21

A method to build symbolic representations of LOTOS specifications

Riccardo Sisto
Politecnico di Torino
Dip. di Automatica e Informatica, Politecnico di Torino,
Corso Duca degli Abruzzi, 24, I-10129 Torino (Italy)
*phone: +39 11 5647073, fax: +39 11 5647099, e-mail:*sisto@polito.it

Abstract

A symbolic representation of a large state/transition system, based on Binary Decision Diagrams (BDD's), is generally much more compact than an explicit representation like a Labelled Transition System (LTS). This is due to regular and repetitive patterns occurring in state/transition systems. By exploiting this property, huge state spaces can be represented symbolically, and the resulting BDD representations can be profitably used for activities such as model checking and sequential circuit synthesis. This paper presents a method to build BDD representations from process algebraic specifications, taking LOTOS as a reference. The method exploits the compositionality of process algebras to avoid the enumeration of all the states and transitions. First, small labelled transition systems are created for representing the basic building blocks of the specification. These are converted to BDDs, which in turn are combined together, according to the various process algebraic operators, to obtain the overall BDD. An example is used throughout the paper to illustrate the method.

Keywords

Protocol Engineering, LOTOS, Symbolic Model Checking, Binary Decision Diagrams.

1 INTRODUCTION

It is commonplace in protocol engineering to formally represent protocols by means of state/transition models such as state machines and Labelled Transition Systems (LTS's). One of the main problems with these representations is state explosion, which generally limits their applicability to oversimplified versions of the real-life protocols.

Several techniques have been devised to cope with the state explosion problem. Some of them are based on abstractions or parametrizations of specifications (Cousot, 1977). Others start from the idea that state explosion mainly comes from the interleaving of actions performed by parallel components: in order to reduce the problem, the explicit

P. Dembiński et al. (eds.), *Protocol Specification, Testing and Verification XV*

representation or exploration of all the possible event interleavings is avoided (Quemada, 1992, Valmari, 1990) or true concurrency semantics models such as event structures are adopted (Winksel, 1989). In the area of formal verification by model checking, specific techniques such as on-the-fly techniques, partial techniques, or reduction techniques have been proposed to simplify the exploration of the state space (Fernandez et al., 1992, Holzmann, 1994, West, 1986).

Another class of techniques for reducing the state explosion problem is based on the symbolic representation of state transition relations. One of such techniques, Binary Decision Diagrams (BDD's), has already been applied to hardware verification and synthesis with surprisingly successful results. Sequential circuits with up to 10^{120} states have been formally verified, and the CPU time required for verification increases as a small polynomial in the number of the circuit components (Burch et al., 1994). These results suggest that similar techniques could be profitably used in the context of protocol engineering as well.

In this paper, the possibility of employing symbolic techniques in the design of protocols is investigated, considering specifically the domain of process algebraic specifications, and LOTOS (ISO, 1989) in particular. Process algebraic specifications have many advantages, among which compositionality and conciseness. In principle these features make it possible to avoid the explicit representation of the state transition relation, but, as a matter of fact, many design activities taking process algebraic specifications as inputs are carried out by first generating the corresponding Labelled Transition System or a variant of it (Bolognesi and Caneve, 1989, Garavel and Najm, 1989).

The main objective of this paper is to show that the technique of BDDs can be adopted to get alternative models out of process algebraic specifications. These models are likely to be tractable even when the usual LTS-like models have unmanageable sizes. Moreover, BDD-based model checking and hardware synthesis are efficient well-established techniques.

The problem is that, while in sequential circuit design techniques are also available for automatically deriving BDD representations from hardware specifications given in languages such as VHDL, or given directly as logic gate networks (Burch et al., 1994), similar techniques are not available if the source specification formalism to be used is a process algebra. A trivial solution to this problem would be to build the LTS of the source process algebraic specification, and then convert the LTS to a BDD. However, this kind of solution, which has already been adopted to get BDD representations from ATP specifications (Nicollin et al., 1992) is not very satisfactory, in that the usefulness of BDD representations stands mainly in the ability to manage specifications for which the LTS is too big to be represented or even traversed.

In this paper, a more satisfactory solution is presented, which exploits compositionality of process algebras, so as to build BDD representations without enumerating all the states and transitions. The technique which is proposed consists in building small labelled transition systems for representing the basic building blocks of the specification. These are converted to BDDs which in turn are combined together, according to the various process algebraic operators, to obtain the overall BDD.

The paper is organized as follows. In section 2, BDD concepts are recalled, and in section 3 the use of BDDs to represent LTSs is discussed and some notation is introduced. Section 4 describes the mapping from specifications written in LOTOS to BDD representations,

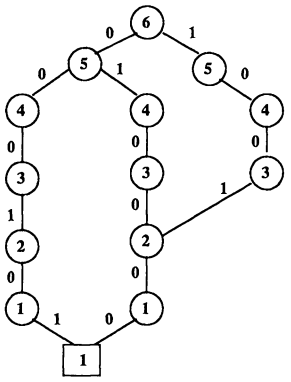

Figure 1 A sample BDD

whereas in section 5 the usefulness and applicability of the proposed method is discussed. Finally, section 6 contains conclusive remarks and directions for further research.

2 BINARY DECISION DIAGRAMS

Ordered Binary Decision Diagrams (BDD's) are a canonical form representation for boolean functions which is very compact and can be manipulated very efficiently (Bryant, 1986). A BDD is a directed acyclic graph with a root and two leaves. The leaves represent the boolean values 0 and 1, whereas the other nodes are each labelled with one of the boolean variables of the function. Each node has two outgoing edges, labelled with the boolean values 0 and 1, representing the possible values of the variable which labels the node. Variables are strictly totally ordered, and they occur in nodes according to such ordering along any path starting from a leaf and ending in the root. Figure 1 shows a sample BDD where nodes labelled by variables are represented as circles with a number identifying the corresponding variable, whereas leaf nodes are represented as squares. Note that, in order to make the graph more readable, the leaf node labelled 0 and the edges ending in it are not represented explicitly: if a node has only one outgoing edge, the other edge is assumed to be directed to the missing leaf node.

