Model checking Demos using PBI: A Simple Approach

Jonathan Hayman
HP Laboratories Bristol
HPL-2003-196
October 7th , 2003*

E-mail: jmh00@doc.ic.ac.uk

model checking, demos, bunched implications, concurrency

The application of a resource logic to the non-temporal analysis of processes acting on resources, 'Hayman 2003. This defines the non-temporal semantics of simulation language, Demos, in order to be able to model processes acting on shared resources. It also defines the semantics of a logic, PBI, an extension of the Logic of Bunched Implications, to query these models. This document describes a simple model checker capable of checking a subclass of PBI formulae against Demos models. It is assumed that the reader is familiar with, 'The application of resource logic to the non-temporal analysis of processes acting on resources,' Hayman 2003.

Model checking Demos using **PBI**: A Simple Approach

Jonathan Hayman[*]
Hewlett-Packard Laboratories, Bristol

Summer 2003

Abstract

[Hay03] defines the non-temporal semantics of a simulation language, Demos, in order to be able to model processes acting on shared resources. It also defines the semantics of a logic, **PBI**, an extension of the Logic of Bunched Implications [POY02, Pym02, OP99], to query these models.

This document describes a simple model checker capable of checking a subclass of **PBI** formulæ against Demos models. It is assumed that the reader is familiar with [Hay03].

1 Introduction

It is difficult to reason about how processes running in parallel or concurrently may interact. This comes from the fact that a process may *interfere* with other processes acting in their environment.

Demos is a system for simulating processes interacting on shared resources. Though we are interested in analysis rather than simulation, we shall use the syntax of Demos to represent processes. In [Hay03], we defined a subclass of Demos, σDemos, that models the salient characteristics of non-temporal process interaction and proved our interpretation of such interaction to be correct.

Defined in [Hay03], **PBI** is a logic of processes and resources. It was defined, in particular, to query σDemos models, and allows properties such as deadlock freedom and sub-component deadlock freedom to be expressed.

This document describes the operation of a (semi-)recursively defined program capable of checking an interesting subclass σDemos models against an interesting subclass of **PBI** formulæ.

2 A logic for model checking

PBI was presented in [Hay03] as a Hennessy-Milner style logic (HML). Though useful for proving properties of systems, the inclusion of infinitary conjunction

[*]`jmh00@doc.ic.ac.uk`

(and disjunction) in HML necessitates a deduction system that is able to reason about arbitrary infinitary modal formulæ. As alluded to by the inclusion of sections on CTL and modal-μ in [Hay03], this makes the HML logic not particularly suited to a the development of a simple model checking algorithm.

2.1 CTL$^-$

CTL [CES86] is a relatively simple temporal logic particularly suited to model checking. As well as the finite part of HML, there are two additional modalities relating to properties holding over paths[1].

The first, $\forall[\phi \mathbf{U} \psi]$, expresses that along every path derivable from a given state, ϕ holds until ψ holds, and ψ does eventually hold. The second, $\exists[\phi \mathbf{U} \psi]$, expresses that along some path from the current state, ϕ holds until ψ holds, and ψ does eventually hold.

$s_0 \models \mathrm{p}$	iff	$s_0 \in \lVert \mathrm{p} \rVert$
$s_0 \models \neg\phi$	iff	$s_0 \not\models \phi$
$s_0 \models \phi \wedge \psi$	iff	$s_0 \models \phi$ and $s_0 \models \psi$
$s_0 \models \phi \vee \psi$	iff	$s_0 \models \phi$ or $s_0 \models \psi$
$s_0 \models \langle - \rangle \phi$	iff	for some state s' and action $\alpha \in \mathcal{A}$, $s_0 \xrightarrow{\alpha} s'$ and $s' \models \phi$.
$s_0 \models [-]\phi$	iff	for all states s' and actions α, $s_0 \xrightarrow{\alpha} s'$ implies $s' \models \phi$
$s_0 \models \forall[\phi \mathbf{U} \psi]$	iff	for all paths $[s_0, s_1, \ldots]$, $\exists i \geqslant 0[\ s_i \models \psi \wedge \forall j[0 \leqslant j < i \implies s_j \models \phi]\ ]$
$s_0 \models \exists[\phi \mathbf{U} \psi]$	iff	for some path $[s_0, s_1, \ldots]$, $\exists i \geqslant 0[\ s_i \models \psi \wedge \forall j[0 \leqslant j < i \implies s_j \models \phi]\ ]$

Table 1: Semantics of CTL formulæ

Simpler still is CTL$^-$. The four operators, defined in terms of the CTL modalities above, allow us to simply express many interesting properties.

