Evaluating Symbolic Traversal Algorithms
Applied to Asynchronous Concurrent Systems

Marc Solé and Enric Pastor

UPC/DAC Report No. RR-2004/15 April 2004

This work has been supported in part by the Ministry of Education of Spain (CICYT) under contract TIC 2001-2476-C03-02 and grant AP2001-2819.
Evaluating Symbolic Traversal Algorithms Applied to Asynchronous Concurrent Systems

Marc Solé and Enric Pastor
Department of Computer Architecture
Technical University of Catalonia
08860 Castelldefels (Barcelona), Spain
{msole, enric}@ac.upc.es

April 2, 2004

Abstract

Symbolic reachability analysis based on Binary Decision Diagrams (BDDs) is a technique that allows the implementation of efficient invariant checking algorithms, and improves the performance of CTL/LTL verification. However, in practice it is well known that the BDD blowup problem limits the size of the systems that can be successfully verified. Along the years multiple variants of the basic reachability scheme have been introduced. However, most of them perform nicely only on synchronous systems or under the assumption that the Transition Relation (TR) can be effectively decomposed as a conjunction. This paper intends to demonstrate that the reachability algorithms based on these premises do not fit properly on asynchronous concurrent systems. Therefore it justifies developing alternative reachability algorithms for concurrent systems, or even for mixed synchronous/asynchronous schemes.

1 Introduction

State space traversal is the main computational bottleneck for most verification techniques. For invariant checking, all reachable states of the system are calculated and the desired invariants are tested to hold in all of them. If the system fails to satisfy the invariants, it is necessary to identify a counter-example that reproduces the sequence of actions that the system performs before failing. The computational complexity (the well-known state explosion problem) of invariant checking is revealed when large systems or systems that exhibit high degrees of concurrency are analyzed.

The verification community has developed efficient traversal methods [4]. However most of them are designed to improve the traversal process for synchronous systems, and do not have necessarily direct application to asynchronous concurrent systems (systems that may include asynchronous circuits, Petri nets and distributed systems in general, etc.). In synchronous systems, transition relations (TRs) are usually partitioned as conjunctions because it is impossible to represent the TR as a monolithic object. Generally, we can understand that there exist two types of optimization techniques for traversal: scheduling the application of the transition relation parts, or scheduling the order in which states are explored. The application of both strategies is basically orthogonal.

When we have a conjunctively partitioned TR, the sequence of application of each part should be scheduled in order to reduce the BDD sizes for intermediate results. The application order in this case is important because the way the variables are quantified depends on it, affecting the size of the representation. This is usually referred as the quantification schedule problem [3]. Traversal techniques based on the quantification
schedule of the TR have no practical application for asynchronous concurrent systems. In this latter case we usually have a disjunctive collection of TRs, each one describing the behavior of some component. Each individual TR is applied assuming interleaved semantics.

Classical traversal algorithms use Breadth First Search (BFS) state exploration schemes. However, it is well known that the BDD size of the intermediate reached states may be too large to be represented. In order to avoid that problem, alternative exploration schemes have been derived, most of them scheduling the state exploration mixing Breadth and Depth First Search schemes. The objective of scheduling the order in which states are explored is to manipulate intermediate results with dense representation (i.e. with a high “states versus size” ratio). Effective schedules have been derived by looking at the structure of the BDDs themselves, by exploring dense sets and postponing sparse sets to latter iterations of the traversal, when the probabilities of BDD recombination are higher. As we will see, scheduling the state exploration has also a direct application on asynchronous concurrent systems. However, the interleaved nature of this type of systems requires much more specialized scheduling schemes.

The objective of this paper is to show the special characteristics of asynchronous concurrent systems and motivate the necessity of developing specialized traversal algorithms for them. To fulfill this objective we will perform a comparative study between some of the most common existing traversal strategies and a number of strategies specially developed for asynchronous systems. A number of tools have been selected for that comparison. On one side NuSMV [13] and VIS [32], and on the other side TranSyT [31], a formal verification tool specialized for asynchronous systems. The results will highlight the importance of properly selecting the verification tool according to the type of system being analyzed. We will show that, in general, verification time can be greatly reduced, and in some cases specialized techniques will dictate the difference between enabling an efficient verification or collapsing the capabilities of the selected computer. We hope that after reading this paper, engineers trying to verify a concurrent system will think twice before selecting the verification infrastructure and traversal algorithms.