Given an assignment of the function boolean variables, the corresponding value of the function can be determined by traversing the BDD starting at the root and branching at each node according to the value assigned to the variable that labels the node. The value of the function is given by the label associated with the leaf that is reached. For example, if the assignment is $\{6,5,4,3,2,1\} = 000101$, the value taken by the function represented in Figure 1 is 1, whereas for $\{6,5,4,3,2,1\} = 111111$ it is 0.

For any boolean function and ordering of variables, there exists a unique canonical minimal BDD representation, which makes it possible to test for function equality very efficiently. A BDD is canonical if no node has both outgoing edges inciding on the same node, and no distinct nodes V and V' are such that the subgraphs rooted by V and V' are isomorphic. Efficient algorithms for combining functions according to the main boolean

algebraic operators and for restricting or composing functions represented as canonical BDD's are also available (Bryant, 1986).

Let V be an ordered set of n boolean variables. The notation $f(V)$ indicates a boolean function defined over V. If K is an assignment of V, i.e. K is an n-uple of boolean values, the notation $V = K$ denotes the boolean function on V that takes value 1 iff the assignment of V is K. For example, if $V = \{v1, v2\}$, and $K = (0, 1)$, $V = K$ equals $\neg v1 \wedge v2$. If $V1 \subseteq V$, and $g(V1)$ is a function on $V1$, $f(V)|_{g(V1)}$ denotes function f restricted on $g(V1)$. For example, if $f(V) = v1 \wedge v2$, $V1 = \{v1\}$, and $g(V1) = v1$, we have $f(V)|_{g(V1)} = v2$.

3 BDD REPRESENTATION OF LTS'S

A Labelled Transition System (LTS) is a quadruple $< S, s0, \Sigma, R >$ where:

- S is a countable set of states;
- $s0 \in S$ is the initial state;
- Σ is a countable set of events;
- $R \subseteq S \times \Sigma \times S$ is the transition relation.

A variant of this definition is an LTS with *multiple* initial states, which is a quadruple $< S, S0, \Sigma, R >$, where S, Σ, and R are defined as before, but $S0 \subseteq S$ is *a set of* initial states.

In a finite LTS, states can be represented by means of a finite set of boolean state variables, each state being represented by one of the possible assignments of these variables. Similarly, events can be represented by means of a finite set of boolean event variables.

(Burch et al., 1994) suggests a method for representing sets of states and unlabelled transition relations (i.e. relations defined on $S \times S$) by means of BDD's when states are represented by means of boolean state variables. A single state s can be represented as the boolean function which takes value 1 only for the state variable assignment representing s. If V is the set of state variables, this function is denoted $s(V)$. A set of states U can be represented as the boolean function taking value 1 for all and only the variable assignments representing the states in U. This function is denoted $U(V)$. Finally, an unlabelled transition relation $T \subseteq S \times S$ is a set of ordered state pairs. It can be represented as a boolean function defined over two sets of state variables: the present state variables V and the next state variables V'. This function, which is indicated by $T(V, V')$, takes value 1 if and only if the assignments of V and V' represent respectively two states s, s' such that T holds for (s, s') (i.e. $(s, s') \in T$).

In order to represent a labelled transition relation $R \subseteq S \times \Sigma \times S$ as a BDD, we can use a boolean function denoted $R(V, E, V')$, where E is the set of boolean event variables. $R(V, E, V')$ takes value 1 if and only if the assignments of V, E and V' represent respectively a state s, an event e, and a next state s' such that $(s, e, s') \in R$.

As a conclusion, a finite LTS with multiple initial states $< S, S0, \Sigma, R >$ can be represented by means of the quadruple $< V, V0, E, R >$, where: V is the set of boolean state variables, $V0 = \{s_{0i}(V)\}$ is the set of BDD's representing the initial states, E is the set of boolean event variables, and $R(V, E, V')$ is the BDD representing R.

4 THE ALGORITHM TO BUILD BDD REPRESENTATIONS

In this section, the specification language LOTOS (ISO, 1989) is taken as the reference specification formalism, even though the method can be easily adapted to other process algebras. Moreover, to simplify the treatment of the problem, only basic LOTOS is considered.

Without loss of generality, it is assumed that the source LOTOS specification is already flattened, i.e. it is made up of a flat set of process definitions (no nested definitions), and gates and processes are assigned unique identifiers. Any LOTOS specification satisfying the static semantics requirements can be flattened, according to IS8807 (ISO, 1989).

Moreover, to simplify the description of the method, both the operands of parallel, enabling, and disabling operators are assumed to be process instantiations. This assumption is indeed not restrictive, since any specification can be easily and automatically modified so as to fulfill it, by eventually adding more process definitions.

Since the aim here is to obtain finite sized models, it is necessary also to exclude LOTOS specifications with an unbounded number of states or transitions. The feature of the language that has to be eliminated so as to ensure finiteness is unbounded (recursive) creation of processes and gates, which can be eliminated by requirig that

- Recursions are guardedly well defined;
- Any expression taking the form $A|[G]|B$ is such that all the definitions of processes instantiated directly or indirectly by A and B do not contain $A|[G]|B$ as a sub-expression.
- Any expression E taking the form $A[> B$ or $A >> B$ is such that all the definitions of processes instantiated directly or indirectly by A do not contain E as a sub-expression.

In the description of the method, the basic LOTOS specification of Figure 2 will be used as an illustrative example. It is the specification of the transport protocol handler presented in (ISO, 1989, annex C), where processes `Connected`, and `Data_Term_Phase` have been added so as to fulfill the requirements stated above.

