A PVS Theory of Symbolic Transition Systems

Savi Maharaj

Department of Computing Science and Mathematics
The University of Stirling
Stirling FK9 4LA,
Scotland, UK
`savi@cs.stir.ac.uk`

Abstract. This paper describes some work carried out in the context of a project aimed at developing tools for reasoning about the formal description technique Full LOTOS. One of the tasks within this project was the development of a theory of *symbolic transition systems* for Full LOTOS, and of a corresponding modal logic. This task was carried out with the help of the PVS theorem prover, which served as a convenient vehicle for experimenting with alternative definitions of the concepts being developed. The proof of correctness of the modal logic was also partially carried out within PVS. Our main conclusion is that the discipline imposed by use of the fully formal PVS notation provided valuable help with the detection and correction of errors and omissions in our experimental definitions, thereby making it easier to arrive at a correct set of definitions.

1 Introduction

The ISO standard formal description technique, Full LOTOS [1], is a widely-used formal method. One of the key strengths of LOTOS is that it combines a process algebra (called Basic LOTOS, and deriving from both CCS[10] and CSP[8]) with an algebraic specification language, ACT ONE [5] for specifying the properties of datatypes. This combination makes for a highly expressive language which has been applied to a broad variety of applications. However, one of the problems which has hampered the use of LOTOS is a shortage of tools such as model-checkers to allow analysis of LOTOS specifications.

One reason why tool development has been slow is that the standard semantics of LOTOS very quickly gives rise to infinite data structures, which are not amenable to analysis by computer. The standard semantics of a LOTOS process is a *structured, labelled, transition system*, which is basically a state-transition graph in which every edge is labelled by an event. These events may consist of a gate name alone (*simple events*) or of a gate name together with a data value (*structured events*). Infinitely-branching transition systems arise when a LOTOS specification contains a structured event involving a variable, where there are infinite possible values for that variable. For example, the event `g? x:nat`, meaning roughly "input any natural number `x` at gate `g`", will give rise to an infinitely branching transition system, containing one branch for each possible value of `x`.

The solution which we have explored is the development of an alternate semantics, based upon *symbolic* transition systems (STSs). The basic intuition behind a symbolic transition system is that its transitions may be labelled by data expressions involving variables, rather than specific data values. Additionally, each transition is labelled by a boolean expression (possibly involving variables) representing the conditions under which that transtion is possible. The use of STSs give rise to much smaller representations for LOTOS processes involving data.

STSs were introduced by Hennessy and Lin[7] as a means of giving a symbolic semantics to value-passing CCS. We have adapted this work to suit Full LOTOS. The details of our symbolic semantics for LOTOS may be found in [4].

This paper describes part of the next stage of our project, in which we develop a logic for reasoning about STSs. This logic was developed in parallel with a definition of *symbolic bisimulation* on STSs. In order to keep track of the many details of these definitions, and to manage version control as we experimented with alternative possibilities, we chose to encode the definitions within the PVS theorem prover [9]. PVS was also used to assist with the proof of a result relating the equivalence induced by the logic to symbolic bisimulation.

The rest of this paper is structured as follows: Section 2 gives the definition of a symbolic transition system (STS) and shows how this was represented in PVS. Section 3 describes the problem of defining syntactic substitution on STSs, and how this was dealt with. Section 4 gives the definition of the logic of the modal logic, FULL, that we have developed for Full LOTOS, and describes how this logic was encoded within PVS. Section 6 briefly describes how the correctness of the logic was demonstrated, and discusses to what extent this proof was carried out using PVS. Section 7 concludes.

2 Symbolic Transition Systems

We shall assume that we have a countable set of *variables*, Var, ranged over by x, y, etc., and a (possibly infinite) set of *values*, Val, ranged over by v. We also assume a set of *data expressions*, Exp, which includes Var and Val and is ranged over by E, and a set of *boolean expressions*, BoolExp, ranged over by b. We also assume that we have a set of gates, G, ranged over by g. The set of simple events, SimpleEv, ranged over by a, is defined as G $\cup$ $\{\mathbf{i}, \delta\}$. (In LOTOS **i** represents a silent, internal event and δ is a special event which takes place when a process is exited.) The set of structured events, StructEv contains all gate-expression combinations gE, as well as all combinations δE. Since the two kinds of structured events are handled exactly the same, we have chosen to ignore δ in this paper, treating it as if it were a member of G. For simplicity, we do not allow structured events consisting of multiple data expressions; only *singleton* data offers are allowed. It is possible, but tedious, to extend our analysis to the case of multiple data offers.