- $s \models \exists \mathbf{F} \phi \overset{\text{def}}{\iff} s \models \exists[\top \mathbf{U} \phi]$
  This means that there is a path in which ϕ holds in some future state — ϕ *potentially* holds.
- $s \models \forall \mathbf{F} \phi \overset{\text{def}}{\iff} s \models \forall[\top \mathbf{U} \phi]$
  This means that along every path from s, there is a state in which ϕ holds — ϕ is *inevitable.*

[1]We shall use the definition of path found in [Hay03] rather than [CES86]: A path is a *possibly* infinite sequence of states $[s_0, s_1, \ldots]$ such that $\forall i.\ s_i \longrightarrow s_{i+1}$

- $s \models \exists\mathbf{G}\phi \quad \overset{\text{def}}{\iff} \quad s \models \neg\forall[\top\mathbf{U}\neg\phi]$
  This means that there is a path from s where ϕ holds in every state.
- $s \models \forall\mathbf{G}\phi \quad \overset{\text{def}}{\iff} \quad s \models \neg\exists[\top\mathbf{U}\neg\phi]$
  This means that for every path from s, ϕ holds in every state — ϕ is *globally* true.

It is interesting to note that CTL is heavily restricted by its notion of path: it does not take into account that transitions may be labelled or allow quantification over particular types of state. Consequently, it does not correspond directly to the native modalities of **PBI**, $\langle A\rangle$, $\langle A\rangle^{+}$, $\langle A\rangle^{-}$ and the box modalities; rather, it corresponds to the derived $_{\mathbf{any}}$ modalities.

The reader is referred to [Hay03] for a slightly more verbose account of CTL and pointers into the literature.

2.2 Modal-μ

In essence, the modal-μ calculus allows recursive propositions to be defined. Take, for example, the formula

$$Z \overset{\text{def}}{=} \langle A\rangle\top \wedge [A]Z;$$

Z holds in states that it is possible to derive a transition from (states that have a *derivative*) where all next states have this property, too.

For an arbitrary formula $\phi(Z)$ with a free logical variable Z, it may be the case that there are a number of solutions for the set of states in which Z holds satisfying $Z = \phi(Z)$. Each such set is called a *fixed point* of $Z = \phi(Z)$. Of particular interest are the least fixed points and the greatest fixed points; these will exist if $\phi(Z)$ is *monotonic*. $s \models \mu Z.\phi(Z)$ is defined to hold iff s is in the least set of states such that $Z = \phi(Z)$, and $\nu Z.\phi(Z)$ is defined to hold iff s is in the greatest set of states such that $Z = \phi(Z)$. μ and ν are called *fixpoint operators*. Again, the reader is referred to [Hay03] for a more verbose account of this, and to [BS01] for an excellent introduction to the μ-calculus.

As modal-μ relies neither upon presupposed diamond or box modalities nor upon a notion of path, the fixpoint operators are directly applicable to **PBI**. Model checking modal-μ in the simplest case involves iteratively labelling states to reach a fixed point.

3 Implementation

Though the CTL^{-} temporal operators do not directly correspond to the modalities of **PBI**, they do allow many interesting questions to be posed in the logic. As our goal is to create a very simple model checker in a functional language, we shall use CTL^{-} rather than modal-μ.

We have only considered the core part of the Demos language in this implementation. For simplicity, we shall have a `repeat`$\{P\}$ structure, equivalent to `while` [`true`] `do` P, rather than `while`.

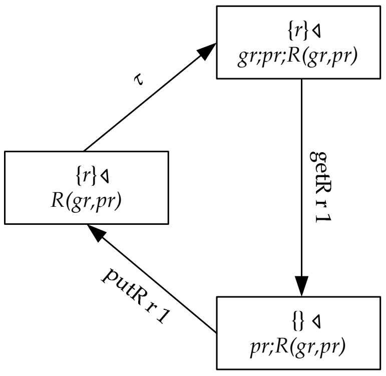
3.1 State space generator

The first component of the model checker is the state space generator. For a given state, this returns a graph where the nodes are reachable σDemos states and the arcs represent one-step transitions; the arcs are labelled by the action labels. Nodes also contain a list of propositions used in the evaluation of CTL$^-$ modal formulæ.

For example, the state space of

$$\mathbf{R} \lhd [\![P]\!] \equiv \wr r \int \lhd [\![\texttt{repeat}\{\texttt{getR}(r,1);\texttt{putR}(r,1)\}]\!]$$

is (abbreviating $\texttt{getR}(r,1)$ by gr, $\texttt{putR}(r,1)$ by pr and $\texttt{repeat\{P\}}$ by $R(P)$)



Our approach to building the state space graph is to add a node and then recursively add all its derivatives (thereby performing a *depth-first* expansion). When we encounter a node already in the graph, we assume that it has already been expanded. This avoids the insertion duplicate states and recursing infinitely on cycles.