4.1 Preliminary definitions

The BDD representation of a LOTOS behavior expression B is actually the representation of the corresponding LTS $< V_B, I^B, E_B, R^B >$, where V_B is the set of boolean state variables, $I^B(V_B)$ is the BDD representing the initial state, E_B is the set of boolean event variables, and $R^B(V_B, E_B, V_B')$ is the BDD representing the transition relation.

V_B is assumed to be a set of ordered elements $V_B = \{v_1, .., v_n\}$. This makes it possible to identify state variables by means of an integer index. The number of elements in V_B is indicated by $size(V_B)$ (where $n = size(V_B)$) and depends on the number of states of B.

Let G_B be the set of gates occurring in B, and $Act_B = G_B \cup \{i, \delta\}$ the set of events. E_B is such that, for each event $e \in Act_B$ occurring in B, there exists a unique assignment corresponding to e. The notation $e(E_B)$ indicates a boolean function which takes value 1 iff the assignment of the variables in E_B corresponds to e. Similarly, if H is a set of events, $H(E_B) = 1 \Leftrightarrow \exists e \in H | e(E_B) = 1$, i.e. if the assignment of variables E_B corresponds to an element of H. The BDD's of these functions can be determined directly when the encoding of gates has been decided.

In the following, for simplicity, all the behavior expressions B defined within a LOTOS

```
specification transp[CReq,CInd,CRes,CCnf,DatReq,DatInd,DisReq,DisInd] :noexit
behaviour

Handler[CReq,CInd,CRes,CCnf,DatReq,DatInd,DisReq,DisInd]

where

process Handler[CReq,CInd,CRes,CCnf,DatReq,DatInd,DisReq,DisInd]:noexit :=
Conn_Phase[CReq,CInd,CRes,CCnf,DisReq,DisInd]
>>      Connected [CReq,CInd,CRes,CCnf,DatReq,DatInd,DisReq,DisInd]
endproc

process Connected[CReq,CInd,CRes,CCnf,DatReq,DatInd,DisReq,DisInd]:noexit :=
Data_Term_Phase[DatReq,DatInd,DisReq,DisInd]
>>      Handler[CReq,CInd,CRes,CCnf,DatReq,DatInd,DisReq,DisInd]
endproc

process Data_Term_Phase[DatReq,DatInd,DisReq,DisInd]:noexit :=
Data_Phase[DatReq,DatInd] [>   Term_Phase[DisReq,DisInd]
endproc

process Conn_Phase[CReq,CInd,CRes,CCnf,DisReq,DisInd]:exit :=
   i; Calling[CReq,CInd,CRes,CCnf,DisReq,DisInd]
[]    Called[CReq,CInd,CRes,CCnf,DisReq,DisInd]
endproc

process Calling[CReq,CInd,CRes,CCnf,DisReq,DisInd]:exit :=
CReq; (CCnf; exit [] DisInd; Conn_Phase[CReq,CInd,CRes,CCnf,DisReq,DisInd])
endproc

process Called[CReq,CInd,CRes,CCnf,DisReq,DisInd]:exit :=
CInd;
(i; CRes; exit [] i; DisReq; Conn_Phase[CReq,CInd,CRes,CCnf,DisReq,DisInd])
endproc

process Data_Phase[DatReq,DatInd]:noexit :=
(i; DatReq; Data_Phase[DatReq,DatInd]) [] (DatInd; Data_Phase[DatReq,DatInd])
endproc

process Term_Phase[DisReq,DisInd]:exit :=
(i; DisReq; exit) [] (DisInd; exit)
endproc

endspec
```

Figure 2 The specification of the transport protocol handler

specification are represented using the same set of event variables $E_B = E$, where E is such that for any event occurring in the whole specification there exists a unique assignment.

The BDD representation of a LOTOS behavior expression B can be determined directly if the representations of its sub-expressions are already known. Let $B1$ and $B2$ be any behavior expressions for which a BDD representation is known, and let $n1 = size(V_{B1})$, $n2 = size(V_{B2})$. The representation of B can be computed as follows.

- If $B = hide\ H\ in\ B1$, where H is a set of gates, the representation of B can be computed from the representation of $B1$ by simply replacing events at gates in H by internal events:

$$\begin{aligned} V_B &= V_{B1} \\ I^B(V_B) &= I^{B1}(V_B) \\ R^B(V_B, E, V_B') &= (R^{B1}(V_B, E, V_B') \wedge \neg H(E)) \vee (R^{B1}(V_B, X, V_B')|_{H(X)} \wedge i(E)) \end{aligned}$$

- If $B = B1|[G]|B2$, where G is a set of gates, the state of B can be represented as an ordered pair $(s1, s2)$, where $s1$ is the state of $B1$, and $s2$ is the state of $B2$. Hence, the set of state variables V_B can be considered as being divided into two parts: $V_1 = \{v_1, .., v_{n1}\}$ is the subset of variables representing $s1$, and $V_2 = \{v_{n1+1}, .., v_{n1+n2}\}$ is the subset of variables representing $s2$. The representation of B is given by:

$$\begin{aligned} V_B &= \{v_1, .., v_{n1+n2}\} = V_1 \cup V_2 \\ I^B(V_B) &= I^{B1}(V_1) \wedge I^{B2}(V_2) \\ R^B(V_B, E, V_B') &= (F_{11} \vee F_{22} \vee F) \qquad where \end{aligned}$$

$$\begin{aligned} F_{11} &= R^{B1}(V_1, E, V_1') \wedge (V_2 \Leftrightarrow V_2') \wedge \neg(G(E) \vee \delta(E)) \\ F_{22} &= R^{B2}(V_2, E, V_2') \wedge (V_1 \Leftrightarrow V_1') \wedge \neg(G(E) \vee \delta(E)) \\ F &= R^{B1}(V_1, E, V_1') \wedge R^{B2}(V_2, E, V_2') \wedge (G(E) \vee \delta(E)) \end{aligned}$$

F_{11} and F_{22} represent transitions involving $B1$ or $B2$ only, while F represents interactions of $B1$ and $B2$.