The remainder of the paper is organized as follows. Section 2 introduces and categorizes the existing research related to state traversal methodologies. Those aspects that are more relevant when comparing synchronous versus asynchronous oriented traversal algorithms will be highlighted. Section 3 provides background on the model used for asynchronous concurrent systems and on the peculiarities of their reachability analysis. Section 4 presents the translation mechanism used to remap asynchronous concurrent systems onto other tools like SMV and VIS. The selected comparison scheme and detailed descriptions of the traversal algorithms are described in Section 5. Experimental results on the application of the selected traversal algorithms on a number of benchmarks will be presented and analyzed in Section 6. Finally, Section 7 concludes the paper.

2 Traversal Strategies for Synchronous and Asynchronous Systems

Soon after the introduction of the first BDD-based symbolic traversal algorithms (e.g. [3]), the need was recognized to seek alternative reachability schemes for some systems to overcome the bottleneck produced by the BDD size explosion. Major research has been carried out along these years on several traversal strategies mostly oriented to synchronous systems.

However, as early as in Burch et al. [22] it was understood that traversal strategies designed for synchronous systems were not necessarily useful when applied to asynchronous systems. Two major elements concentrate the differences between synchronous versus asynchronous system symbolic analysis: (1) the inherent disjunctive structure of the TR for asynchronous systems, in opposition to the conjunctive structure of synchronous systems; and (2) the interleaving of concurrent actions in asynchronous systems, in opposition to the non-interleaved nature of synchronous systems.

In this section we will provide an overview of the major traversal schemes derived so far, classifying
them according to the type of strategy and describing its applicability to asynchronous system verification. We will organize this overview in three separate types of strategies:

- Conjunctive Transition Relation Scheduling
- Disjunctive Transition Relation Scheduling
- State Exploration Scheduling

Note that an additional strategy type could be included if we take into account the recently developed hybrid simulation-traversal reachability strategies [16]. However, we believe that this is out of our scope — a different comparison could be organized for those approaches. Therefore, we only concentrate on pure traversal strategies.

2.1 Conjunctive Transition Relation Scheduling

Conjunctive organization of the TR is the dominant methodology in BDD-based symbolic traversal. Both synchronous and asynchronous systems can be organized in such a way. The relation is partitioned as:

\[ N(V, V') = \bigwedge_{i=0}^{n-1} N_i(V, V'), \]

with one term \( N_i(V, V') \) per latch in the synchronous case:

\[ N_i(V, V') = (v'_i \Leftrightarrow f_i(V)), \]

where \( f_i(V) \) is the next-state function of the gate. In the asynchronous case one term \( N(V, V') \) is used for both combinational and sequential gates, in which non-deterministically the gate may change the state of the system \( (v'_i \Leftrightarrow f_i(V)) \) or may simply not change at all \( (v'_i \Leftrightarrow v_i) \):

\[ N(V, V') = (v'_i \Leftrightarrow f_i(V)) \lor (v'_i \Leftrightarrow v_i). \]

Generalizing this concept, TRs can be organized as an unrestricted conjunction of clusters. Cluster creation, from the classic point of view, has three phases. It starts from a set of conjunctive functions without structure, and then is followed by an ordering, clustering and, again, ordering process. The resulting clusters intend to exploit early quantification and variable locality, i.e. variables should enter the intermediate Image computation as late as possible and leave it as soon as possible, thus keeping the active set of variables in the intermediate steps of the computation small.

Traditionally, approaches to the conjunction scheduling problem follow a coarse grain method. Coarse-grain methods typically start from entire next-state functions and try to cluster them with the objective of determining an schedule that can eliminate variables as early as possible from the intermediate image computation product [17, 33, 21, 26]. Only when the next-state function of a variable is too large cut-points are applied to partition its cluster. An alternative fine-grain approach has been introduced in [23] in which instead of starting the clustering problem from next-state functions, it starts from single gates or small sub-circuits, and then carefully clustered to minimize the number of variables that are alive during image computation.