Basically, an STS is a directed graph whose nodes are tagged with sets of free variables, and whose branches are labelled with a boolean condition (the *transition condition*) and an event. Formally, the definition of STS is as follows:

Definition 1. *(Symbolic Transition Systems) A symbolic transition system consists of:*

- *a set of states, containing a distinguished initial state,* T_0*, with each state* T *tagged with a set of free variables, denoted* $fv(T)$*.*
- *a set of transitions written as* $T \xrightarrow{b \quad \alpha} T'$*,*
  where $\alpha \in SimpleEv \cup StructEv$ *and* b *is a Boolean expression and* $fv(T') \subseteq fv(T) \cup fv(\alpha)$ *and* $fv(b) \subseteq fv(T) \cup fv(\alpha)$ *and* $\#(fv(\alpha) - fv(T)) \leq 1$

Following convention, we shall often identify an STS with its initial state. For example, the set of free variables of an STS S, $fv(S)$, is defined as the set of free variables of the initial state of S.

2.1 Representing STSs in PVS

In order to allow simple examples of STSs to be encoded and experimented with, data values were represented by the concrete type `nat`, rather than being left as an abstract parameter. A simple theory of *data expressions* over natural numbers with addition was developed. Similarly, a theory of *boolean expressions* over natural numbers was developed, to be used for representing transition conditions. These theories are omitted for the sake of brevity.

States and *Gates* are represented by type `nat`, wrapped up in appropriate constructors using the inductive datatype mechanism. *Simple* and *structured events* are represented in a similar way. These definitions are straightforward, and are not shown.

A *transition* is then represented as a record containing the source and destination states, the transition condition, and the transition event.

An STS is represented by introducing a type of records consisting of an initial state `t0`, a set of states, a function associating a set of free variables with each state, and a set of transitions. These records are then constrained by a number of predicates representing the various conditions on free variables shown in Definition 2.

3 Handling Substitution

In defining many of the concepts needed for this work, it was necessary to decide how to handle the situation where a value is *substituted* for one of the free variables of an STS. It turns out that naive, syntactic substitution of the value gives rise to incorrect results. This is illustrated in Figure 2. The STS shown diagrammatically on the left represents a simple, one element buffer, and has a single free variable `x`. Suppose that the first action taken by this buffer is to

```
-------------------------------------------------------------
Transition : TYPE = [# source : State,
                       condition  : BoolExp,
                       event  : Event,
                       dest : State
                    #]

pre_STS : TYPE = [# t0 : State,
                    states : setof[State],
                    freevars : [(states) -> setof[Variable]],
                    transitions : setof[Transition] #]

Property1 % ...start state belongs to the set of states
Property2 % ...a transition may introduce at most one new variable
Property3 %
Property4 % Properties 3-6 encode the restrictions on free
Property5 % variables given in Definition 1.
Property6 %

STS : TYPE = ({x:pre_STS | Property1(x) AND Property2(x) AND
                           Property3(x) AND Property4(x) AND
                           Property5(x) AND Property6(x)})
-------------------------------------------------------------
```

Fig. 1. PVS representation of Symbolic Transition Systems

input the value 3. If we perform a simple syntactic substitution of 3 for `x`, we obtain the STS on the right, which represents a buffer that can output the value 3, and is thereafter incapable of inputting any value other than 3. This is clearly incorrect.

The problem with defining substitution on STSs was mentioned briefly in [7], but without a full explanation of the source of the problem. As a result, in the initial, on-paper versions of our work, we assumed that syntactic substitution on STSs was possible, and glossed over the details of how it was carried out. When the work was translated to PVS, however, it became necessary to have a full, formal definition of substitution, and it became apparent that syntactic substitution was not possible in the presence of STSs with cycles (such as the example of Figure 2).

In our view, one of the major benefits of using PVS for this work has been the discipline enforced by the need for full, formal definition, which forces the user to closely scrutinize ideas or assumptions that might otherwise have been left unexamined. In our case, it turned out that a correct understanding of substitution on STSs was vital towards obtaining a correct set of definitions for the modal logic we were trying to develop.

How, then, should substitution be handled? In [7], the problem is solved by introducing the concept of a "term", which consists of an STS paired with a substitution. The idea is that the substitution is carried around together with the STS, and applied only at points when it is meaningful to do so (for example, when a boolean condition is to be evaluated). This solution can successfully be adapted for LOTOS, and, indeed, is the approach that was taken in the final version of this work [2], which was carried out on paper only.