- If $B = B1 >> B2$, any state of B is either a state of $B1$ or a state of $B2$. Hence, the set V_B can be considered as being composed of a variable s, which represents whether the current state is a state of $B1$ ($s = 0$) or a state of $B2$ ($s = 1$), plus a set of variables used to represent alternatively the state of $B1$ or the state of $B2$, according to the value of s. Let $V_B = \{v_1, .., v_l, v_{l+1}\}$, where $l = max(n1, n2)$, and the selector variable is $s = v_{l+1}$. With this choice, we have $V_{B1} \subseteq V_B$, and $V_{B2} \subseteq V_B$. The remaining items of the representation of B are given by:

$$\begin{aligned} I^B(V_B) &= \neg v_{l+1} \wedge I^{B1}(V_{B1}) \\ R^B(V_B, E, V_B') &= F_{11} \vee F_{22} \vee F_{12} \qquad where \end{aligned}$$

$$\begin{aligned} F_{11} &= R^{B1}(V_{B1}, E, V_{B1}') \wedge \neg v_{l+1} \wedge (v_{l+1}' \Leftrightarrow v_{l+1}) \wedge \neg\delta(E) \\ F_{22} &= R^{B2}(V_{B2}, E, V_{B2}') \wedge v_{l+1} \wedge (v_{l+1}' \Leftrightarrow v_{l+1}) \end{aligned}$$

$$F_{12} = \exists X[R^{B1}(V_{B1}, Y, X)|_{\delta(Y)}] \wedge I^{B2}(V'_{B2}) \wedge \neg v_{l+1} \wedge v'_{l+1}$$

F_{11} and F_{22} represent transitions of $B1$ and $B2$ respectively, whereas F_{12} represents transitions corresponding to enabling events.

- If $B = B1[> B2$, the set of state variables is the same as for the enabling operator. The representation of B is given by:

$$\begin{aligned} V_B &= \{v_1, .., v_l, v_{l+1}\}, \text{ where } l = max(n1, n2) \\ I^B(V_B) &= \neg v_{l+1} \wedge I^{B1}(V_{B1}) \\ R^B(V_B, E, V'_B) &= F_{11} \vee F_{22} \vee F_{12} \qquad where \end{aligned}$$

$$\begin{aligned} F_{11} &= R^{B1}(V_{B1}, E, V'_{B1} \wedge \neg v_{l+1} \wedge (v_{l+1} \Leftrightarrow v'_{l+1})) \\ F_{22} &= R^{B2}(V_{B2}, E, V'_{B2} \wedge v_{l+1} \wedge (v_{l+1} \Leftrightarrow v'_{l+1})) \\ F_{12} &= R^{B2}(X, E, V'_{B2})|_{I^{B2}(X)} \wedge \neg v_{l+1} \wedge v'_{l+1} \end{aligned}$$

- If $B = B1[H/G]$, where H and G are two lists of gates with the same length, the representation of B can be computed from the representation of $B1$ by relabelling gates in G by means of the corresponding gates in H. Let us indicate by $[H/G](E1, E2)$ the function that takes value 1 iff $E1$ represents the relabelled version of the event represented by $E2$. The representation of B is given by:

$$\begin{aligned} V_B &= V_{B1} \\ I^B(V_B) &= I^{B1}(V_B) \\ R^B(V_B, E, V'_B) &= (R^{B1}(V_B, X, V'_B) \wedge G(X))|_{[H/G](E,X)} \end{aligned}$$

The choice, action prefix and nullary operators have not been considered here, because they are dealt with in a different way later in the paper.

The composition rules described above are useful, but they cannot be used directly, in the presence of recursion. In the following sections, a systematic approach to build a BDD representation of a LOTOS specification is described.

4.2 Partitioning the set of process definitions

Let $P0, .., Pn$ be the process definitions of a given specification S, where $P0$ defines the behavior of the whole specification. The first step in the construction of a BDD for S is to define a partition of the set of process definitions $P = \{P0, P1, .., Pn\}$.

Let us indicate by $Beh(Pi)$ the behavior expression associated with process definition Pi. In the following, the shorthand *process Pi* will be used instead of *the process whose definition is Pi* whenever it is non-ambiguous.

Let us define the *instantiation relation* $\rightarrow \subseteq P \times P$ in the following way: $Pi \rightarrow Pj$ (Pi instantiates Pj) if and only if $Beh(Pi)$ contains an instantiation of process Pj.

The transitive closure of $\rightarrow$ is the *indirect instantiation relation* $\stackrel{+}{\rightarrow}$. In practice, $Pi \stackrel{+}{\rightarrow} Pj$ iff Pi instantiates Pj directly or indirectly.

The transitive reflexive closure of $\rightarrow$ will be indicated by $\stackrel{*}{\rightarrow}$, and is defined as:

$$Pi \stackrel{*}{\rightarrow} Pj \Leftrightarrow (Pi = Pj) \vee (Pi \stackrel{+}{\rightarrow} Pj).$$

Finally, the symmetric closure of $\stackrel{*}{\rightarrow}$ is the *mutual instantiation relation* $\stackrel{*}{\leftrightarrow}$:

$$Pi \stackrel{*}{\leftrightarrow} Pj \Leftrightarrow (Pi \stackrel{*}{\rightarrow} Pj) \wedge (Pj \stackrel{*}{\rightarrow} Pi).$$

In practice, $Pi \stackrel{*}{\leftrightarrow} Pj$ means that Pi and Pj are either the same process definition or they define processes which can instantiate each other, i.e. mutually recursive processes.

