculation of $(t \,\mathrm{sat}\, F[\nu X[\mathcal{A}']F/X])$ and therefore we will require the latter to be dominated by the former, with respect to $\ll$. Note that here $\mathcal{A} \subseteq \mathcal{A}'$ and this is the basis of the definition of $\ll$. Let $Tags$ be the simple function which, given a formula F and a variable X, returns the set of terms present in the tag set of X in F. Using this then we write $F \ll F'$ iff F is a proper subformula of F' or $RVar(F) = RVar(F')$ and $Tags(X, F) \supset Tags(X, F')$ for each X.

Proposition 5 *If $\mathcal{T}(\mathcal{G})$ is finite then $\ll$ is a well-founded order on tag restricted formulae.*

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

where $w = new(t, \forall x.F)$ and $\lfloor B \rfloor = X_t$ if $B \equiv t \,\mathrm{sat}\, \nu X[\mathcal{A}]F$ and $\lfloor B \rfloor = B$ otherwise.

Figure 8: The sat construction.

Proof. Suppose otherwise, that is there is an infinite decreasing chain

$$\ldots F_n \ll \ldots \ll F_1 \ll F_0$$

for some tag restricted F_0. Let d be the depth of the formula F_0 so that after every d steps in the chain the tag set must of some recursion variable must increase. Let $N = |RVar(F_0)| \times 2^{|\mathcal{T}(\mathcal{G})|} \times d$ then we have $Tags(F_N, X) = \mathcal{T}(\mathcal{G})$ for each $X \in RVar(F_N)$. Furthermore $Tags(F_{N'}, X) = Tags(F_N, X)$ for all $N' > N$ and all X. Therefore there exists a $d' \leq d$ such that F_{N+d+1} is not a subformula of F_{N+d} and $Tags(F_{N+d'}, X) = Tags(F_{N+d'+1}, X)$ for each X which contradicts $F_{N+d'+1} \ll F_{N+d'}$. □

This means that, given a finite graph and only finitely many substitutions to create terms with, we have a well-founded ordering on a class of formulae. It was shown in [5] that careful choice of fresh variables ensured that the set of terms created by a finite graph could be kept finite. Therefore we may create $\mathcal{T}(\mathcal{G})$ so that it is finite whenever $\mathcal{G}$ is by only allowing substitutions using variables from the variables of $\mathcal{G}$ and the finite set of fresh variables used for renaming. It follows that $\ll$ is well-founded on tag restricted formulae and therefore that $(t \,\mathrm{sat}\, F)$ is well-defined.

As another application of the well-foundedness of $\ll$ we show that the characteristic formula construction corresponds to our semantics correctly.

Theorem 6 *For finte $\mathcal{G}$ and empty tag set F,*

$$\delta \models t \; sat F \;\; \textit{iff} \; t\delta \in [\![F]\!]\rho\delta.$$

Proof. We prove, by well-founded induction on tag restricted formulae using $\ll$, a stronger result that $\delta \models [\![t \,\mathrm{sat}\, F]\!]\eta$ iff $t\delta \in [\![F]\!]\rho\delta$ where η is some greatest fixpoint environment to be defined below.

Consider a recursion variable X in (tag restricted) F whose tag set contains the pair $(t \,\mathrm{sat}\, \nu X[\mathcal{A}]F', t)$. We wish to define $\eta(X_t)$ to be the greatest fixpoint of the formula $t \,\mathrm{sat}\, \nu X[\mathcal{A}]F'$, but this formula may contain free recursion variables so our solution would depend on an environment for these free variables. However, we can choose, because of tag restriction, an X and t such that $t \,\mathrm{sat}\, \nu X[\mathcal{A}].F'$ is recursion closed. We let $\eta(X_t) = [\![t \,\mathrm{sat}\, \nu X.F']\!]$. Given $\eta(X_t)$ we may now consider formula which have at most X_t free in them. So we choose Y and t' such that $t' \,\mathrm{sat}\, \nu Y[\mathcal{A}']F''$ contains at most X_t as a free variable and let $eta(Y_{t'}) = [\![t' \,\mathrm{sat}\, \nu Y[\mathcal{A}']F'']\!][\eta(X_t)/X_t]$. Continuing in this manner we obtain $\eta(X_t)$ for all pairs X and t which occur in the tag sets of F and observe that the collection of all such expressions forms an environment η. Moreover, $\eta(X_t) = [\![t \,\mathrm{sat}\, \nu X[\mathcal{A}]F]\!]\eta$ whenever $(t \,\mathrm{sat}\, \nu X[\mathcal{A}]F, t)$ is contained in the tag set of X.

We can now proceed with our well-founded induction. We show a couple of the interesting cases. Case: Application.

	$\delta \models t \,\mathrm{sat}\, A.(\tilde{e}/\tilde{z})$	
iff	$\delta \models (t \,\mathrm{sat}\, A).(\tilde{e}/\tilde{z})$	definition
iff	$\delta[\tilde{e}/\tilde{z}] \models t \,\mathrm{sat}\, A$	
iff	$t\delta[\tilde{e}/\tilde{z}] \in [\![A]\!]\rho\delta[\tilde{e}/\tilde{z}]$	by induction.
iff	$t\delta \in [\![A.(\tilde{e}/\tilde{z})]\!]\rho\delta.$	as $\tilde{z} \notin fv(t)$.

Case: ν-fixpoint with $t \in \mathcal{A}$. Let $\mathcal{A}'$ be such that $\mathcal{A} = \mathcal{A}' \cup (t \,\mathrm{sat}\, \nu X[\mathcal{A}']F, t)$.