At present, the PVS version of this work utilises a different, stop-gap solution: we simply assume that the STS contains no loops. The reason for this is simply that at the point when it was realized that "terms" were necessary, we found that we could make faster progress by working on paper rather than making the effort required to encode all our definitions in PVS. Essentially, at that point, it seemed that most of the benefits to be reaped from the use of the tool had been realized, and any further effort would be met with diminishing returns. This issue will be discussed further in Section 6.

The work presented in the rest of this paper reflects the developments actually carried out in PVS, and does not use the concept of "terms". Therefore, the logic and the bisimulation relation shown in Sections 4 and 6 are defined on STSs without cycles. It is is straightforward to adapt these definitions to use "terms", and the resulting definitions may be seen in [2] and [3]. In future work, we intend to encode these definitions of within PVS in order to have a complete theory of STSs in PVS.

4 A Modal Logic

The modal logic which we have developed for LOTOS is based on Hennessy and Lin's extension of Hennessy-Milner logic to value-passing CCS [6]. The details

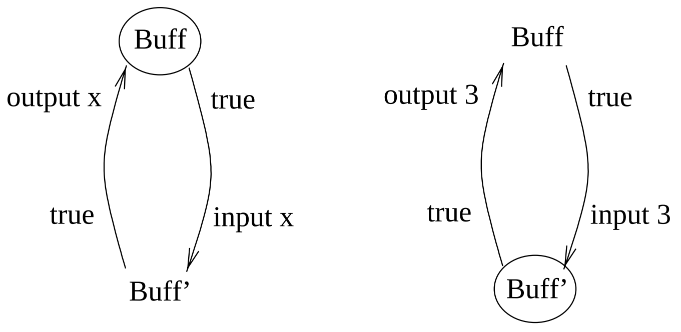

Fig. 2. Failed substitution on `Buff` (initial state is circled)

of the logic are explained in full, with illustrative examples, in [3]. We do not repeat those details here, but instead give a brief summary of the logic.

Syntactically, the logic consists of a basic Hennessy-Milner style logic, extended with four new modalities for expressing properties related to transitions with data. The new modalities are obtained by combining $[\ ]$ and $\langle\ \rangle$ in every possible way with the existential and universal quantifiers for dealing with data. The syntax of the result is shown in Definition 4, where Φ represents formulae to be used to express properties of closed STSs. The logic is defined over an underlying theory of boolean expressions over data values.

Definition 2. *(Syntax of FULL)*

$$\begin{array}{rcl} \Phi & ::= & \mathit{bool} \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [a]\Phi \mid \langle a\rangle\Phi \\ & & \mid \langle\exists x\ g\rangle\Phi \mid \langle\forall x\ g\rangle\Phi \mid [\exists x\ g]\Phi \mid [\forall x\ g]\Phi \end{array}$$

The semantics of the logic is given as a relation between STSs and formulae, written $T \models \Phi$. A version of this semantics, using "terms" rather than STSs, is presented in [3] and the meaning of the new modalities in the logic is fully explained. As an example, here we give the semantics of one of the new modalities, "exists-diamond" which combines $\langle\ \rangle$ with $\exists$.

A closed STS, T, satisfies the formula $\langle\exists x\ g\rangle\Phi$, provided that there is some value v such that one of the following hold:

– there is a transition $T \xrightarrow{\mathrm{tt}\quad gv} T'$ and $T' \models \Phi[v/x]$
– there is a transition $T \xrightarrow{b\quad gz} T'$, where z is a new variable bound by the transition event, and $b[v/z] \equiv \mathrm{tt}$ and $T'[v/z] \models \Phi[v/x]$

The syntax of the logic is encoded in PVS as an inductive type, `Form`. The PVS mechanism for defining recursive functions then provides a convenient method for encoding the semantics of the logic, as well as auxiliary functions such as substitution on formulae. A fragment of the definition of the semantics is shown in Figure 4. Here, `t` ranges over STSs, `f` over formulae, and `enjoys?` represents the $\models$ relation.

```
-----------------------------------------------------------------------
enjoys?(t,f) : RECURSIVE bool =
   CASES f OF

      bool_f (be) : valid? (be),

      and_f(f1,f2) : enjoys?(t,f1) AND enjoys?(t,f2),

      or_f(f1,f2) : enjoys?(t,f1) OR enjoys?(t,f2),

      % ... box, diamond cases not shown

      existsDiamond_f(g,x,f1) :
         EXISTS (v:Value):
            (EXISTS (tr:(structInitTransOnGate(t,g))) :
                IF (boundvars(t,tr) = emptyset)  % tr doesn't bind a variable
                THEN
                   (valid? (equalExp(value(v),data_offer(tr))) AND
                    valid? (condition(tr)) AND
                    enjoys?(post(t,tr), subF(x,v,f1)))
                ELSE
                   EXISTS (y:Variable) : (member(y,boundvars(t,tr)) AND
                       valid? (subB(y,value(v),condition(tr))) AND
                       enjoys?(subSTS(y,v,(post(t,tr))),subF(x,v,f1)))
                ENDIF),

       % ... remaining 3 data modalities not shown

   ENDCASES
   MEASURE depth(f)
----------------------------------------------------------------------
```