The mutual instantiation relation is an equivalence which defines a partition on P. Each class $C \in P/\stackrel{*}{\leftrightarrow}$ is a set of mutually recursive process definitions, i.e. each process in C may instantiate directly or indirectly any other process in C. Let us indicate by $[Pi]$ the class to which Pi belongs. For example, in the specification in Figure 2, the classes are:

```
Class 0 : {Conn_Phase, Calling, Called}
Class 1 : {Term_Phase}
Class 2 : {Data_Phase}
Class 3 : {Data_Term_Phase}
Class 4 : {Handler, Connected}
Class 5 : {transp}
```

A partial ordering on $P/\stackrel{*}{\leftrightarrow}$ can now be defined based on the $\stackrel{*}{\rightarrow}$ relation:

$$[Pi] \leq [Pj] \Leftrightarrow \exists (Ph \in [Pi], Pk \in [Pj]) \mid Pk \stackrel{*}{\rightarrow} Ph.$$

$$[Pi] < [Pj] \Leftrightarrow ([Pi] \leq [Pj]) \wedge ([Pi] \neq [Pj]).$$

This partial order has the following meaning: $[Pi] < [Pj]$ implies that: processes in $[Pj]$ can instantiate directly or indirectly processes in $[Pi]$, but the converse is false. In other words, process definitions in $[Pi]$ are needed to determine the behavior of processes in $[Pj]$, whereas the behavior of processes in $[Pi]$ can be determined without knowledge of process definitions in $[Pj]$. For example, in the specification of Figure 2, [Data_Phase] < [Data_Term_Phase].

Any LOTOS specification is such that

1. $\nexists Pk \in P \mid Pk \stackrel{+}{\rightarrow} P0$
2. $\forall Pk \in P \quad P0 \stackrel{*}{\rightarrow} Pk$

The first statement means that $P0$ (the specification) cannot be instantiated. The second means that all the process definitions are reachable from the process defined by $P0$. These properties imply that $[P0]$ contains only $P0$, and $[Pk] \leq [P0] \quad \forall Pk \in P$. As a consequence, $[P0]$ is the greatest element of $P/\stackrel{*}{\leftrightarrow}$.

The construction of the BDD representation of a LOTOS specification is accomplished by building a BDD representation for each one of the classes of process definitions. The partial order defined above determines the order in which classes must be processed: if

Table 1 Expandibility definition

B		$is\_fexp_C(B)$	$is\_pexp_C(B)$
g;B'		true	true
stop		true	true
exit		true	true
B1|[G]|B2		false	false
B1 >> B2		false	false
B1 [> B2		false	false
P[G]	$P \in C$	$is\_texp_C(Beh(P[G]))$	$is\_pexp_C(Beh(P[G]))$
P[G]	$P \notin C$	false	false
B1[]B2		$is\_texp_C(B1) \wedge is\_texp_C(B2)$	$is\_pexp_C(B1) \vee is\_pexp_C(B2)$
hide G in B1		$is\_texp_C(B1)$	$is\_pexp_C(B1)$

$[Pi] < [Pj]$, $[Pj]$ will be processed after $[Pi]$. This ensures that the BDD representations of the classes containing process definitions referenced in $[Pj]$ have already been computed when $[Pj]$ is processed. The processing of $[P0]$ comes last ($[P0]$ is the greatest element) and yields the BDD of the whole specification. A possible evaluation order for the classes of the specification in Figure 2 is the one in which they have been enumerated above.

4.3 The BDD representation of a class

The behavior of the processes composing a class can be represented by means of an LTS for each member of the class. Since the process definitions composing a class are mutually recursive, the LTS of a given process definition is likely to contain many states which are observationally equivalent to corresponding states in the LTS's of the other process definitions in the same class. In most cases, the LTS's of the process definitions that constitute a class are even characterized by isomorphic state sets, isomorphic event sets, and isomorphic transition relations (this applies, for example, to the LTS's of processes belonging to classes 0 and 4 in the sample specification). The only item which really distinguishes them is the initial state. For this reason, it is advantageous to merge the LTS's of the process definitions constituting a class into a single LTS with multiple initial states, one initial state for each process definition. Therefore, the BDD representation of a class C is defined as $< V_C, I^C, E, R^C >$, where $V_C = \{v_1, .., v_n\}$, $I^C = \{I^{Beh(Pi)}(V_C) | Pi \in C\}$, E is the global representation of events occurring in the specification, and R^C is the BDD representation of the state transition relation.

4.4 Building the BDD representation of a class

Let us give some preliminary definitions.

Given a behavior expression B and a class of process definitions C, B is said *fully expansible in C* if predicate $is\_fexp_C(B)$, defined in Table 1, is true. B is said *partially expansible in C* if predicate $is\_pexp_C(B)$, defined in Table 1, is true. Finally, B is said *non-expansible in C* if $is\_pexp_C(B)$ is false. Note that $is\_fexp_C(B) \Rightarrow is\_pexp_C(B)$.

Table 2 contains a set of equalities modulo strong bisimulation congruence, that will be applied in the following. In Table 2, $P[G]$ denotes a process instantiation referencing

Table 2 Strong bisimulation congruence preserving transformations

(1)	$P[G]$	=	$Beh(P[G])$	
(2)	hide G in $B1[]B2$	=	(hide G in $B1$)[](hide G in $B2$)	
(3)	hide G in $g;B1$	=	g; hide G in $B1$	if $g \notin G$
(4)	hide G in $g;B1$	=	i; hide G in $B1$	if $g \in G$

process definition P, where G is the list of actual gates, whereas $Beh(P[G])$ denotes $Beh(P)$ with formal gates of P replaced by actual gates G.