2.2 Disjunctive Transition Relation Scheduling

Disjunctive organization of the TR of a system was early identified as a source of optimization for asynchronous systems in [22, 3]. In this work the TR of a circuit is partitioned

\[ N(V, V') = N_0(V, V') \lor \ldots \lor N_{n-1}(V, V'), \]
with one term \( N_i(V, V') \) per both combinational and sequential gates, in which the gate changes the state of the system \( (v'_i \Leftrightarrow f_i(V)) \) and all other variables do not change at all \( \land_{j \neq i}(v'_j \Leftrightarrow v_j) \):

\[
N_i(V, V') = (v'_i \Leftrightarrow f_i(V)) \land_{j \neq i}(v'_j \Leftrightarrow v_j).
\]

Disjunctive partitions for synchronous systems are generally built around the Shannon expansion theorem. Splitting through a variable potentially reduces the size of the two resulting BDDs and effectively reduces the support of the functions. BDDs can be recursively decomposed until they meet the desired size constraints.

Narayan et al. [29, 30] present strategies for such type of partitioning and characterizes their canonicity properties. They present a reachability algorithm in which each partition is identified by a characteristic function, and those functions are kept constant along the traversal.

Cabodi et al. [6] present a similar traversal strategy. However, partitions are not kept constant, but redesigned at each traversal iteration.

Both Narayan and Cabodi exploit a disjunctive partition of the state space by defining a characteristic function for each partition. Such function can be used to simplify the TR at each partition. However, they never exploit this information as a means to improve the reachability speed provided by a BFS reachability scheme.

Finally, Moon et al. [27] explored the combination of both conjunction image computation and function splitting.

### 2.3 State Exploration Scheduling

Some authors have studied the influence of ordering the application of the TR to avoid the BDD explosion problem. Their goal is to schedule the exploration of the state space by taking only selected portions of the TR, or by delaying the exploration of certain states.

In [5] Cabodi et al. suggest that the information from a disjunctive partitioning can be further exploited in certain cases (specially for counters). Using structural information from the circuit, the TR from a particular counter \( T_{pk} \) can be obtained, apply iterative squaring on it \( T_{ps} \), and use it along the traversal process to reduce its sequential exploration depth.

In [34] Ravi and Somenzi proposed a high density traverse, which does not use the set of newly reached states as the from set for the next iteration. Instead it uses a subset of the newly reached states that has a more compact representation. This is a partial traversal, so afterward must be completed. [42] attempts to improve the subsetting mechanism by differentiating control and data-path, and keeping subsets that preserve all possible control behaviors.

In [7] Cabodi et al. use activity profiles for each BDD node in the TRs and prune the BDDs to perform a partial traversal, completed again, in the end. The activity profiles are obtained in a preliminary reachability learning phase.

In [20] Hett et al. propose a sequence of partial traverses that combine subsets of the newly reached states and dynamic TR pruning. Both manipulations are applied using the Hamming distance as the main part of the heuristic function.

In [35, 2] Ravi et al. allow the user to provide hints to guide symbolic search and alleviate the BDD explosion problem. Each hint indicates which portion of the TR should be used at each step to avoid a BDD blowup. User-defined hints are used to simplify the TR, but require the user to understand the design and also predict the BDD behavior.

State space exploration using guided techniques has become subject of wide interest. These techniques tackle the guidance of reachability analysis toward failure detection rather than to complete state space
computation. Several techniques have been introduced to guide the search toward uncovered regions of the state space.

Ganai et al. [16] introduced a combination of adaptive simulation with retrograde analysis. Adaptive simulation is based on random simulation with a backtracking mechanism to avoid getting stuck during the search. Retrograde analysis involves a combination of forward analysis with pre-images from the failure states.