To data structure created is implemented, for performance reasons, using SML pointers to nodes. Because the state space may be large, modifications of the state space are usually defined imperatively rather than recursively. In particular, modifications to the list of propositions used in checking the CTL$^-$ modalities shall not result in a new state space object being generated.

3.2 Logic component

The second component of the model checker is the logic unit. This uses the state space generator to check whether or not propositions in CTL hold in a given state. The checker function operating on a state $\mathbf{R} \lhd [\![P]\!]$ in a state space $\mathsf{space}(\mathbf{S} \lhd [\![Q]\!])$ to check a proposition ϕ,

$$\mathsf{check}\ \phi\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{S} \lhd [\![Q]\!]),$$

is defined recursively as follows.

3.2.1 Logical units and propositions

$\top$ holds in any state, and $\bot$ holds in none. I_P holds in states where there is only a null process; I_R holds in states where there is no available resource. Resource propositions hold where the state comprises only that resource, with no process. Process propositions hold where the state comprises only that process, with no resources.

- $\mathsf{check}\ \top\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!]) \ \overset{\text{def}}{=}\ \top$.
- $\mathsf{check}\ \bot\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!]) \ \overset{\text{def}}{=}\ \bot$.
- $\mathsf{check}\ I_R\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!]) \ \overset{\text{def}}{=}\ \mathbf{R} \lhd [\![P]\!] \equiv \wr\,\int \lhd [\![P]\!]$.
- $\mathsf{check}\ I_P\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!]) \ \overset{\text{def}}{=}\ \mathbf{R} \lhd [\![P]\!] \equiv \mathbf{R} \lhd [\![\varepsilon]\!]$.
- $\mathsf{check}\ I\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!]) \ \overset{\text{def}}{=}\ \mathbf{R} \lhd [\![P]\!] \equiv \wr\,\int \lhd [\![\varepsilon]\!]$.
- $\mathsf{check}\ r\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!]) \ \overset{\text{def}}{=}\ \mathbf{R} \lhd [\![P]\!] \equiv \wr r \int \lhd [\![\varepsilon]\!]$, for resource atoms r.
- $\mathsf{check}\ Q\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!]) \ \overset{\text{def}}{=}\ \mathbf{R} \lhd [\![P]\!] \equiv \wr\,\int \lhd [\![Q]\!]$, for process atoms Q.

3.2.2 Additives

The conjunction of two propositions holds if both hold in the given state; the disjunction holds if one or both hold. As the logic is boolean (because the monoid uses an equality), to check implication we first check the antecedent. If it is false, we return true. Otherwise, we return the result of checking the sequent. We check both propositions in the same state space graph.

- $\mathsf{check}\ (\phi \wedge \psi)\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!]) \ \overset{\text{def}}{=}\ \mathsf{check}\ \phi\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!])$ and also $\mathsf{check}\ \psi\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!])$. (Similarly for disjunction).

- $\mathsf{check}\ (\psi \rightarrow \psi)\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!]) \ \overset{\text{def}}{=}\ \mathsf{if\ check}\ \phi\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!])$ then $\mathsf{check}\ \psi\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!])$, otherwise true.

3.2.3 Multiplicatives

The multiplicatives are slightly more complicated. One approach, which we did not adopt, to checking $\mathbf{R} \lhd [\![P]\!] \models \phi * \psi$ is to determine from the syntax of ϕ the resource and process component required to make ϕ true. We subtract this from $\mathbf{R} \lhd [\![P]\!]$, and check in the resulting state space ψ. This, however, would not be complete — we would have trouble checking *e.g.,* $\top * \psi$.

An alternative approach to checking $\mathbf{R} \lhd [\![P]\!] \models \phi * \psi$, which we did adopt, is to check whether there is any partition of $\mathbf{R} \lhd [\![P]\!]$ with one part satisfying ϕ and the other satisfying ψ. Though this inefficiently involves checking all possible sub-partitions, it is immediate from the definition of the semantics of $*$ that this is complete for all formulæ (assuming completeness for checking subformulæ). Each sub-partition has its own state space expanded to form a new graph.

It is not possible to extend this approach to checking $\phi \multimap\!\!* \, \psi$; to do so would involve checking whether all process resource pairs satisfying ϕ, when added to the current process resource pair, make ψ true. This, in turn, would require us to check *every* possible process resoruce pair, which is impossible as the set of such pairs is infinite.