	$\delta \models [\![t \,\mathrm{sat}\, \nu X[\mathcal{A}]F]\!]\eta$
iff	$\delta \models [\![X_t]\!]\eta$
iff	$\delta \models \eta(X_t)$
iff	$\delta \models [\![t \,\mathrm{sat}\, \nu X[\mathcal{A}']F]\!]\eta$
iff	$t\delta \in [\![\nu X[\mathcal{A}]F]\!]\rho\delta.$

Case: ν-fixpoint with $t \notin \mathcal{A}$. Let $\mathcal{A}' = \mathcal{A} \cup (t \,\mathrm{sat}\, \nu X.[\mathcal{A}]F, t)$.

	$\delta \models [\![t \,\mathrm{sat}\, \nu X[\mathcal{A}]F]\!]\eta$	
iff	$\delta \models [\![\nu X_t.t \,\mathrm{sat}\, F[\nu X[\mathcal{A}']F/X]]\!]\eta$	definition
iff	$\delta \models \eta(X_t)$	
iff	$\delta \models [\![t \,\mathrm{sat}\, F[\nu X[\mathcal{A}']F/X]]\!]\eta$	fixed point
iff	$t\delta \in [\![F[\nu X[\mathcal{A}']F/X]]\!]\rho\delta$	well-founded induction
iff	$t\delta \in [\![F]\!]\rho[[\![\nu X[\mathcal{A}']F]\!]\rho/X]\delta$	Lemma 2
iff	$t\delta \in [\![\nu X[\mathcal{A}]F]\!]\rho\delta$	

To establish that the well-founded induction is valid we must show that

$$F_1 \stackrel{def}{=} F[\nu X[\mathcal{A}']F/X] \ll \nu X[\mathcal{A}]F \stackrel{def}{=} F_2.$$

It is clear that F_1 is tag restricted so it suffices to show that $Tags(F_2, X) \subset Tags(F_1, X)$. Now $Tags(F_2) = T \cup Tags(F, X)$ where $T = \{t \mid (B, t) \in \mathcal{A}\}$ and $Tags(F_1, X) \supseteq T \cup \{t\} \cup Tags(F, X)$. Therefore $Tags(F_2, X) \subseteq Tags(F_1, X)$ easily. To show inequality just note that $t \in Tags(F_1, X)$ and $t \notin Tags(F_2, X)$ by assumption. □

Lemma 7 *For finite $\mathcal{G}$ and tag restricted F with η as above: $[\![t \,sat F]\!]\eta \vdash t : F$.*

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 we have our result. □

Theorem 8 *(Completeness) For all formula F with empty tag sets, finite $\mathcal{G}$: If $t \models_B F$ then $B \vdash t : F$.*

Proof. Suppose $t \models_B F$. Then Theorem 6 implies that for every $\delta \models B$ we have $\delta \models t \,\mathrm{sat}\, F$, which is to say $B \models t \,\mathrm{sat}\, F$. The previous Lemma tells us that $t \,\mathrm{sat}\, F \vdash t : F$ is derivable so an application of Cons gives $B \vdash t : F$. □

5 Restricted Parameters

We now consider an alternative way of restricting the logic in order to obtain a completeness result. Instead of removing least fixpoints completely we limit the usage of the parameters fed to them. We say that a formula F has *restricted parameters* if for each subformula in F of the form $A.(\tilde{e}/\tilde{z})$ we have that for each $e_i \in \tilde{e}$ either

- $e_i = z_i$ for some $z_i \in RPar(F)$ so that e_i is simply a recursion parameter, or
- $var(e_i) \cap RPar(F) = \emptyset$ so e_i contains no recursion parameters at all.

Expressions such as $(z + 1/z)$ are excluded. For the remainder of this section we assume that F has restricted parameters.

The effect of this restriction is to reduce the role of parameters to that of remembering values (or value expressions) which occur on the arcs of the underlying graphs. When dealing with a finite graph there will be only finitely many value expressions passed as parameters as a formula is unwound. Consider the following example of a formula which demonstrates that the expressive power of our logic is not compromised excessively under parameter restriction: The abstraction A_{fib} is restricted but makes an essential use of parameterisation:

$$A_{fib} = \mu X.[c!x](x = z_1 \wedge [c!y](y = x + z_2 \wedge X.(x + y, y/z_1, z_2))).$$

We see that the formula $A_{fib}.(1, 0/z_1, z_2)$ states "I can perform a finite output stream on channel c which follows the Fibonacci sequence."

The key result of this section then is that, under reasonable conditions, the proof system of Section 3 is complete for finite symbolic graphs and restricted parameter formulae.

Theorem 9 *(Completeness) For all formulae F with empty tag sets, finite $\mathcal{G}$, $fv(B) \subseteq fv(t)$,*