Both Ganai et al. [16] and Yang et al. [41] suggest the manual insertion of guide-posts. User defined guide-posts are variables inserted in the system, which if activated during the traversal indicate that we are in the right way to find a failure. In [40] an automatic guide-post insertion mechanism is proposed. Kuehlmann et al. [24] suggest using the state reachability probability as a guide for state prioritizing. Again, Ganai et al. [15] propose a rarity-based guide that tracks latch toggle activity to improve state coverage.

3 Model Background

3.1 Modeling asynchronous systems

The main formalism that we will use to reason about concurrent systems is the finite transition system. A finite transition system (TS) [1] is a 4-tuple \( \mathcal{A} = \langle S, \Sigma, T, S_0 \rangle \), where \( S \) is a finite set of states, \( \Sigma \) is a non-empty alphabet of events, \( T \) is a transition relation such that \( T \subseteq S \times \Sigma \times S \), and \( S_0 \) are the initial states. Transitions are denoted by \( s \rightarrow s' \). An event \( e \) is enabled at state \( s \) if \( \exists s \rightarrow s' \in T \). Given an event \( e \) its firing region \( Fr(e) \) is defined as \( Fr(e) : \Sigma \rightarrow 2^S \) such that \( Fr(e) = \{ s \in S | \exists s \rightarrow s' \in T \} \).

Event \( e \) is said to be fireable at state \( s \) if \( s \in Fr(e) \).

A TS is a formalism oriented to modeling asynchronous systems that emphasizes the execution of abstract events. The concurrent execution of events is described by means of interleaving; that is, weaving the execution of events into sequences. Given the significance of individual events, the transition relation of a TS can be naturally partitioned into a disjoint set of relations, one for each event \( e \in \Sigma : T_e = \{ s \rightarrow s' \in T \} \).

We will use the VME bus protocol example in Figure 1 as motivation for the rest of the paper. Figure 1(a) describes the general structure of the VME controller, Figure 1(b) shows a fragment (only the read cycle) of a possible asynchronous implementation [14], and Figure 1(c) shows the Petri net like specification used for synthesis. The verification of this component is carried out by composing the derived implementation and the mirror image of the specification and then checking that no hazards, dead-locks, etc. exists [36].

3.2 Reachability analysis

To represent events symbolically we use a set of Boolean variables \( V \) that encode the states in the TS and a Boolean relation \( T(V, V') \) to encode the TR. The set of reachable states \( S(T(V)) \) that is reachable in any number of steps from the initial states \( S_i(V) \) is defined as the least fix-point of the following recurrence:

\[
S_0(V) = S_i(V) \\
S_{i+1}(V) = S_i(V) \cup \text{Img}(T(V, V'), S_i(V)) \ .
\]

where \( \text{Img}(T(V, V'), S_i(V)) \) is the one-step image computation applying the TR \( T(V, V') \) on a set of states \( S_i(V) \). For a given TS, this algorithm generates the state space of a system in a Breath First Search (BFS) style. The number of iterations performed by such traversal is determined by the maximum number of steps from the initial state to the first occurrence of each of the reachable states (called the sequential depth of the TS).

The state generation ratio of this technique is limited due to several factors. The sequential depth in asynchronous systems is usually large because all internal nodes in circuits should be considered (only
latches are considered in synchronous systems). The overall TR, either monolithic or in some partitioned form, will be applied at every step of the BFS process, even though only a few events are fireable at each step. BFS state exploration never takes into account the interleaved nature of asynchronous systems and the causality relations that exist between the various events.

Our conclusion is that a coarse-grained application of the TR, even if partitioned, leads to a high overhead penalty with a little state generation ratio. In order to alleviate this problem we suggest an alternative state exploration scheme, named chaining, (see Section 5) that is based on two key observations:

- A fine-grained utilization of the TRs associated to individual or groups of events will provide a better state generation versus computation effort ratio, because TRs will be only applied when they are bound to generate new states.

- An alternative BFS/DFS state exploration scheme will improve the exploration of the state space, because it can be better adapted to its interleaved nature.