Instead, for the multiplicative implication, we adopt the first approach, defining a partial function syntactically from propositions to resource-process pairs that indicates the component necessary to make the antecedent true. The partial function, defined below, shall be undefined where we have more or less than one pair making the antecedent true, *e.g.,* I_R or $\bot$. We take $\setminus$ to be multiset subtraction.

$$\begin{aligned}
\mathsf{resources}\ I &\overset{\text{def}}{=} \wr\,\int \\
\mathsf{resources}\ r &\overset{\text{def}}{=} \wr\, r\,\int \\
\mathsf{resources}\ P &\overset{\text{def}}{=} \wr\,\int \\
\mathsf{resources}\ \phi * \psi &\overset{\text{def}}{=} (\mathsf{resources}\ \phi) \uplus (\mathsf{resources}\ \psi) \\
\mathsf{resources}\ \phi \mathrel{-\!\!*} \psi &\overset{\text{def}}{=} (\mathsf{resources}\ \psi) \setminus (\mathsf{resources}\ \phi)
\end{aligned}$$

Similarly, the process component required to make a formula true is defined as follows. In this case, $\setminus$ is process subtraction.

$$\begin{aligned}
\mathsf{processes}\ I &\overset{\text{def}}{=} [\![\varepsilon]\!] \\
\mathsf{processes}\ r &\overset{\text{def}}{=} [\![P]\!] \\
\mathsf{processes}\ P &\overset{\text{def}}{=} [\![\varepsilon]\!] \\
\mathsf{processes}\ \phi * \psi &\overset{\text{def}}{=} (\mathsf{processes}\ \phi) \mid (\mathsf{processes}\ \psi) \\
\mathsf{processes}\ \phi \mathrel{-\!\!*} \psi &\overset{\text{def}}{=} (\mathsf{processes}\ \psi) \setminus (\mathsf{processes}\ \phi)
\end{aligned}$$

It follows that the model checker is complete with respect to a subclass of antecedents, Φ_a:

$$\Phi_a ::= r \mid P \mid I \mid \Phi_a * \Phi_a \mid \Phi_a \mathrel{-\!\!*} \Phi_a$$

Implementationally, where $\mathsf{resources}\ \phi$ or $\mathsf{processes}\ \phi$ is undefined, an exception shall be raised thus preserving the soundness of the system. Again, we construct a new graph object for the state space of the new state.

- $\mathsf{check}\ (\phi * \psi)\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!]) \ \overset{\text{def}}{=}$
  for some partition $\mathbf{R} \lhd [\![P]\!] \equiv (\mathbf{S} \lhd [\![Q]\!]) \circ (\mathbf{S}' \lhd [\![Q']\!])$,
  $\mathsf{check}\ \phi\ (\mathbf{S} \lhd [\![Q]\!])\ \mathsf{space}(\mathbf{S} \lhd [\![Q]\!])$ and also $\mathsf{check}\ \phi\ (\mathbf{S}' \lhd [\![Q']\!])\ \mathsf{space}(\mathbf{S} \lhd [\![Q]\!])$.
- Let $\mathbf{S} \lhd [\![Q]\!]$ be $(\mathsf{resources}\ \phi) \lhd (\mathsf{processes}\ \phi)$ in
  $\mathsf{check}\ (\phi \mathrel{-\!\!*} \psi)\ (\mathbf{R} \lhd [\![P]\!])\ \mathsf{space}(\mathbf{R} \lhd [\![P]\!]) \ \overset{\text{def}}{=}$
  $\mathsf{check}\ \psi\ (\mathbf{R} \uplus \mathbf{S} \lhd [\![P \| Q]\!])\ \mathsf{space}(\mathbf{R} \uplus \mathbf{S} \lhd [\![P \| Q]\!])$

3.2.4 HM Modalities

To check $\mathbf{R} \triangleleft [\![P]\!] \models \langle - \rangle_{\mathbf{any}} \phi$, all we need do is check if ϕ holds in any state adjacent in the state space graph to $\mathbf{R} \triangleleft [\![P]\!]$. This can be checked in the same graph. Similarly, $\mathbf{R} \triangleleft [\![P]\!] \models [-]_{\mathbf{any}} \phi$ involves checking ϕ in all states adjacent in the state space graph to $\mathbf{R} \triangleleft [\![P]\!]$. At this stage, we choose not to implement the specific additive and multiplicative modalities of **PBI**, nor checking against labels.

- $\mathsf{check}\ (\langle - \rangle_{\mathbf{any}} \phi)\ (\mathbf{R} \triangleleft [\![P]\!])\ \mathsf{space}(\mathbf{R} \triangleleft [\![P]\!]) \overset{\text{def}}{=}$ for some state $\mathbf{S} \triangleleft [\![Q]\!]$ connected by an arc from $\mathbf{R} \triangleleft [\![P]\!]$, $\mathsf{check}\ \phi\ (\mathbf{S} \triangleleft [\![Q]\!])\ \mathsf{space}(\mathbf{R} \triangleleft [\![P]\!])$.
- $\mathsf{check}\ ([-]_{\mathbf{any}} \phi)\ (\mathbf{R} \triangleleft [\![P]\!])\ \mathsf{space}(\mathbf{R} \triangleleft [\![P]\!]) \overset{\text{def}}{=}$ for all states $\mathbf{S} \triangleleft [\![Q]\!]$ connected by an arc from $\mathbf{R} \triangleleft [\![P]\!]$, $\mathsf{check}\ \phi\ (\mathbf{S} \triangleleft [\![Q]\!])\ \mathsf{space}(\mathbf{R} \triangleleft [\![P]\!])$.