$$t \models_B F \text{ implies } B \vdash t : F.$$

□

$$
\begin{array}{lcl}
[\![B']\!]_s\rho B\widehat{\varepsilon} & = & \left\{\begin{array}{ll} \mathcal{T}(\mathcal{G}) & \text{If } B\widehat{\varepsilon} \models B' \\ \emptyset & \text{Otherwise}\end{array}\right. \\
[\![F \wedge F']\!]_s\rho B\widehat{\varepsilon} & = & [\![F]\!]_s\rho B\widehat{\varepsilon} \cap [\![F']\!]_s\rho B\widehat{\varepsilon} \\
[\![F \vee F']\!]_s\rho B\widehat{\varepsilon} & = & \bigcup\{[\![F]\!]_s\rho B_1\widehat{\varepsilon} \cap [\![F']\!]_s\rho B_2\widehat{\varepsilon} \mid B\widehat{\varepsilon} \models B_1 \vee B_2\} \\
[\![\langle\tau\rangle F]\!]_s\rho B\widehat{\varepsilon} & = & \left\{\begin{array}{l} t \mid \exists\{c_i\}_I \cdot B\widehat{\varepsilon} \models \bigvee_I c_i, \forall i.\exists t \xrightarrow{b_i,\tau} t_i' \text{ with } c_i \models b_i \\ \qquad\qquad \text{and } t_i' \in [\![F]\!]_s\rho(B \wedge c_i)\widehat{\varepsilon}\end{array}\right\} \\
[\![[\tau]F]\!]_s\rho B\widehat{\varepsilon} & = & \left\{t \mid \forall t \xrightarrow{b',\tau} t' \text{ implies } t' \in [\![F]\!]_s\rho(B \wedge b')\widehat{\varepsilon}\right\} \\
[\![\langle c!x\rangle F]\!]_s\rho B\widehat{\varepsilon} & = & \left\{\begin{array}{l} t \mid \exists\{c_i\}_I . B\widehat{\varepsilon} \models \bigvee_I c_i \cdot \forall i.\exists t \xrightarrow{b_i,c!e_i} t_i' \text{ with } c_i \models b_i \\ \qquad\qquad \text{and } t_i' \in [\![F[e_i/x]]\!]_s\rho(B \wedge c_i)\widehat{\varepsilon}\end{array}\right\} \\
[\![[c!x]F]\!]_s\rho B\widehat{\varepsilon} & = & \left\{t \mid \forall t \xrightarrow{b',c!e} t' \text{ implies } t' \in [\![F[e/x]]\!]_s\rho(B \wedge b')\widehat{\varepsilon}\right\} \\
[\![\langle c?\rangle G]\!]_s\rho B\widehat{\varepsilon} & = & \left\{\begin{array}{l} t \mid \exists\{c_i\}_I . B\widehat{\varepsilon} \models \bigvee_I c_i \cdot \forall i.\exists t \xrightarrow{b_i,c?} (y_i)t_i' \text{ with } c_i \models b_i \\ \qquad\qquad \text{and } (y_i)t_i' \in [\![G]\!]_s\rho(B \wedge c_i)\widehat{\varepsilon}\end{array}\right\} \\
[\![[c?]G]\!]_s\rho B\widehat{\varepsilon} & = & \left\{t \mid \forall t \xrightarrow{b',c?} (y)t' \text{ implies } (y)t' \in [\![G]\!]_s\rho(B \wedge b')\widehat{\varepsilon}\right\} \\
[\![\exists x.F]\!]_s\rho B\widehat{\varepsilon} & = & \left\{\begin{array}{l} (y)t \mid \exists \text{ new } w, b(w) \cdot B\widehat{\varepsilon} \models \exists w.b(w) \\ \qquad\qquad \text{and } t[w/y] \in [\![F[w/x]]\!]_s\rho(B \wedge b(w))\widehat{\varepsilon}\end{array}\right\} \\
[\![\forall x.F]\!]_s\rho B\widehat{\varepsilon} & = & \{(y)t \mid \exists \text{ new } w \cdot t[w/y] \in [\![F[w/x]]\!]_s\rho B\widehat{\varepsilon}\} \\
[\![A.(\tilde{e}/\tilde{z})]\!]_s\rho B\widehat{\varepsilon} & = & ([\![A]\!]_s\rho)B[\widehat{\varepsilon(\tilde{e})}/\tilde{z}] \\
[\![X]\!]_s\rho & = & \rho(X) \\
[\![\mu X[\mathcal{A}]F]\!]_s\rho & = & \mu f.(\lambda b.[\![F]\!]_s\rho[f/X]b \setminus \lambda\mathcal{A}) \\
[\![\nu X[\mathcal{A}]F]\!]_s\rho & = & \nu f.(\lambda b.[\![F]\!]_s\rho[f/X]b \cup \lambda\mathcal{A})
\end{array}
$$

where $\lambda\mathcal{A}(b) = \{t \mid (b', t) \in \mathcal{A} \text{ and } b \models b'\}$ and $\setminus$ and $\cup$ denote pointwise set difference and union respectively.

Figure 9: Symbolic interpretations of formulae

The remainder of this section is devoted to establishing this. We design a new semantics, called the *symbolic semantics*, for the logic to get our completeness result. A symbolic interpretation takes a recursion environment and a boolean expression, rather than an evaluation, as an environment for the free data variables. It will be useful to maintain, throughout the completeness proof, a very strict form for these boolean expressions. We assume that they have the form

$$B \wedge (\tilde{z} = \tilde{e})$$

where B is a boolean expression not containing any recursion parameters and $\tilde{e}$ is a finite vector of data expressions not containing any recursion parameters. For notational convenience we describe such a boolean as follows. Let ε range over substitutions of the form $[\tilde{e}/\tilde{z}]$ where $\tilde{e}$ contains no recursion parameters (note that the identity substitution is the special case of this where the vectors are zero length). Given $\varepsilon = [\tilde{e}/\tilde{z}]$ and a boolean expression B not containing recursion parameters we write $\widehat{\varepsilon}$ for the boolean expression $\tilde{z} = \tilde{e}$ and $B\widehat{\varepsilon}$ for $B \wedge \widehat{\varepsilon}$. Using this strict form of boolean environment we present the symbolic interpretation of our logic in Figure 9. The first thing that we ought to check is that these symbolic semantics coincide in some way with the evaluation based

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

Proposition 10 *If F (not necessarily recursion closed) has empty tag sets then $t \models_{\rho,B\widehat{\varepsilon}} F$ iff $t \models^s_{\rho^*,B\widehat{\varepsilon}} F$ (where $t \in \rho^*(X)B\widehat{\varepsilon}$ iff $\delta \models B\widehat{\varepsilon}$ implies $t\delta \in \rho(X)\delta$).*

Proof. We use structural induction on F. Some of the cases are shown below.