Chaining applies individual TRs of events in a predetermined order such that the number of new states generated at each step is maximized. After the application of the TR of an event, the newly generated states are immediately used as domain for the next event in order, hence coining the term chaining. Figure 2 shows the general concept for two TRs A and B. If A and B are applied to the same set FROM in a BFS style, a certain number of states is reached (see Figure 2(a) and (b)). However, chaining would apply A to FROM and generate a new set of states (FROM + TO(A) in Figure 2(c)), and afterward apply B to this set (in Figure 2(d)). The number of reached states increases with almost the same computational effort. In practice, chaining can significantly reduce the number of iterations and cost of the BFS algorithm [36, 38].

The method is specially effective if the appropriate TR application order of the events is selected. Figure 3 illustrates the differences between BFS and chained traversal in the read cycle of the VME example (see Figure 1). Figure 3(a) shows that BFS traversal requires 10 applications of the full TR, that contains 10 individual TRs (thus requires 100 individual TR applications), even though in many iterations just one event is fireable. Figure 3(b) shows that chained traversal requires the application of 11 individual TRs (in the optimal case). The event order is indicated in the figure itself. Actual results from the composed VME system (specification + implementation) are even more significant. BFS requires 53 iterations to complete,
requiring the application of 1512 individual TRs; a chained traversal reduces this value to 16 iterations, with 476 applications of TRs.

Figure 4 introduces a general view of our chained traversal scheme. All algorithms for asynchronous systems introduced in Section 5 fit into the same scheme. First, the granularity of the TR is selected, either directly using the TRs of individual events or creating clusters. Once TRs are selected, the traversal itself can start. Two nested loops are used. The external loop develops the BFS-like state exploration, identifying frontier sets, detecting fixpoints and setting up the context for the inner loop. The internal loop exploits the chaining strategy and the selected fine-grained TRs. First, a TR is selected according to the current frontier set and other available contextual information. The TR is applied, potentially generating new states. These states are immediately accumulated into the frontier set (chaining), and the contextual information is updated. When, according to the context and the available frontier set, chaining is no longer useful, the inner loop is finished in order to build a new frontier set.

## 4 Generating Benchmarks for SMV and VIS

The tool TranSyT [31] uses a textual formalism to describe Boolean encoded Transition Systems called TSIF. This format declares the variables and events that are part of the system, and for each event associates a TR. This TR captures the state change produced by the execution of the event and generally only involves a small subset of all variables in the system. All other variables are assumed to remain unchanged. The execution of all events is *interleaved* in order to generate the desired state space.

The following example depicts how TSIF describes the events for output signal LDS in the VME example in Figure 1(c):

```
EVENT LDS+/1 LDS
EQN TR CONJUNCTIVE
pin0 NS(pin0)' pin1' NS(pin1) p3 NS(p3)';
NS(LDS) = LDS'; LDS = 0;
END

EVENT LDS+/2 LDS
```
Figure 3: (a) BFS traversal, and (b) BFS/DFS (chained) traversal of the VME read cycle.

EQN TR CONJUNCTIVE
\[ \text{pin7} \text{ NS(pin7)' pin8' NS(pin8) p3 NS(p3)';} \]
\[ \text{NS(LDS) = LDS'; LDS = 0; END} \]

EVENT LDS- LDS
EQN TR CONJUNCTIVE
\[ \text{p2 NS(p2)' pin12' NS(pin12);} \]
\[ \text{NS(LDS) = LDS'; LDS = 1;} \]
END

Variables pin0, pin1, pin2, pin7, pin8, pin12, p2, p3, are used to encode the places of the underlying safe Petri net, while LDS directly encodes the output signal. The next-state operator (NS) indicates the value that these values should take in case the event is executed. Note that the TRs are specified as the conjunction of two parts, the token-flow of the Petri net and the signal switch.