The Partially Expanded Labelled Transition System (PELTS) of a class C is a pruned version of the LTS with multiple initial states which describes the behavior of the class members. It is defined as the result of the following construction algorithm:

1. For each process definition $Pi \in C$ such that $Pi = P0$ or $Pj \rightarrow Pi$ for some $Pj \notin C$, an initial state si is created, and si is labelled by the behavior expression $Beh(Pi)$.
2. For each state s, created in the previous step of the algorithm:

   - If the behavior expression B which labels s is at least partially expansible in C, the following expansion step is carried out:
     By applying the strong bisimulation congruence preserving transformations in Table 2, B is put in the form $B = []\{Bk\}$, where each Bk takes one of the following forms:

     (a) $gk; Bk'$
     (b) exit
     (c) A non expansible behavior expression

   - For each Bk taking form a), a new state sk labelled Bk' is created, if not yet present, and a transition (s, gk, sk) is added to the transition relation. Similarly, for each Bk taking form b), a new state sk labelled stop is created, if not yet present, and a transition (s, δ, sk) is added to the transition relation.

3. Step 2 is repeated until no new states are created.

As an example, let us consider the specification in Figure 2. The PELTS of classes 0, 1, and 2 are depicted in Figure 3. They are characterized by fully expanded states only. Note that a state corresponding to process `Called` is not present in the PELTS of class `[Conn_Phase]`, because it is not instantiated by processes belonging to other classes. The PELTS's of the other classes contain only unexpansible states. For example, the PELTS of class 3 contains a single state corresponding to process `Data_Term_Phase`.

Once the PELTS has been built, it is straightforward to represent it by means of BDD's. Let the resulting representation be $< V_0, I^0, E, R^0 >$, where $I^0 = \{I^{Pi} | Pi \in C\}$.

For example, in Figure 3, the binary strings in the states represent a possible state encoding. The BDD in Figure 1 is a possible representation for the boolean function R^0 of the PELTS of class [`Data_Phase`], where $E = \{1,2,3,4\}$, $V_0 = \{6\}$, $V_0' = \{5\}$, $i = 0000$, `DatInd` $= 1010$, and `DatReq` $= 0010$.

If the PELTS of a class contains only fully expanded states, it is the full representation

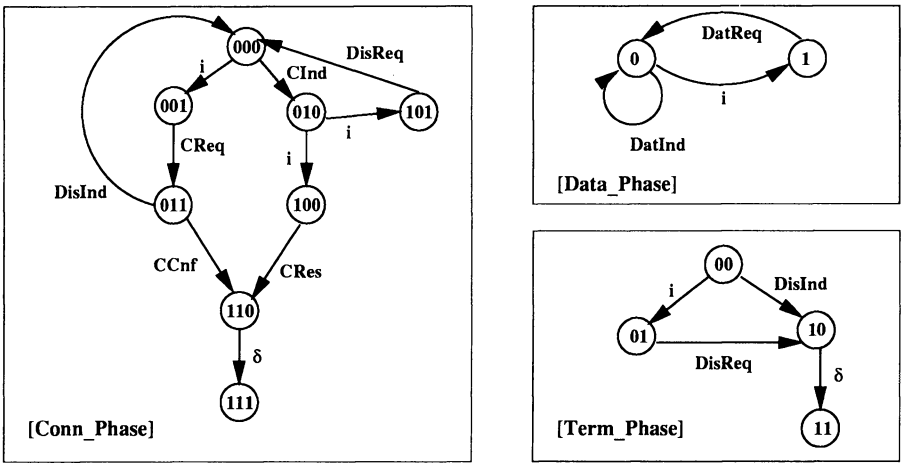

Figure 3 Graphical representation of the PELTS's for classes 0, 1, 2 of the sample specification

of the corresponding class. Instead, if it contains at least one non fully expansible state, it represents the behavior of the class members only partially, and a complete model must be computed by integrating the missing behavior components into the model.

Let us consider the states si labelled by non fully expanded expressions Bi. In general, Bi can be put in the form $Bi = Ei[]([]Bij)$, where Ei is the fully expansible component of Bi (if any), and Bij are the non-expansible components of Bi. From Table 1, and from the conditions imposed to obtain finiteness, each Bij must take one of the following forms:

$$\text{hide } H \text{ in } Ql \tag{1}$$
$$\text{hide } H \text{ in } (Ql|[G]|Qr) \tag{2}$$
$$\text{hide } H \text{ in } (Ql >> Pr) \tag{3}$$
$$\text{hide } H \text{ in } (Ql[> Pr) \tag{4}$$

where H is an eventually empty set of gates, Ql and Qr are instantiations of processes belonging to other classes, and Pr may be any process instantiation.

For each Bij taking one of the forms (1) or (2), a BDD representation $< V_{ij}, I^{ij}, E, R_{ij} >$ can be built by applying the composition rules given in section 4.1. In fact, the BDD representations of Ql and Qr are already known, because they are references to process definitions of already processed classes. The same applies if Bij takes one of the forms (3) or (4), and Pr is a reference to a process definition of another class. This is the case, for example, with class [Data_Term_Phase]: the PELTS of this class contains a single state labelled with an enabling expression referencing other classes. The representation of this expression can be computed directly: the resulting transition relation BDD is represented in Figure 4, along with the BDD representing the transition relation of class [Term_Phase].