Case: Recursion variable.

$t \models_{\rho,B\widehat{\varepsilon}} X$ iff $t \in \rho(X)\delta$ whenever $\delta \models B\widehat{\varepsilon}$. This amounts to saying $t \in \rho^*(X)B\widehat{\varepsilon}$, which is just $t \models^s_{\rho^*,B\widehat{\varepsilon}} X$.

Case: Disjunction.

The if direction is straightforward. $t \models^s_{\rho^*,B\widehat{\varepsilon}} F_1 \vee F_2$ implies that there exists two booleans B_1, B_2 such that $B\widehat{\varepsilon} \models B_1 \vee B_2$ with $t \models^s_{\rho^*,B_i\widehat{\varepsilon}} F_i$, for $i = 1, 2$. By induction we get $t \models_{\rho,B_i\widehat{\varepsilon}} F_i$ for $i = 1, 2$ so whenever $\delta \models B\widehat{\varepsilon}$ we know that $\delta \models B_i$ for $i = 1$ or $i = 2$, in either case $t\delta \in [\![F_1 \vee F_2]\!]\rho\delta$. Hence $t \models_{\rho,B\widehat{\varepsilon}} F_1 \vee F_2$.

Conversely, suppose that $t \models_{\rho,B\widehat{\varepsilon}} F_1 \vee F_2$. Let (for $i = 1, 2$) B_i be defined by $\delta \models B_i$ iff $\delta \models B\widehat{\varepsilon}$ and $t\delta \in [\![F_i]\!]\rho\delta$. Then $B\widehat{\varepsilon} \models B_1 \vee B_2$ by hypothesis and induction gives $t \models^s_{\rho^*,b_i\widehat{\varepsilon}} F_i$ for $i = 1, 2$. Thus $t \models^s_{\rho^*,B\widehat{\varepsilon}} F_1 \vee F_2$.

Case:$\langle\tau\rangle$.

Suppose $t\delta \in [\![\langle\tau\rangle F]\!]\rho\delta$ whenever $\delta \models B\widehat{\varepsilon}$. Let $\{(b_i, t'_i)\}_I = \left\{(b', t') \mid t \xrightarrow{b',\tau} t'\right\}$. So we know that whenever $\delta \models B\widehat{\varepsilon}$ there exists an $i \in I$ such that $\delta \models b_i$ and $t'_i\delta \in [\![F]\!]\rho\delta$. Define c_i so that $\delta' \models c_i$ iff $\delta' \models (B \wedge b_i)\widehat{\varepsilon}$. We then know $c_i \models b_i$, $B\widehat{\varepsilon} \models \bigvee_I c_i$ and, by induction, $t_i \in [\![F]\!]_s \rho^*(B \wedge c_i)\widehat{\varepsilon}$.

Conversely suppose that $t \in [\![\langle\tau\rangle F]\!]_s \rho^* B\widehat{\varepsilon}$. By definition we know that there exists $\{c_i\}_I$ such that $B\widehat{\varepsilon} \models \bigvee_I c_i$ and for each $i \in I$ there is a $t \xrightarrow{b_i,\tau} t'$ such that $c_i \models b_i$ and $t' \in [\![F]\!]_s \rho^*(B \wedge c_i)\widehat{\varepsilon}$. Let $\delta \models B\widehat{\varepsilon}$, then $\delta \models c_i$ for some i. Thus $\delta \models b_i$ and we get $t\delta \xrightarrow{\tau} t'\delta$ and, by induction, $t'\delta \in [\![F]\!]\rho\delta$. Therefore $t\delta \in [\![\langle F\rangle]\!]\rho\delta$.

Case: Application.

Suppose $t \models_{\rho,B\widehat{\varepsilon}} A.(\tilde{e}/\tilde{z})$ so that whenever $\delta \models B\widehat{\varepsilon}$ we know $t\delta \in [\![A]\!]\rho\delta[\tilde{e}/\tilde{z}]$. Now $\delta \models \widehat{\varepsilon}$ iff $\delta[\tilde{e}/\tilde{z}] \models \widehat{\varepsilon'}$ where $\varepsilon' = [\varepsilon(\tilde{e})/z]$. Therefore by induction we know $t \models^s_{\rho^*,B\widehat{\varepsilon'}} A$ whence $t \models^s_{\rho^*,B\widehat{\varepsilon}} A.(\tilde{e}/\tilde{z})$. The converse is similar. □

We now see how fruitful the symbolic semantics are by returning to the dual version of the Reduction Lemma, Lemma 1, which failed previously.

Lemma 11 *(Reduction lemma revisited) Let M be the lattice of monotone functions from the partial order $(\mathcal{B}, \leq)$ to $(\mathcal{P}T, \subseteq)$ and let $\varphi : M \to M$ be monotone. Then for any $t \in T, b \in \mathcal{B}$,*

$$t \in \mu x.\varphi(x)(b) \text{ iff } t \in \varphi(\mu x.(\varphi(x) \setminus \lambda(b,t)))(b)$$

where $\lambda(b,t)(b') = \begin{cases} \{t\} & \text{If } b' \leq b \\ \emptyset & \text{otherwise.} \end{cases}$

Proof. The if direction is easy. For the only if direction we suppose that $t \in \mu x.\varphi(x)(b)$. Standard fixpoint theory [8] tells us that there exists an ordinal α such that $t \in \varphi^\alpha(b)$ where $\varphi^0 = \lambda\emptyset$ (the constant empty function), $\varphi^{n+1} = \varphi(\varphi^n)$ and $\varphi^\gamma = \bigcup_{\beta<\gamma} \varphi^\beta$ when γ

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

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

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 $t \,\mathrm{sat}\, F$ 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 $B\widehat{\varepsilon}$ statements. This requires knowing the ε part of the environment when calculating $(t \,\mathrm{sat}\, F)$. Figure 10 shows how this is done. For each variable X, each term t, and each environment ε, we have a new variable $X_{t\varepsilon}$ and create a formula $\nu X_{t\varepsilon} \ldots$. We do not require a similar construction for least fixpoints here because, as we saw in Proposition 4, any proof involving least fixpoints can be transformed into a proof involving finite unwindings and

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 is also well-founded for formulae of this sublogic; this depends on the fact that only a finite number of substitutions, ε, are used as we unwind a formula against a finite graph.