3.2.5 CTL$^-$ modalities

Recall from [Hay03] the definition of the CTL$^-$ modalities in modal-μ:

- $\forall \mathbf{G} \phi \overset{\text{def}}{\iff} \nu Z. \phi \wedge [-] Z$;
- $\exists \mathbf{G} \phi \overset{\text{def}}{\iff} \nu Z. \phi \wedge (\langle - \rangle Z \vee [-] \bot)$;
- $\forall \mathbf{F} \phi \overset{\text{def}}{\iff} \mu Z. \phi \vee ([-] Z \wedge \langle - \rangle Z)$;
- $\exists \mathbf{F} \phi \overset{\text{def}}{\iff} \mu Z. \phi \vee \langle - \rangle Z$.

These provide a useful intuition as to how the CTL$^-$ modalities shall be recursively defined in SML.

Let us first consider $\forall \mathbf{G} \phi$. The modal-μ formula defines it to hold in states where ϕ holds and in all derivatives $\forall \mathbf{G} \phi$ holds. In order to prevent recursing infinitely on cycles, we shall mark all states in which we are evaluating $\forall \mathbf{G} \phi$. Note that the marking has to be $\forall \mathbf{G} \phi$ (or, alternatively just ϕ) rather than setting a boolean in order to allow arbitrary nesting of CTL$^-$ modalities. If we revisit a node (*i.e.,* visit a node marked $\forall \mathbf{G} \phi$), we know that ϕ will hold on every node through this cycle, so we take this to be a true evaluation of $\forall \mathbf{G} \phi$ for this state — though if other paths from the node fail to have ϕ holding throughout, some other conjunct shall fail. Taking a marked state as true for $\forall \mathbf{G} \phi$ is alluded to by the use of the greatest fixpoint operator, ν, as opposed to μ.

Similarly, $\exists \mathbf{G} \phi$ holds where we have ϕ holding in the current state and $\exists \mathbf{G} \phi$ holding in some derivative if there is one. Again, we take revisiting a marked node to be a positive result.

$\forall \mathbf{F} \phi$, which holds where ϕ holds in the current state or where all derivatives have $\forall \mathbf{F} \phi$ holding and there is a derivative, is slightly different. If we encounter a cycle (revisit a state marked $\forall \mathbf{F} \phi$), then no state on this cycle satisfies ϕ. Thus not every path has a state satisfying ϕ, so we return a negative result by returning false on this conjunct. This is alluded to by the use of the least fixpoint operator, μ, in the modal-μ definition.

$\exists\mathbf{F}\phi$, similarly, holds if ϕ holds in the current state or there is a derivative in which $\exists\mathbf{F}\phi$ holds. Again, we take revisiting a node to mean that the particular disjunct corresponding to this path does not hold.

We let $\mathsf{marked}(\mathbf{R} \triangleleft [\![P]\!], \phi, G)$ hold if the state $\mathbf{R} \triangleleft [\![P]\!]$ is marked with a formula ϕ in G, and let $\mathsf{mark}(\mathbf{R} \triangleleft [\![P]\!], \phi, G)$ be a function that returns the state space G with $\mathbf{R} \triangleleft [\![P]\!]$ marked ϕ. The derivatives of $\mathbf{R} \triangleleft [\![P]\!]$ in G shall be $\mathsf{derivs}(\mathbf{R} \triangleleft [\![P]\!], G)$. We shall temporarily abbreviate $\mathsf{check}\ \phi\ (\mathbf{S} \triangleleft [\![Q]\!])\ G$ to $\mathsf{C}_G^{\mathbf{S}\triangleleft[\![Q]\!]}\phi$.

$$\mathsf{C}_G^{\mathbf{R}\triangleleft[\![P]\!]}\forall\mathbf{G}\phi \quad \overset{\text{def}}{=} \quad \mathsf{C}_G^{\mathbf{R}\triangleleft[\![P]\!]}\phi \wedge \begin{cases} \top, \text{ if } \mathsf{marked}(\mathbf{R} \triangleleft [\![P]\!], \forall\mathbf{G}\phi, G) \\ \bigwedge_{\mathbf{S}\triangleleft[\![Q]\!]\in\mathsf{derivs}(\mathbf{R}\triangleleft[\![P]\!],G)} \mathsf{C}^{\mathbf{S}\triangleleft[\![Q]\!]}_{\mathsf{mark}(\mathbf{R}\triangleleft[\![P]\!],\forall\mathbf{G}\phi,G)}\forall\mathbf{G}\phi \end{cases}$$