Fig. 3. The semantics of the logic in PVS

5 Symbolic Bisimulation

An important aim in the design of the logic was that it should correspond to a natural notion of bisimulation on STSs, in the sense that bisimilar STSs should satisfy exactly the same logical formulae. We say that sugh a logic is *adequate* with respect to the relevant bisimulation. The variant of bisimulation used in this paper is *symbolic* bisimulation, which is based upon the definition of early symbolic bisimulation for value passing CCS introduced in [7].

Symbolic bisimulation is a complicated concept, particularly in the case of LOTOS where there are multiple ways in which the transitions of one STS may be matched by those of another. (For example, a transition which binds a new variable may be matched either by a corresponding variable-binding transition, or by a non variable-binding transition with an appropriate data offer. This is different from CCS where a `?` transition may be matched only with a `?` transition, and `!` only with `!`.) We shall only give a brief explanation of symbolic bisimulation here, but refer the reader to [4] for more detail.

Definition 3 gives the full definition of symbolic bisimulation on STSs. It takes as a parameter a boolean expression over the free variables of the two STSs being checked. This boolean expression represents the context, or condition under which two STS, T and U are bisimilar. The intuition behind the definition is that whenever one STS, say T, can perform a transition, leading to some state T', it is possible to find a finite set of boolean expressions, B which partition the context under which that transition can be performed. Then, for each partition within this set, the STS T must be able to perform an appropriate transition, leading to a state T' which is bisimilar to S' under the appropriate boolean condition.

Definition 3. *Layered symbolic bisimulation on STSs*
Given two STSs T and U and a boolean expression b,

1. $T \sim_0^b U$
2. *for all $k > 0$, $t \sim_k^b u$ provided that:*
   (a) **(simple event)**
   if T has a transition $T \xrightarrow{b_T \;\; a} T'$, then there is a finite set of booleans B over $fv(T)$ such that $b \wedge b_T \Rightarrow \bigvee B$ and for each $b' \in B$, there is a transition $U \xrightarrow{b_U \;\; a} U'$ such that $b' \Rightarrow b_U$ and $T' \sim_{k-1}^{b'} U'$
   (b) **(structured event, no new variable)**
   if T has a transition $T \xrightarrow{b_T \;\; gE_T} T'$ where $fv(E_T) \subseteq fv(T)$, then there is a finite set of booleans B over $fv(T) \cup \{z\}$ such that
   $b \wedge b_T \wedge z = E_t \Rightarrow \bigvee B$ (where z is a new variable),
   and for each $b' \in B$ either
   there is a transition $U \xrightarrow{b_U \;\; gE_U} U'$ where $fv(E_U) \subseteq fv(U)$ and
   $b' \Rightarrow b_U$ and $b' \Rightarrow E_T = E_U$ and $T' \sim_{k-1}^{b'} U'$
   or
   there is a transition $U \xrightarrow{b_U \;\; gy} U'$ such that $b' \Rightarrow b_U[z/y]$ and
   $T' \sim_{k-1}^{b'} U'[z/y]$

(c) **(structured event, new variable)**
whenever T has a transition $T \xrightarrow{b_T \; gx} T'$, then there is a finite set of booleans B over $fv(T) \cup \{z\}$ (where z is a new variable) such that $b \wedge b_T[z/x] \Rightarrow \bigvee B$, and for each $b' \in B$ either
there is a transition $U \xrightarrow{b_U \; gE_U} U'$ where $fv(E_U) \subseteq fv(U)$ and $b' \Rightarrow b_U$ and $b' \Rightarrow z = E_U$ and $T'[z/x] \sim_{k-1}^{b'} U'$
or there is a transition $U \xrightarrow{b_U \; gy} U'$ such that $b' \Rightarrow b_U[z/y]$ and $T'[z/x] \sim_{k-1}^{b'} U'[z/y]$
(d), (e), (f) Symmetrically, the transitions of U must be matched by T.