4.1 Mapping a TS into SMV

Translating TSIF format into SMV format is fairly straightforward. Basically, each event in the Transition System is transformed into a process in SMV. The process will have as parameters only those variables (both signals and places) that appear in the support of the original TR. For our example of signal LDS we have three processes, one for LDS−, LDS + /2, and LDS + /1 respectively:

\[ \text{event11: process LDSn__L__LDS(LDS, p2, pin12);} \]
\[ \text{event12: process LDSps2__L__LDS(LDS, pin7, pin8, p3);} \]
\[ \text{event13: process LDSps1__L__LDS(LDS, pin0, pin1, p3);} \]

Each process specifies the behavior of those variables that are defined to change in the original TR. A next-state equation is individually created for each variable that has its next-state value defined in the TR. The equation is separated in three parts: (1) minterms in with the variable rises, (2) minterms in with the variable falls, and (3) in the remaining cases the variable does not change. In our example, coming from a Petri net, we only have either rising/falling variables combined with constant variables. Special non-deterministic constructors are required for those minterms in which the next-state variable can take both values 0 and 1.
// Define the TR granularity level
{T_0 \ldots T_m} := \text{compose}\_\text{decompose}\_\text{TR}([T_n, \ldots T_m]);

// Initial reachability and frontier sets
S := From := S_{in};

\text{repeat} // start BFS iteration
  context := setup\_\text{context}(From, \{T_0 \ldots T_m\});
  \text{repeat} //start chained iteration
    T_i := select\_\text{TR}(context, From);
    From := From \cup \text{Img}(From, T_i);
    update\_\text{context}(context, From, T_i);
  \text{until} (\text{empty\_selection}(context, From));
// Update frontier set
New := From = S;
From := New;
S := S + New;
\text{until} (New == 0);
\text{return} S; /*Reached states from S_{in}*/

Figure 4: General scheme of a chained BFS/DFS traversal algorithm.

\text{LDS} - \text{specifies a falling switch for LDS if } p2 = 1 \text{ and } \text{pin12} = 0. \text{Additionally, } p2 \text{ should lose a token, while pin12 should gain a token. Otherwise, values remain the same.}

\text{MODULE LDSn\_L\_LDS(LDS, p2, pin12)}
\text{ASSIGN}
\text{next}(\text{LDS}) := \text{case}
  \text{LDS} & p2 & !\text{pin12}: 0;
  1 : \text{LDS};
\text{esac};
\text{next}(p2) := \text{case}
  \text{LDS} & p2 & !\text{pin12}: 0;
  1 : p2;
\text{esac};
\text{next}(\text{pin12}) := \text{case}
  \text{LDS} & p2 & !\text{pin12}: 1;
  1 : \text{pin12};
\text{esac};

\text{LDS + /2} \text{specifies a rising switch for LDS if } \text{pin7} = 1, \text{pin8} = 0 \text{ and } p3 = 1. \text{Additionally, pin7 and p3 should lose a token each, while pin8 should gain a token.}

\text{MODULE LDSps2\_L\_LDS(LDS, pin7, pin8, p3)}
\text{ASSIGN}
\text{next}(\text{LDS}) := \text{case}
  !\text{LDS} & \text{pin7} & !\text{pin8} & p3: 1;
  1 : \text{LDS};
\text{esac};
\text{next}(\text{pin7}) := \text{case}
  !\text{LDS} & \text{pin7} & !\text{pin8} & p3: 0;
  1 : \text{pin7};
\text{esac};
\text{next}(\text{pin8}) := \text{case}
  !\text{LDS} & \text{pin7} & !\text{pin8} & p3: 1;
  1 : \text{pin8};
\text{esac};
\text{next}(p3) := \text{case}

10
Finally, LDS \(+/1\) specifies a rising switch for LDS if pin0 == 1, pin1 == 0 and p3 == 1. Additionally, pin0 and p3 should lose a token each, while pin1 should gain a token.

4.2 Mapping a TS into VIS

Translating TSIF format into BLIF format has an additional complexity because: (1) BLIF requires the definition of each signal to be specified in a single gate-network; (2) BLIF has no support for interleaving semantics. Both problems are solved by means of a clever network definition, partially inspired by the way SMV internally builds its TRs.