Proposition 13 *If $\mathcal{T}(\mathcal{G})$ is finite then $\ll$ is well-founded on parameter and tag restricted formulae.*

Proof. We suppose without loss of generality that our fixpoint formulae only use a single recursion parameter each. Because the new notion of tag restriction allows a term t to appear several times in each tag set, one for each different ε, it is sufficient, in light of Proposition 5, to check that we only encounter finitely many ε environments as we calculate $(\varepsilon \triangleright t \,\mathrm{sat}\, F)$. By inspecting the definition of $(\varepsilon \triangleright t \,\mathrm{sat}\, F)$ we notice that given an $\varepsilon = [e/z]$ then a new ε' is created only at the application stage, that is

$$\varepsilon \triangleright t \,\mathrm{sat}\, A.(e'/z') = \varepsilon' \triangleright t \,\mathrm{sat}\, A$$

where $\varepsilon' = [\varepsilon(e')/z']$. Now the restriction on parameters tells us that either

- $\varepsilon' = [e/z']$ when e' is simply the parameter z or
- $\varepsilon' = [e'/z']$ when e' does not contain recursion parameters.

This observation allows us to describe a general form which the ε must satisfy as we calculate $(\varepsilon \triangleright t \,\mathrm{sat}\, F)$. We need to describe the possible data expressions which e, e' may be.

We define *DApps* inductively over the depth of formulae in Figure 11 and note that $DApps(F)$ is finite for any formula F. We let

$$Outs(\mathcal{T}(\mathcal{G})) = \left\{ e \in ValExp \mid \exists t, t' \in \mathcal{T}(\mathcal{G}).t \xrightarrow{b, c!e} t' \text{ for some } b, c \right\}$$

be the collection of the data expressions which appear on the output arcs of $\mathcal{T}(\mathcal{G})$. For finite $\mathcal{T}(\mathcal{G})$ we see that this set is also finite. Let $NV(\mathcal{T}(\mathcal{G}), F)$ be the finite stack of variables used to for creating new variables. Let $BV?(F)$ be the variables bound by quantifiers in F and let $BV!(F)$ be the variables bound by $\langle c! \rangle$ and $[c!]$ modalities in F. We may assume that $BV?(F)$ and $BV!(F)$ are disjoint by α-conversion and they are clearly both finite.

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

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

All environments ε' used when calculating $\varepsilon \triangleright t \,\mathrm{sat}\, F$ can be described in the general form

$$[e[\tilde{w}, \tilde{e}'/\tilde{x}, \tilde{y}]/z']$$

where $e \in DApps(F)$, $\tilde{x} \in BV?(F)$, $\tilde{y} \in BV!(F)$, $\tilde{w} \in NV(\mathcal{T}(\mathcal{G}), F)$, $\tilde{e}' \in Outs(\mathcal{T}(\mathcal{G}))$ and $z' \in RPar(F)$. All of the above sets are finite hence there can be finitely many different ε. □

Proposition 14 *For all formulae F with empty tag sets, finite $\mathcal{G}$*

$$B\hat{\varepsilon} \models \varepsilon \triangleright t \; sat \, F \;\; \textit{iff} \;\; t \models^s_{B\hat{\varepsilon}} F.$$

Proof. We prove by well-founded induction on tag restricted F that, with η defined as in Theorem 6, we have

$$B\hat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, F]\!]\eta \text{ iff } t \models^s_{B\hat{\varepsilon}} F.$$

. We outline a few cases here:

The cases for the fixpoints use Lemmas 2 and 12.

Case: Disjunction.

Suppose $t \in [\![F_1 \vee F_2]\!]_s \rho B\hat{\varepsilon}$ so that $t \in [\![F_1]\!]_s \rho B_1\hat{\varepsilon}$ and $t \in [\![F_2]\!]_s \rho B_2\hat{\varepsilon}$ for some B_1, B_2 such that $B\hat{\varepsilon} \models B_1 \vee B_2$. By induction we know that $B_1\hat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, F_1]\!]\eta$ and $B_2\hat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, F_2]\!]\eta$. Thus $B\hat{\varepsilon} \models (B_1 \vee B_2)\hat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, F_1 \vee \varepsilon \triangleright t \,\mathrm{sat}\, F_2]\!]\eta \equiv [\![\varepsilon \triangleright t \,\mathrm{sat}\, (F_1 \vee F_2)]\!]\eta$. The converse is given immediately by induction.

Case: $[c!]$.

Suppose that $t \in [\![[c!x]F]\!]_s \rho B\hat{\varepsilon}$. We know that whenever we have a transition $t \xrightarrow{b', c!e'} t'$ we have $t' \in [\![F[e'/x]]\!]_s \rho(B \wedge b')\hat{\varepsilon}$. If we assume that $\delta \models B\hat{\varepsilon}$ and $\delta \models b'$ for some b' such that $t \xrightarrow{b', c!e'} t'$ then by induction we know that $\delta \models [\![\varepsilon \triangleright t' \,\mathrm{sat}\, F[e'/x]]\!]\eta$ and thus $B\hat{\varepsilon} \models [\![b' \to \varepsilon \triangleright t' \,\mathrm{sat}\, F[e'/x]]\!]\eta$ for all such b'. Whence $B\hat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, [c!x]F]\!]\eta$.