We have proved, partially on paper, and partially within PVS, that the logic FULL is adequate with respect to symbolic bisimulation on STSs (without loops). The PVS aspect of this proof is discussed in Section 6. In our later work [2, 3], in which we used "terms" rather than STSs, we found it more convenient to prove adequacy with respect to a less complex definition of bisimulation. This proof was carried out on paper, but was heavily influenced by the insights gained in doing the PVS assisted proof relating to symbolic bisimulation.

The PVS encoding of the definition of symbolic bisimulation is technically straightforward but lengthy. A fragment, showing just one (rule 2(b)) of the six rules that make up the definition, is shown in Figure 4.

The encoding uses several preliminary definition which were found convenient, and whose details are not shown. For example, given an STS `T`, the function `structInitTransitions` applied to `T` returns the set of all structured transitions (that is, those involving a data offer) available from the initial state of `T`. There are also some auxiliary functions, such as `gateOf` and `dataOf`, for extracting various components from a transition, and functions for carrying out syntactic substitution (for example, `subB`, `applySubstSTS_aux`.)

6 Proving Adequacy

To prove the adequacy of the logic with respect to symbolic bisimulation, we must show that the equivalence upon STSs induced by the logic corresponds to symbolic bisimulation (for closed STSs). There are two directions to be proved. First, we showed (Proposition 1) that if two closed STSs are bisimilar, then they satisfy exactly the same logical formulae. This proof was done partially within PVS. Second, we showed the converse (Proposition 3), namely, that if two closed STSs are not bisimilar, then they can be distinguished by some logical formula. This proof was done entirely on paper, and is not discussed here.

For both theorems, it was useful to define the *depth* of a formula. This is a simple, syntactic depth, derived from the grammar of Definition 2.

Proposition 1. *For all n, for all closed STSs T and U, if $T \sim_n^{true} U$ then, for all formulae Φ such that depth$(\Phi) \leq n$, $T \models \Phi$ if and only if $U \models \Phi$.*

Proposition 1 was proved by first proving a more general result about open STSs, Proposition 2.

```
bisim_n(b,T,U,n) : RECURSIVE bool =
  IF n = 0 THEN true  % at layer 0 all STSs are bisimilar
  ELSE
  % ...rules for dataless transitions omitted...
  % the structured transitions of T must be simulated by U:
    (FORALL (tT : (structInitTransitions(T))) :
       EXISTS (z:({x:Variable|NOT(member(x,union(vars(T),vars(U))))})) :
       EXISTS (Bs:list[(BoolExpOverVars(union(freevars(T),singleton(z))))]) :
         IF boundvars(T,tT) = emptyset  % T transition does not bind a variable
         THEN
            imply? (andalso(andalso(b,condition(tT)),
                       equalExp(variable(z),
                                dataOf(structuredEv(event(tT))))),
                    disjoin(Bs))
            AND
             (FORALL (b1:({x:BoolExp | member(x,Bs)})) :
              EXISTS (tU:(structInitTransitions(U))) :
                 (gateOf(structuredEv(event(tU))) =
                  gateOf(structuredEv(event(tU))))
                 AND
                 IF      % U transition does not bind a variable
                    boundvars(U,tU) = emptyset
                 THEN
                    imply? (b1,
                            andalso(condition(tU),
                               equalExp(dataOf(structuredEv(event(tT))),
                                        dataOf(structuredEv(event(tU))))))
                    AND
                       bisim_n(b1,post(T,tT),post(U,tU),n-1)
                 ELSE    % U transition binds new variable
                    imply?(b1,subB(boundvar(U,tU),variable(z),condition(tU)))
                    AND
                       bisim_n(b1,post(T,tT),
                               applySubstSTS_aux(boundvar(U,tU),z)
                                                (post(U,tU)),
                                                n-1)
                 ENDIF)
        ELSE  % T transition binds new variable - details omitted
        ENDIF)
  % ...symmetrical rules omitted...
  ENDIF

  MEASURE n
```

Fig. 4. PVS encoding of symbolic bisimulation

Proposition 2. *For all n, for all boolean expressions b, for all STSs T and U, if $T \sim_n^b U$ then, for all formulae Φ such that $depth(\Phi) \leq n$, for all substitutions σ such that $T\sigma$ and $U\sigma$ are both closed, if $b\sigma \equiv true$ then $T\sigma \models \Phi$ if and only if $U\sigma \models \Phi$.*

The proof of Proposition 2 was carried out almost entirely in PVS. The proof is basically by induction on n. The $n = 0$ case is straightforward. For the $n > 0$ case, there is a further induction on the structure of the formula Φ. As there are 9 constructors for formulae (Definition 2), 2 cases to consider in the semantics of those formulae that involve data, and a multitude of rules in the definition of bisimulation, the proof results in the generation of a large number of subgoals. PVS was invaluable for keeping track of all the subgoals and ensuring that none was missed.