Basically, each variable in the Transition System is transformed into a network of gates in BLIF format. The network ends at a BLIF latch that separates current-state from next-state; and a set/reset gate where to plug separated set and reset subnetworks. Figure 5 provides a detailed view of the selected architecture that is described for variable LDS. The following BLIF code shows both the latch and the set/reset function in that case:

```
.latch LDS_ LDS 0
## LOGIC FOR VARIABLE LDS
## Set / reset structure
.names LDS_set LDS_reset LDS LDS_
1-- 1
-01 1

Set and reset functions are created, one for each event that changes the value of the variable. Note that one extra selection signal is added to each one of these sub-networks. This signals corresponds to a non-deterministic decoding function will be latter used to implement the required interleaving semantics.

## Set/reset functions for atom: LDS-[LDS]
.names LDS p2 pinl2 LDS_reset1_
110 1

.names LDS_reset1_ LDSn__L__LDS_sel LDS_reset1
```

11
## Set/reset functions for atom: LDS+/2[LDS]
.names LDS pin7 pin8 p3 LDS_set1_0101
.names LDS_set1_ LDSps2__L__LDS_sel LDS_set1
## Set/reset functions for atom: LDS+/1[LDS]
.names LDS pin0 pin1 p3 LDS_set2_0101
.names LDS_set2_ LDSps1__L__LDS_sel LDS_set2

Finally, all set and reset sub-networks are or-ed together to be connected to the set/reset gate.

## Set OR gate
.names LDS_set1 LDS_set2 LDS_set
## Reset OR gate
.names LDS_reset1 LDS_reset

Several extra signals have been added to the BLIF network, one for each event in the system, and three of them related to LDS network: LDS\_n\_LDS\_sel1, LDSps2\_L\_LDS\_sel1, and LDSps1\_L\_LDS\_sel1. These signals are generated by a one-hot decoder driven by a number of artificially created primary inputs. In that way, at each state, all enabled events may be executed, but always in a mutual exclusive fashion, thus resembling the required interleaving semantics.

## PROCESS SELECTOR DECODING FUNCTION
... .names sel0 sel1 sel2 sel3 sel4 LDSn__L__LDS_sel 11010
.names sel0 sel1 sel2 sel3 sel4 LDSps2__L__LDS_sel 10110
.names sel0 sel1 sel2 sel3 sel4 LDSps1__L__LDS_sel 10110

The process will have as parameters only those variables (both signals and places) that appear in the support of the original TR. For our example of signal LDS we have:

## NuSMV, VIS and TranSyT Traversal Algorithms

In this section we will give a brief overview of the traversal methods implemented in three selected verification tools: NuSMV, VIS and TranSyT. The first is a well known verification tool [13] basically aimed to synchronous verification, although its language allows the specification of asynchronous concurrent systems. In the case of VIS [32], it is also a well known tool but its input language is much more synchronous circuit oriented, although it is possible to translate deterministic asynchronous systems to this kind of representation (see Section 4.2. On the other hand, TranSyT [31] is completely oriented to concurrent system
verification. All tools use the same underlying data structure (BDDs)\(^1\) but have exponential different running times when computing the reachability set of several concurrent systems (see Section 6).

When talking about verification tools for concurrent systems, it must be noted that Ciardo et al. achieved very impressive results for rather regular examples using a more general data structure (MDDs) in conjunction with a “saturation” algorithm [12, 9]. Their method, however, is aimed to analyze Petri Nets and needs the user to specify a partitioning suitable for the problem. As a result, the tool they developed, named SMART [11, 8, 25], is not fully automated, and this is specially true when analyzing circuits. Although all the problems derived from the conversion of a circuit into an input file that SMART can accept have a well defined algorithmic solution, this paper shows that exponential wins can also be achieved with respect to conjunctive traversal methods, by only changing the way in which the traversal is performed, thus obtaining ease of use as well as faster analysis. Because we intend to compare traversal methods, we only compared tools based on the same data structure, so SMART was not included in the result tables. Although for regular (scalable) examples, improvements are not as good as the ones obtained by Ciardo et al. [10], the techniques presented in this paper need only minor changes to be implemented in already existing verification tools using a BDD package, thus we think they can immediately benefit research groups working on concurrent systems.