$$\mathsf{C}_G^{\mathbf{R}\triangleleft[\![P]\!]}\exists\mathbf{G}\phi \quad \overset{\text{def}}{=} \quad \mathsf{C}_G^{\mathbf{R}\triangleleft[\![P]\!]}\phi \wedge \begin{cases} \top, \text{ if } \mathsf{marked}(\mathbf{R} \triangleleft [\![P]\!], \exists\mathbf{G}\phi, G) \\ \bigvee_{\mathbf{S}\triangleleft[\![Q]\!]\in\mathsf{derivs}(\mathbf{R}\triangleleft[\![P]\!],G)} \mathsf{C}^{\mathbf{S}\triangleleft[\![Q]\!]}_{\mathsf{mark}(\mathbf{R}\triangleleft[\![P]\!],\exists\mathbf{G}\phi,G)}\exists\mathbf{G}\phi \end{cases}$$

$$\mathsf{C}_G^{\mathbf{R}\triangleleft[\![P]\!]}\forall\mathbf{F}\phi \quad \overset{\text{def}}{=} \quad \mathsf{C}_G^{\mathbf{R}\triangleleft[\![P]\!]}\phi \vee \begin{cases} \bot, \text{ if } \mathsf{marked}(\mathbf{R} \triangleleft [\![P]\!], \forall\mathbf{F}\phi, G) \\ \bigwedge_{\mathbf{S}\triangleleft[\![Q]\!]\in\mathsf{derivs}(\mathbf{R}\triangleleft[\![P]\!],G)} \mathsf{C}^{\mathbf{S}\triangleleft[\![Q]\!]}_{\mathsf{mark}(\mathbf{R}\triangleleft[\![P]\!],\forall\mathbf{F}\phi,G)}\forall\mathbf{F}\phi \end{cases}$$

$$\mathsf{C}_G^{\mathbf{R}\triangleleft[\![P]\!]}\exists\mathbf{F}\phi \quad \overset{\text{def}}{=} \quad \mathsf{C}_G^{\mathbf{R}\triangleleft[\![P]\!]}\phi \vee \begin{cases} \bot, \text{ if } \mathsf{marked}(\mathbf{R} \triangleleft [\![P]\!], \exists\mathbf{F}\phi, G) \\ \bigvee_{\mathbf{S}\triangleleft[\![Q]\!]\in\mathsf{derivs}(\mathbf{R}\triangleleft[\![P]\!],G)} \mathsf{C}^{\mathbf{S}\triangleleft[\![Q]\!]}_{\mathsf{mark}(\mathbf{R}\triangleleft[\![P]\!],\exists\mathbf{F}\phi,G)}\exists\mathbf{F}\phi \end{cases}$$

It should be noted that the recursive calls to evaluate the CTL$^-$ modalities must delete any labels they have placed. Not only may they interfere with succeeding evalutions (*e.g.,* consider the evaluation of $\exists\mathbf{F}\langle-\rangle_{\mathbf{new}}\top \wedge \exists\mathbf{F}\langle-\rangle_{\mathbf{new}}\top$ — the root state would be labelled, thus immediately giving a false result), but they may interfere within recursive calls to the evaluation of the multiplicative modality. As the labels are only used for detecting cycles, they should persist through checking derivatives but should be removed through checking sibling states.

Finally, to show that our labelling approach generalises to nested CTL$^-$ formulæ, notice that the definition of CTL$^-$ formulæ is well founded; there is no formula ϕ such that $\phi = \mathbf{M}\phi$ for some modality $\mathbf{M}$. Therefore, we will never have a collsion of labels.

Observing the above definitions, we see that the system is complete (modulo the exclusive inclusion of the $_{\mathbf{any}}$ modalities as opposed to the complete set of additive and multiplicative modalities) other than in multiplicative implication. The following grammar describes the formulæ for which the checker is complete:

$$\begin{array}{rcl} \Phi & ::= & \top \mid \bot \mid I \mid I_R \mid I_P \mid r \mid \Phi \\ & & \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \rightarrow \Phi \\ & & \mid \Phi * \Phi \mid \Phi_a \multimap\!\!* \ \Phi \\ & & \mid \langle-\rangle_{\mathbf{any}}\Phi \mid [-]_{\mathbf{any}}\Phi \\ & & \mid \forall\mathbf{G}\Phi \mid \exists\mathbf{G}\Phi \mid \forall\mathbf{F}\Phi \mid \exists\mathbf{F}\Phi \\ & & \mid \neg\Phi \end{array}$$

4 System interaction

ASCII being somewhat inadequate for writing logical formulæ, our implementation of the model checker in SML alters the notations used in our theoretical account. Table 2 gives the representations used in the implementation.