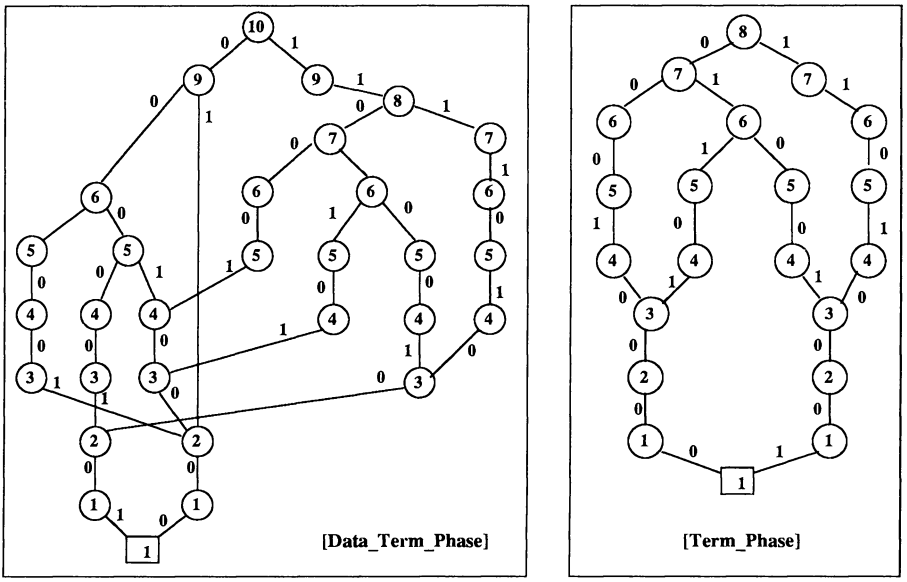

Figure 4 The BDD's of two transition relations

The encoding of events is: δ = 1000, DisReq = 0001, and DisInd = 1001, whereas the encoding of states is: $V = \{8,6\}$, $V' = \{7,5\}$ for class [Term_Phase], and $V = \{8,6,10\}$, $V' = \{7,5,9\}$ for class [Data_Term_Phase].

Instead, the case of expressions Bij taking one of the forms (3) or (4), with Pr referencing a process definition in class C, will be treated later on.

Class [Data_Term_Phase] represents a special case, because its BDD representation can be computed directly by combining representations of other classes. In general, the PELTS of a class may contain more Bij expressions to be integrated in the model. The behaviors defined by expressions Bij are all mutually exclusive, in that they define alternative behaviors which may be taken by the specification. Initially, the behavior of a process Pi defined in class C is represented by the state of the PELTS of C defined by $I^{Pi}(V_0)$. As soon as one of the expressions Bij is activated, the state is represented by an assignment of variables V_{ij}. For this reason, the global state s of a process behaving as specified by a class C can be thought of as made up of two components: $s = (z, w)$, where z represents which of the behavior expressions Bij is active (if any), whereas w encodes the state within the state space of the relevant Bij. More precisely, z can take a value for each Bij (denoted by zij), plus a value (denoted by $z0$) to represent the condition that none of the $Bij's$ is active.

The state variables of a class C are therefore $V_C = Z_C \cup W_C$, where Z_C is the set of variables encoding z, while W_C is the set of variables encoding w.

We take, as usual, $V_C = \{v_1, ..., v_l\}$, and we define $W_C = \{v_1, ..., v_h\}$, with $h = max(size(V_0), \max_{ij}(size(V_{ij})))$, and $Z_C = \{v_{h+1}, ..., v_{h+k}\}$ with $k = \lceil log_2(N) \rceil$, where N is the number of Bij expressions in the PELTS of C.

The initial state of a member Pi of class C is given by $I^{Pi}(V_C) = (Z_C = z0) \wedge I^{Pi}(V_0)$ while the BDD representation of the transition relation is

$$R_C(V_C, E, V'_C) = F_{00} \vee (\bigvee_{ij} F_{11}(ij)) \vee (\bigvee_{ij} F_{01}(ij)) \qquad where$$

$$\begin{aligned} F_{00} &= z0(Z_C) \wedge (Z_C \Leftrightarrow Z'_C) \wedge R_0(V_0, E, V'_0) \\ F_{11}(ij) &= zij(Z_C) \wedge (Z_C \Leftrightarrow Z'_C) \wedge R_{ij}(V_{ij}, E, V'_{ij}) \\ F_{01}(ij) &= z0(Z_C) \wedge zij(Z'_C) \wedge I^{Bi}(V_0) \wedge R_{ij}(X, E, V'_{ij})|_{I^{ij}(X)} \end{aligned}$$

F_{00} represents transitions starting from and ending to states of the PELTS. $F_{11}(ij)$ represents transitions starting from and ending to states belonging to the representation of expression Bij, while $F_{01}(ij)$ represents transitions starting from states of the PELTS and ending in states belonging to the representation of expression Bij.

The case of expressions Bij taking one of the forms (3) or (4), with Pr referencing a process definition of class C, is now considered. In this case, the BDD representation of Ql is known, because, owing to the assumptions made, it necessarily represents a process definition of another class, while the behavior of Pr is represented by one of the initial states of the PELTS of C. Let us now consider explicitly the case of enabling operators. Transitions from states of Pr to states of Pr are already represented by the F_{00} component of R_C. Transitions from states of Ql to states of Ql instead can be represented by means of $F_{11}(ij) \vee F_{01}(ij) \wedge \neg\delta(E)$, if we indicate by $< V_{ij}, I^{ij}, E, R^{ij} >$ the BDD representation of Ql (successful termination events are excluded since they correspond to enabling). Finally, transitions corresponding to enabling events must be represented by means of an additional component $F_{10}(ij)$. If $Bij = Ql_{ij} >> Pr_{ij}$, we have:

$$F_{10}(ij) = z0(Z'_C) \wedge I^{Pr_{ij}}(V'_0) \wedge i(E) \wedge \delta(ij) \qquad where$$

$$\delta(ij) = (zij(Z_C) \wedge (\exists X[R^{ij}(V_{ij}, \Gamma, X)|_{\delta(\Gamma)}])) \vee (z0(Z_C) \wedge (\exists X[R^{ij}(Y, \Gamma, X)|_{\delta(\Gamma) \wedge I^{ij}(Y)}])$$

$F_{10}(ij)$ represents transitions ending in the initial state of Pr_{ij} $(z0(Z'_C) \wedge I^{Pr_{ij}}(V'_0))$, labelled with the internal event i $(i(E))$, and starting from any state of Ql_{ij} where a δ event is possible$(\delta(ij))$.