The cases of formulae not involving data (ie, booleans, conjunction, disjunction, simple box and diamond) were relatively straightforward to prove. The other cases, concerning the new modalities for data, were more complex, involving close attention to details such as the free variable conditions on boolean expressions and STSs.

A recurring irritation in doing the proof in PVS was the need to prove trivial-seeming subgoals regarding the effect of substitution upon such properties as the depth of formulae. Many of these subgoals were postponed, and have not been formally proved at the date of writing, so it is for this reason that we say that the proof was only partially carried out with PVS. The mechanism for postponing subgoals is useful, but it can be confusing to the user when a postponed subgoal unexpectedly reappears after, say, a succession of other subgoals have quickly been proved. It can be difficult to keep track of one's place within the proof. We suggest that this problem might be alleviated by allowing postponed subgoals to be named and thereafter hidden from the user until explicitly asked for.

It has been suggested to us that many of the problems caused by the explicit handling of substitution might have been avoided by the use of higher-order syntax techniques. We have not yet explored this option.

For completeness, Propostion 3 the other half of the adequacy result. This has been proved entirely on paper. The proof of a similar result, relating to "terms" rather than STSs, is outlined in [2].

Proposition 3. *For all n, for all closed terms T and U, if $T \not\sim_n^{true} U$ then there is a formula Φ such that $T \models \Phi$ and $\not\models \Phi$*

7 Conclusion

Our main aim in using PVS for this work was to have computer assistance with keeping track of all the details in our definitions, especially at times when several versions of, for example, the semantics of the logic FULL, were under consideration. Another main aim was to have computer assistance with the adequacy proof, especially with keeping track of all the inductive cases and sub-cases that arose.

After carrying out the work, we judged that the first of these aims had been justified. Having multiple versions stored on the computer as PVS files, rather than written out on paper made it much easier to compare versions, perform edits, and generally keep track of detail. Our only complaint, regarding this aim, was that PVS would issue warnings when there were multiple versions of a theory with the same name.

Regarding the second aim, our verdict is somewhat mixed. PVS does indeed keep track of all the subgoals in a proof, ensuring that no subgoal is inadvertently missed out. However, there were so many subgoals generated in the PVS proof that it was easy to lose sight of the high level proof structure. We found that substantial effort was needed, requiring annotations on paper, in order to keep track of the thread of the proof.

The most surprising result of the exercise, however, was the great benefit we gained from the process of formalizing our definitions. By forcing us to be fully explicit about what we meant by, for example, substitution on STSs, the formalization process revealed to us some inherent problems in our assumptions, and led us towards a realization of the correct definitions. We believe that this was the greatest benefit derived from using PVS for this work.

References

1. T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. In P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors, *The formal Description Technique LOTOS*, Elsevier Science Publishers B.V. (North-Holland), 1989.
2. M. Calder, S. Maharaj and C. Shankland. An Adequate Logic for Full LOTOS. In Proceedings of Formal Method Europe 2001, Springer LNCS 2021, March 2001.
3. M. Calder, S. Maharaj and C. Shankland. A Modal Logic for Full LOTOS based on Symbolic Transition Systems. To appear in *The Computer Journal.*
4. M. Calder and C. Shankland. A Symbolic Semantics and Bisimulation for Full LOTOS. Technical Report CSM-159, Department of Computing Science and Mathematics, the University of Stirling. October 2000.
5. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.
6. M. Hennessy and X. Liu. A Modal Logic for Message Passing Processes. In *Acta Informatica* 32 (1995) 375-393.
7. M. Hennessy and H. Lin. Symbolic bisimulations. In *Theoretical Computer Science* 138 (1995) 353-389.
8. C.A.R. Hoare. *Communicating Sequential Processes*, Prentice-Hall International, 1985.
9. S. Owre, J.M. Rushby and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, Saratogo, NY, June 1992. Springer-Verlag LNCS no. 607.
10. R. Milner. *Communication and Concurrency*, Prentice-Hall (1989).