Conversely, suppose that $B\hat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, [c!x]F]\!]\eta$. Then $(B \wedge b')\hat{\varepsilon} \models [\![\varepsilon \triangleright t' \mathrm{sat}\, F[e'/x]]\!]\eta$ for each b', e' such that $t \xrightarrow{b', c!e'} t'$. This implies that $t' \in [\![F[e'/x]]\!]_s \rho(B \wedge b')\hat{\varepsilon}$ and consequently $t \in [\![[c!x]F]\!]_s \rho B\hat{\varepsilon}$.

Case: *Application*.

Firstly, let ε' denote $[\varepsilon(\tilde{e})/\tilde{z}]$ and suppose $t \in [\![A.(\tilde{e}/\tilde{z})]\!]_s \rho B\hat{\varepsilon}$. Then $t \in [\![A]\!]_s \rho B\widehat{\varepsilon'}$ and by induction we know $B\widehat{\varepsilon'} \models [\![\varepsilon' \triangleright t \,\mathrm{sat}\, A]\!]\eta$. Let $\delta \models B\hat{\varepsilon}$ so that $\delta[\tilde{e}/\tilde{z}] \models B\widehat{\varepsilon'}$. This gives

us that $\delta[\tilde{e}/\tilde{z}] \models [\![\varepsilon' \triangleright t \,\mathrm{sat}\, A]\!]\eta$ and because $\tilde{z}$ does not appear free in $[\![\varepsilon' \triangleright t \,\mathrm{sat}\, A]\!]\eta$ we have $\delta \models [\![\varepsilon' \triangleright t \,\mathrm{sat}\, A]\!]\eta \equiv [\![\varepsilon \triangleright t \,\mathrm{sat}\, A.(\tilde{e}/\tilde{z})]\!]\eta$.

Conversely, $B\widehat{\varepsilon} \models [\![\varepsilon \triangleright A.(\tilde{e}/\tilde{z})]\!]\eta$ implies that $B\widehat{\varepsilon} \models [\![\varepsilon' \triangleright t \,\mathrm{sat}\, A]\!]\eta$. Now, because $\tilde{z}$ does not appear free in $[\![\varepsilon' \triangleright t \,\mathrm{sat}\, A]\!]\eta$ we have that $B\widehat{\varepsilon'} \models B \models [\![\varepsilon' \triangleright t \,\mathrm{sat}\, A]\!]\eta$ and hence, by induction, $t \in [\![A.(\tilde{e}/\tilde{z})]\!]_s \rho B\widehat{\varepsilon}$. □

Lemma 15 *For all tag restricted formulae F, finite $\mathcal{G}$, η as above,*

$$[\![(\varepsilon \triangleright t \; sat F)]\!]\eta\widehat{\varepsilon} \vdash t : F.$$

Proof. Again we use $\ll$ for well-founded induction on F. For the most part the proof is similar to that in [6] with the following notable differences:

The base case for atomic propositions requires that $B[\varepsilon(\tilde{z})/\tilde{z}] \wedge \widehat{\varepsilon} \vdash t : B$. This follows from Id and Cons because $B[\varepsilon(\tilde{z})/\tilde{z}] \wedge \widehat{\varepsilon} \models B$.

The case for application goes as follows: We know by induction that $[\![(\varepsilon' \triangleright t \,\mathrm{sat}\, A)]\!]\eta\widehat{\varepsilon'} \vdash t : A$ is derivable (where $\varepsilon' = [\varepsilon(\tilde{e})/\tilde{z}]$). Using rule App we get $[\![(\varepsilon' \triangleright t \,\mathrm{sat}\, A)]\!]\eta\widehat{\varepsilon'}[\tilde{e}/\tilde{z}] \vdash t : A.(\tilde{e}/\tilde{z})$. Then, as $\widehat{\varepsilon} \models \widehat{\varepsilon'}[\tilde{e}/\tilde{z}]$, by Cons we see that $[\![(\varepsilon \triangleright t \,\mathrm{sat}\, A.(\tilde{e}/\tilde{z}))]\!]\eta\widehat{\varepsilon} \vdash t : A.(\tilde{e}/\tilde{z})$.

For fixpoints we simply use induction and the unfolding rules when t, ε is not in the tag set and use rule ν_0 or an empty Cut otherwise. □

Proof. (Theorem 9, Completeness)

We prove a slightly different statement, that is, $t \models^s_{B\widehat{\varepsilon}} F$ implies $B\widehat{\varepsilon} \vdash t : F$ and witness the theorem, via Proposition 10, as an instance of this when ε is the identity. Suppose $t \models^s_{B\widehat{\varepsilon}} F$, then by Proposition 14 we get $B\widehat{\varepsilon} \models \varepsilon \triangleright t \,\mathrm{sat}\, F$. The previous Lemma provides a derivation of $(\varepsilon \triangleright t \,\mathrm{sat}\, F)\widehat{\varepsilon} \vdash t : F$ so an application of Cons will complete the proof. □

6 Example

We now present an extended example of a proof that a finite symbolic graph, compiled from a process declaration, satisfies a property expressed in our logic. In particular, the property is expressed using restricted parameters only and uses both minimal and maximal fixpoint formulae. The process that we declare allows an input stream of non-negative integers on channel i and for each input received peforms an output of the maximum value received so far on channel o. The process may be rendered as the parallel composition of three components: A starting process St which simply outputs the value 0 on an internal channel k and then stops, a process M which determines the greater of its two inputs on i and k and sends the result on another internal channel m, and finally a process Sp which splits the input received on channel m by sending it on both o and k. The whole process is illustrated in Figure 12.