The disabling operator is treated similarly.

5 SOME CONSIDERATIONS ON THE APPLICABILITY OF THE METHOD

The BDD representations resulting from the proposed method are such that the number of boolean event variables they need grows logarithmically in the number of possible events, and the number of boolean state variables grows linearly in the number of parallel components. In the worst case, the number of vertices in a BDD graph is exponential in the number of variables, but in the average it is polynomial (Bryant, 1986, Burch et al., 1994). This means, for example, that specifications that make extensive use of interleaving,

and are characterized by a number of states and transitions growing exponentially in the number of parallel components, are likely to have BDD representations that grow only polynomially in the number of parallel components.

Another important point is that the BDD representations that are built by the proposed method can be used efficiently for activities such as model checking and hardware synthesis from formal specifications. As regards model checking, techniques are available to verify properties expressed as temporal logic formulas on transition systems represented by means of BDD's (Burch et al., 1994). These techniques follow the classical model checking approach, i.e. they start from the assumptions that the system to be verified is defined by an unlabelled transition relation and by a set of atomic propositions, each of which is true in some states. The temporal logic formulas are built up from atomic propositions.

This classical approach can be adapted to process algebraic specifications, if the current state of a process is defined as a pair $< s, e >$, where s is the process state, as defined in the operational semantics of process algebras, whereas e is the last event that has occurred in the process. With this new state definition, the behavior of a process is defined by an unlabelled transition relation R^*, such that $R^*(< s, e >, < s', e' >)$ holds if and only if $R(s, e', s')$ holds. R^* should be represented as a function $R^*(E, V, E', V')$, which takes value 1 iff a transition from the state represented by the assignments of (E, V) to the state represented by the assignments of (E', V') is possible. Since this function does not depend on E, in practice it is equal to the BDD representation of the corresponding labelled transition relation $R(V, E', V')$. For each possible event x, an atomic proposition P_x can be defined. P_x is true if the last event that has taken place is x, i.e. it is true in states $< s, e >$ having the event component $e = x$.

Another important application of BDD representations of process algebraic specifications in the context of protocol engineering is direct hardware implementation from low-level protocol specifications (Higashino et al., 1994). The mapping from a LOTOS specification to a BDD is indeed the crucial step towards synthesizing a sequential circuit that behaves according to the specification. In fact, by applying conventional BDD synthesis techniques, the sequential circuit can be implemented directly from the BDD representation.

6 CONCLUSIONS

This paper has presented a systematic method to build a symbolic representation of a LOTOS specification based on BDD's. This result makes it possible to represent large specifications, for which the classical state/transition models are intractable, and opens the possibility of applying the efficient BDD-based symbolic model checking and synthesis techniques to LOTOS specifications.

In this paper, only the basic LOTOS subset has been considered. However, an extension of the method to full LOTOS should be possible, and is left for further study.

There are also some other items that should be addressed by further research. One question regards the ordering of variables in BDD representations, which is known to influence heavily the size of the BDD graph. Determining an optimal ordering is known to be NP-complete. Nevertheless, heuristics for obtaining suboptimal results could be found. Another item for further research is experimenting the method on large specifications, in

order to evaluate to what extent the BDD representations are actually advantageous with respect to traditional representations.

REFERENCES

ISO (1989) IS8807: Information Processing Systems, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour.

Bolognesi T. and Caneve M. (1989) Equivalence verification:theory, algorithms and a tool in *The formal description technique LOTOS* , Elsevier Science.

Bryant R.E. (1986) Graph-based Algorithms for Boolean Function Manipulation, *IEEE Trans. on Computers*, **35**, 677-691.

Burch J.R., Clarke E.M., Long D.E., McMillan K.L., Dill D.L. (1994) Symbolic Model Checking for Sequential Circuit Verification, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, **13**, 401-424.

Cousot P., and Cousot R. (1977) Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints, in *Proc. of 4th ACM Symp. on Principles of Programming Languages*, pp. 238-252.

Fernandez J.C., Jard C., Jeron T., and Mounier L. (1992) On the fly verification of finite transition systems, in *Formal Methods in System Design*, Kluwer Academic Publishers.

Garavel H., and Najm E. (1989) TILT: from LOTOS to labelled transition systems, in *The formal description technique LOTOS* , Elsevier Science.

Higashino Y., Yasumoto K., Kitamichi J., and Taniguchi K. (1994) Hardware Synthesis from a Restricted Class of LOTOS Expressions, in *Protocol Specification, Testing and Verification XIV*, Chapman & Hall.

Holzmann G.J. (1994) An Improvement in Formal Verification, in *Proc. of 7th Int. Conference on Formal Description Techniques*, Berne, Switzerland.

Nicollin X., Sifakis J., Yovine S. (1992) Compiling Real-Time Specifications into Extended Automata, *IEEE Trans. on Software Engineering*, **18**, 794-804

Quemada J. (1992) Compressed State Space Representation in LOTOS with the Interleaved Expansion, in *Protocol Specification, Testing and Verification XI*, Elsevier Science.

Valmari A. (1990) A Stubborn Attack on State Explosion, in *2nd Int. Conf. on Computer Aided Verification*, Dimacs Series.

West H.C. (1986) Protocol validation by random state exploration, in *Protocol Specification, Testing, and Verification VI*, Elsevier Science.

Winksel G. (1989) An Introduction to Event Structures, *LNCS 354*, 364-397, Springer-Verlag.

7 BIOGRAPHY

Riccardo Sisto received the Dr. Ing. degree in Electronic Engineering and the PhD degree in Computer Engineering, both from Politecnico di Torino, Italy. Since 1991 he has been working at Politecnico di Torino in the Computer Science Department, where he acts as a researcher in the areas of computer networks, and communication protocol engineering.