All binary operators are right-associative, the precedence being as follows:

$$\neg \succ * \succ -\!\!* \succ \wedge \succ \vee \succ \rightarrow,$$

where $\succ$ is to be read as "binds more strongly than".

We also introduce an extra-logical construct `PRINT` ϕ which binds as tightly as $\neg$. This prints the current state and returns the result of evaluating ϕ. As the evaluation of booleans in SML is lazy (*e.g.,* in checking $\phi \vee \psi$, if ϕ is true, ψ will not be evaluated), this can be used to see the states where a proposition fails, where it passes and the partition allocated to a particular proposition. Suppose, for example, that we want to see the reachable states in which a proposition ϕ does not hold; we would query using the checker-proposition

ϕ `|| PRINT FF`

To see where ϕ did hold, we would use

ϕ `&& PRINT TT`

Suppose we want to see which partitions satisfy ϕ; we would use

`(`ϕ `&& PRINT TT) ** TT`

Conversely, to see what is left after satisfying ϕ,

ϕ `** PRINT TT`

4.1 Intelligence Agency example

Recall from [Hay03] the intelligence agency example. We had two processes, one owned by the CIA and one owned by MI6, that acquired and released control of four communications channels. The Demos code is given in Figure 1.

Stripping away the delcarations, we obtain the following state for the model checker:

```
([''6'',''5'',''C'',''F''],
 [P[getR(''6'',1),hold 2, getR(''5'',1),hold 2, getR(''F'',1),
    hold 2, hold 4, putR(''F'',1), putR(''5'',1), putR(''6'',1)],
  P[getR(''C'',1),hold 2, getR(''F'',1),hold 2, getR(''5'',1),
    hold 2, hold 4, putR(''5'',1), putR(''F'',1), putR(''C'',1)]])
```

The formula for deadlock freedom (for convenience, predefined by the system as `DF`), is

`AG(D TT || IP)`

Supposing the above state is defined as `si`, we can check for deadlock freedom using the `checkst` function:

[2] `->` is reserved in SML

Item	**Representation**	**Example(s)**
Resources		
Resource	string	$r \mapsto$ `''r''`
Resource multiset	List of strings	$\lbag r, s, t \rbag \mapsto$ `[''r'',''s'',''t'']`
Processes		
ε	`Null`	$\llbracket \varepsilon \rrbracket \mapsto$ `Null`
Non-parallel process	`P`[List of commands]	$\llbracket \texttt{getR}(r, 1); \texttt{putR}(r, 1) \rrbracket \mapsto$ `P[getR(''r'',1),putR(''r'',1)]`
Process	List of non-parallel processes	$\llbracket \varepsilon \Vert \texttt{getR}(r, 1); \texttt{putR}(r, 1) \rrbracket \mapsto$ `[Null,P[getR(''r'',1),putR(''r'',1)]]`
State	(Resource mset, Process)-pair	$\lbag r, s \rbag \lhd \llbracket \varepsilon \rrbracket \mapsto$ `([''r'',''s''],[Null])`
Propositions		
$\top$	`TT`	
$\bot$	`FF`	
I_R	`IR`	
I_P	`IP`	
I	`I`	
r	`R` resource	$r \mapsto$ `R''r''`
P	`Pr` process	$\varepsilon \mapsto$ `Pr[Null]`
$\neg\psi$	`~(`ψ`)`	$\neg\bot \mapsto$`~(FF)`
$\phi \wedge \psi$	ϕ `&&` ψ	
$\phi \vee \psi$	ϕ `||` ψ	
$\phi \rightarrow \psi$	ϕ `Imp` ψ[2]	
$\phi * \psi$	ϕ `**` ψ	
$\phi \mathrel{-\!*} \psi$	ϕ `-*` ψ	
$\langle - \rangle_{\textbf{any}}\phi$	`D` ϕ	$\langle - \rangle_{\textbf{any}}\top \mapsto$ `D TT`
$[-]_{\textbf{any}}\phi$	`B` ϕ	
$\forall\mathbf{G}\phi$	`AG` ϕ	
$\exists\mathbf{G}\phi$	`EG` ϕ	
$\forall\mathbf{F}\phi$	`AF` ϕ	
$\exists\mathbf{F}\phi$	`EF` ϕ	