Using this description a syntactic version of this process Max, is easily obtained.

$$Max \stackrel{def}{=} (St \mid M \mid Sp) \setminus \{k, m\}$$

where

- $St \Leftarrow k!0$
- $M \Leftarrow k?x.i?y.(x > y \rightarrow m!x.M, m!y.M)$
  $\quad + i?y.k?x.(x > y \rightarrow m!x.M, m!y.M)$
- $Sp \Leftarrow m?x.o!x.k!x.Sp$

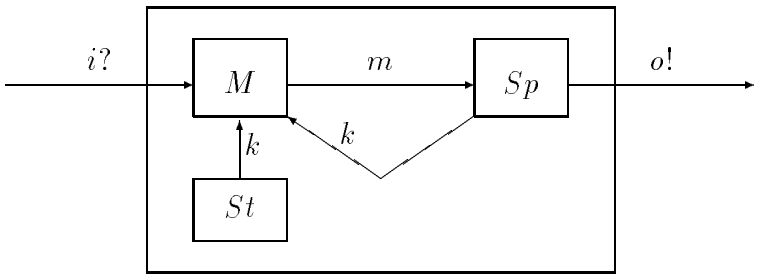

Figure 12: Flow diagram for process Max.

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.

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

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

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

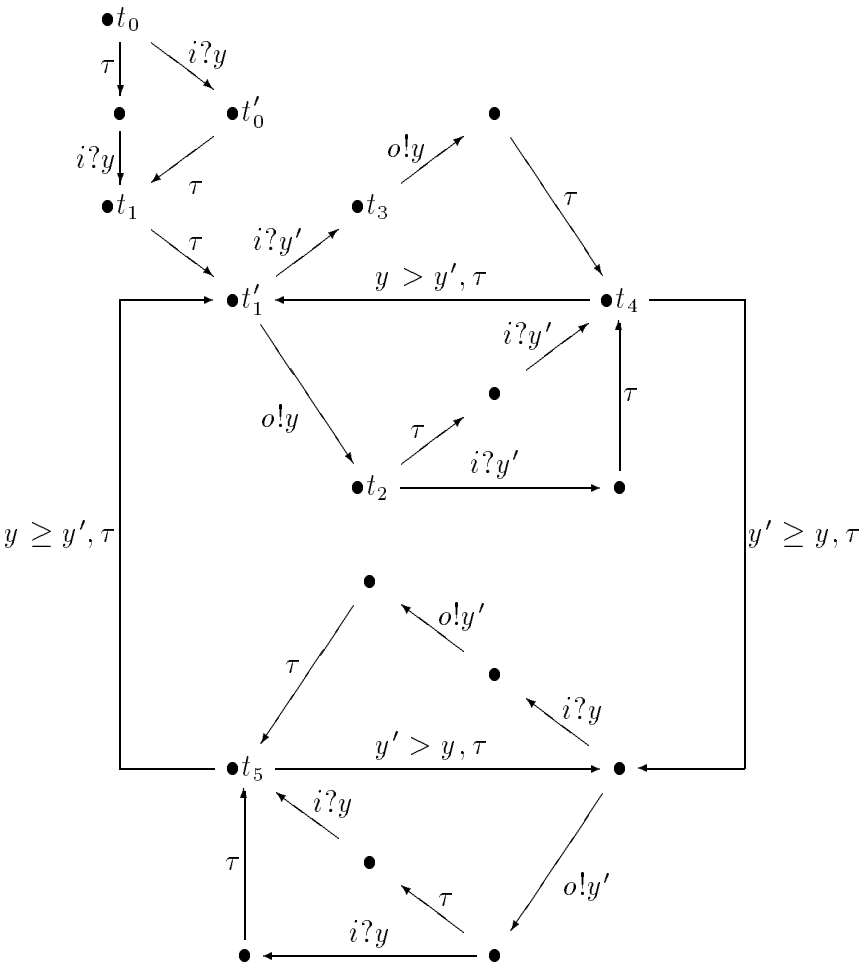

Figure 13: Symbolic graph for Max.

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 $\forall, [\tau], [i?], \wedge, \mu$ and *App* if we can establish

$$\mathbf{tt} \vdash t'_0, t_1 : A.(y, 0/z, z').$$

These can be obtained by using *App* and ν_1 unfolding from

$$\widehat{\varepsilon_0} \vdash t'_0, t_1 : (F_1 \wedge F_2)[A_1/X]$$

where $\widehat{\varepsilon_0} \equiv z = y \wedge z' = 0$ and $A_1 = \nu X[\mathcal{A}]F_1 \wedge F_2$ with $\mathcal{A} = (\widehat{\varepsilon_0}, t'_0)$ (or, accordingly, $(\widehat{\varepsilon_0}, t_1)$). These judgements can be broken up into the four judgements

$$\widehat{\varepsilon_0} \vdash t'_0, t_1 : F_1[A_1/X] \text{ and } \widehat{\varepsilon_0} \vdash t'_0, t_1 : F_2[A_1/X]$$

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) first: this judgement can be divided, using $\wedge$, and then each branch dealt with by using the appropriate modality and μ unfolding rules to obtain the sequent $y > y' \wedge \widehat{\varepsilon_1} \vdash t_4 : F_3[A_2/x][y/x]$. In each case we are using the τ transition going back to node t_1' and following the diamond of transitions to return to t_4. To close this branch of proof we note that $y > y' \wedge \widehat{\varepsilon_1} \models B_{Max}[y/x]$, so we can reduce the statement to $y > y' \wedge \widehat{\varepsilon_1} \vdash t_4 : A_2.(y', y/z, z')$ and, after using rule App, that ν_0 is applicable as $(\widehat{\varepsilon_1}, t_4)$ is in the tag set of A_2.