Table 2: SML representations

```
cons ACT_TIME6 4;
cons ACT_TIMECIA 4;

class MI6 = {
  getR(MI6_line, 1); hold(2);
  getR(MI5_line, 1); hold(2);
  getR(FBI_line, 1); hold(2);

  hold(ACT_TIME6);

  putR(FBI_line, 1); putR(MI5_line, 1); putR(MI6_line, 1);
}

class CIA = {
  getR(CIA_line, 1); hold(2);
  getR(FBI_line, 1); hold(2);
  getR(MI5_line, 1); hold(2);

  hold(ACT_TIMECIA);

  putR(MI5_line, 1); putR(FBI_line, 1); putR(CIA_line, 1);
}

Res(CIA_line, 1); Res(FBI_line, 1);
Res(MI6_line, 1); Res(FBI_line, 1);

Entity(CIA, 12*60 + 40); %12:40
Entity(MI6, 12*60 + 43); %12:43
```

Figure 1: Demos program for the intelligence agency example

```
- checkst DF si;
val it = false : bool
```

If we want to see the state that is deadlocked, we enter the query

```
- checkst (AG(D TT || IP || PRINT FF)) si;
(())#[[getR(F,1),putR(F,1),putR(5,1),putR(6,1)],
      [getR(5,1),putR(5,1),putR(F,1),putR(C,1)]]
val it = false : bool
```

The result is exactly as we proposed in [Hay03].

Finally, recall that we argued that adding a new FBI line would remove deadlock. To check:

```
- checkst (R''F''-*DF) si;
val it = true : bool
```

5 Conclusions

We have described a (relatively) simple, recursive approach to model checking Demos using **PBI**. With it, we are able to analyse the essentials of process interaction through shared resource, for example deadlock.

It remains to implement the remainder of the system described in [Hay03]. Specifically, the following items remain:

PBI modalities It would not be difficult to extend the existing system to include the basic modalities of **PBI**, $\langle A\rangle_{\mathbf{new}}$ and $\langle A\rangle$ as opposed to solely implementing $\langle -\rangle_{\mathbf{any}}$ here (and the corresponding box-modalities). This would allow us to query, for example,

$$\forall\mathbf{F}(\langle -\rangle_{\mathbf{new}}\top \vee I_P),$$

which asks if every path eventually performs a resource action or terminates — in a way checking that a process makes progress.

Action labels Our implementation does not allow restriction of modalities to particular sets of labels. So, for example, $\langle -\rangle_{\mathbf{any}}$ is allowed, but for a restrictive action label set A, $\langle A\rangle_{\mathbf{any}}$ is not allowed. To implement the full labelling system described in [Hay03] would also require that processes are labelled.

Weak modalities Desribed in [Hay03] are *weak*, or *observable*, modalities. Rather than being defined in terms of the transition relation $\xrightarrow{\sigma}$, which allows observable or hidden actions to be derived, they are defined in terms of $\overset{\sigma}{\Longrightarrow}$, which allows for the arbitrary prefix and suffix of silent actions to one observable action.

Extension to σDemos The full transition system of [Hay03] was not implemented: we did not include synchronisation, entity declarations, or a whole host of other features of Demos.

Fair execution In concluding [Hay03], it was remarked that the system could be extended to consider only fair and just execution sequences. Unlike the above extensions, this would require further theoretical consideration.

Synchronous execution As described in [Hay03], a rule for synchronous parallel execution may be implemented to reduce the size of the state spaces considered by the model checker. It may be possible to determine syntactically whether or not a given formula will be equivalent in the synchronous and asynchronous systems and, if so, to use the synchronous system. At the very least it is possible to do this for deadlock freedom.

Naturally, these aspects were omitted because our consideration here is of a simple model checker for Demos. Though the extensions described above should not be very hard to implement, it probably would take a good deal of work. It would also be possible to write a model checker capable of expanding the state space as required, rather than in advance as described here. For example, if we were to evaluate $\langle - \rangle_{\mathbf{any}} \top$ in a model that has a very large state space, it seems very wasteful to expand the whole state space in advance. Alternatively, more recent approaches to model checking could be applied, *e.g.,* using game theory.

Of course, if we were to implement such an elaborate system, we should also reconsider the use of the CTL$^-$ modalities. Though they are suited to a simple implementation, the fixpoint operators of modal-μ would allow a much richer class of formulæ to be expressed, especially given the variety of one-step modalities offered by **PBI**.

References

[BS01] Julian C. Bradfield and Colin Stirling. Modal logics and mu-calculi: an introduction. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 239–330. Elsevier, http://www.dcs.ed.ac.uk/~jcb/, 2001.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[Hay03] Jonathan Hayman. The application of a resource logic to the non-temporal analysis of processes acting on resources. Technical report, Hewlett-Packard Laboratories, Bristol, 2003.

[OP99] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–243, June 1999.

[POY02] David J. Pym, Peter W. O'Hearn, and Hongseok Yang. Possible worlds and resources: The semantics of **BI**. *Journal of Theoretical Computer Science (to appear)*, 2002.

[Pym02] David J. Pym. *The Semantics and Proof Theory of the Logic of Buched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, Dordrecht, July 2002.