We must now turn our attention to establishing (8). Similarly we can divide the judgement into two using $\wedge$ and in both cases follow the appropriate transitions down to the lower part of the graph and through the diamond to get to

$$y' \geq y \wedge \widehat{\varepsilon_1} \vdash t_5 : F_3[A_2/X][y'/x].$$

This judgement is reduced to $y' \geq y \wedge \widehat{\varepsilon_1} \vdash t_5 : A_2.(y, y'/z, z')$ by noting that $y' \geq y \wedge \widehat{\varepsilon_1} \models B_{Max}[y'/x]$ and using the $\wedge$ and Cons rules. We then must apply rule App to get

$$\widehat{\varepsilon_2} \vdash t_5 : A_2$$

where $\widehat{\varepsilon_2} \equiv z = y \wedge z = y'$. Now ν_0 is not yet applicable so we must unfold once more to get

$$\widehat{\varepsilon_2} \vdash t_5 : (F_1 \wedge F_2)[A_3/X]$$

where $A_3 = \nu X[\mathcal{A}, (\widehat{\varepsilon_1}, t_4), (\widehat{\varepsilon_2}, t_5)]F_1 \wedge F_2$. The proof now continues in a similar manner to before; we do a case analysis on y and y' to reduce to

$$y' > y \wedge \widehat{\varepsilon_2} \vdash t_5 : (F_1 \wedge F_2)[A_3/X] \text{ and } y \geq y' \wedge \widehat{\varepsilon_2} \vdash t_5 : (F_1 \wedge F_2)[A_3/X].$$

The left branch follows the modalities back around the lower diamond of the graph and uses the tag $(\widehat{\varepsilon_2}, t_5)$ to close the proof. The right branch uses the τ transition travelling back to the upper part of the graph and uses the tag $(\widehat{\varepsilon_1}, t_4)$ to close the proof and then we are done.

7 Conclusions

In this paper we have shown how to extend the local model checking procedures of [13] to *value-passing* processes. The central idea, based on that in [6], is to have an auxiliary proof system for data-expressions and to express the model checking rules relative to this auxiliary system. By working at the level of *symbolic transition systems* we can then imitate the tag method for fixpoint formulae which was introduced in [13]; inevitably the tags are more complicated, consisting of pairs of process terms and boolean expressions.

We hope that these model checking procedures can be implemented along the lines of the *PAM* system, [9], which checks semantic equivalences between process terms. A major research problem here is to design an effective and efficient interface between auxiliary proof systems for checking data expressions and the main model checking routines. Another fruitful line of research would be the investigation of the characteristic formulae $(T \,\mathrm{sat}\, F)$ which are automatically generated by the completeness theorems. These are in general very complicated but a calculus of reductions may be found which could systematically simplify them and in particular eliminate many of the uses of fixpoint expressions.

On the more theoretical level it is clear that more powerful proof rules are required for least fixpoint formulae. As our counter-example shows it is possible to use minimal fixpoints in the property language to code up the inductive nature of the data domain (in this case the integers). To handle these kinds of descriptions it will be necessary to invent proof rules which incorporate induction on the data domain with the more structural rules for fixpoints.

The completeness programme follows that of [6] in that a reduction of the statement $t \models F$ to a first-order formula $t \,\mathrm{sat}\, F$ is given. However because of the presence of parameterised fixpoint formulae the reduction is more complicated and the resulting boolean expressions are also more complicated since they involve maximal and minimal fixpoint operators. The idea of reducing satisfaction to a system of boolean equations is not uncommon in model checking for the pure modal μ-calculus; examples can be found in [2, 3].

A similar logic was presented in [4] for describing properties of mobile processes. This logic also featured parameterised fixpoints although the language of boolean expressions used there was restricted to basic statements about name matching and the parameters were just vectors of names. A model checker for this logic was defined and a symbolic approach to the semantics of this logic was used to obtain a completeness result. This model checker did not use tag sets but a recent paper by Amadio and Dam [1] provides a generalisation of the tag set method for mobile processes. Their specification logic

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 *pointwise*. 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. Crubille. A linear algorithm to solve fixed-point equations on transition systems. *Information Processing Letters*, 29:57–66, 1988.

[3] R. Cleaveland, M. Dreimüller, and B. Steffen. Faster model checking for the modal mu-calculus. In *CAV'92*, volume 663 of *Lecture Notes in Computer Science*, pages 383–394. Springer-Verlag, 1993.

[4] M. Dam. Model checking mobile processes. In E. Best, editor, *Proceedings CONCUR 93,* Hildesheim, volume 715 of *Lecture Notes in Computer Science*, pages 22–36. Springer-Verlag, 1993.

[5] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[6] M. Hennessy and X. Liu. A modal logic for message passing processes. Technical report, Computing Science Dept. University of Sussex, 1993.

[7] K.G.Larsen. Proof systems for Hennessy-Milner logic with recursion. *Lecture Notes in Computer Science*, page 299, 1988.

[8] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–353, 1983.

[9] Huimin Lin. PAM: A Process Algebra Manipulator (Version 1.0). Computer Science Report 4/93, University of Sussex, February 1993.

[10] R. Milner. *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs, 1989.

[11] C. Stirling. Modal and temporal logics. In S. Abramsky, D. Gabbay, and T.Maibaum, editors, *Handbook of Logic in Computer Science, Vol I.* Oxford University Press, 1990.

[12] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89:161–177, 1991.

[13] G. Winskel. A note on model checking the modal nu-calculus. *Theoretical Computer Science*, 83:157–167, 1991.