5.1 NuSMV image computation methods

The NuSMV tool offers only one traversal method that may be used with any valid conjunctive decomposition of the TR. NuSMV implements three main methods to perform this conjunctive decomposition and order the application of the resulting clusters. The methods are named as: Monolithic, Threshold and IWLS95.

Beside these three main methods, NuSMV offers several parameters that may affect to a great extent the partitioning. For instance, one may enforce NuSMV to group TRs according to their affinity as defined in [26]: the proportion of the number of variables two transitions share over the total number of their variables. By default NuSMV uses affinity in methods Threshold and IWLS95.

Monolithic

This method simply tries to build a monolithic TR from the bit relations. Although this method guaran-

\(^1\)VIS uses MDDs but they are implemented using BDDs
tees the minimum number of BDD operations during traversal (if iterative squaring is not used), to build a monolithic TR starts to be an unfeasible task for intermediate-sized designs. In most of the examples shown in Section 6 monolithic TR building did not even finish in the given time.

**Threshold**

The bit TRs are conjoined in clusters, until the size (in BDD nodes) of the cluster exceeds a certain threshold. If affinity is used, then TRs are first ordered, thus TRs with highest affinity are clustered first. The user may specify this threshold, that has a default value of 1000 nodes. Resulting clusters are not ordered after this process.

**IWLS95**

It is an implementation of the method described in [33]. First of all, the transition relations are ordered using a heuristic, and then conjoined in clusters until a threshold is reached. By default this first ordering is deactivated in NuSMV so, in fact, by using default settings, the clustering produced by this method is the same as the **Threshold**. However, afterward the clusters are ordered using the same heuristic. The heuristic used in both orderings is a combination of weighted factors, and the weights are also user-modifiable parameters.

The factors considered by the heuristic are the following (refer to [33] for an exact definition): the number of variables that can be existentially quantified when the cluster is applied, the number of present state and primary input variables in its support, the number of present state and primary input variables not yet quantified, the number of next state variables introduced by its application, the number of next state variables not yet introduced, the maximum BDD index of the variable to quantify in its support and the maximum BDD index of the variables that remain to quantify out.

### 5.2 VIS image computation methods

The VIS tool offers two different traversal methods. The first one, like NuSMV, can be used with any valid conjunctive decomposition of the transition relation. VIS implements three main methods to perform this conjunctive decomposition and order the application of the resulting clusters. The methods are named as: **Monolithic**, **IWLS95** and **MLP**. The first two are equivalent to what has been said of their NuSMV counterparts, so we will just explain **MLP**. The second traversal method uses a hybrid approach, dynamically deciding whether to use TRs or transition functions. It must be noted that VIS implements some other techniques that are orthogonal to the method used. In particular they provide the hints mechanism [35] as well as high density traversals [34]. As hints demand user interaction we chose not to use this technique. On the other hand, we ran all the methods with and without high density traversal.

**MLP** [26]

Broadly speaking TRs are ordered in a way such that the dependency matrix between relations and variables tends to be triangular. This is done in order to try to achieve a minimum average lifetime for the variables. Once ordered, bit relations are clustered based on their **affinity**.

**Hybrid** [27]

This method is the only one in VIS that dynamically adapts to what is happening during the traversal process. The main idea is that, depending on the structure of the dependency matrix, one should better conjoin or split. If dependency matrix tends to be triangular, then conjunction is used, otherwise a variable is chosen and then transitions are split accordingly. This process, in general, produces problems with easier conjunctions. Based on this observation, the method switches from conjunction to split and the other way round depending on the dependency matrix and how previous decisions performed. It must be noted that in the conjunction phase one can use either **IWLS95** or **MLP**. We chose **IWLS95**.

**High density** [34]
This technique tries to prevent the methods from having to handle too big BDDs. If during the computation of some BDD, the size of the partially computed BDD exceeds a given threshold, the computation is stopped, a dense subset of states is chosen (with a much smaller representation) and the process is resumed. When no new states are discovered, the process is started again with a larger threshold, and this is done iteratively until all states are discovered.

5.3 TranSyT image computation methods

In [38] several traversal algorithms, that conform with the traversal scheme of Section 3, were introduced. Here we give a short description of them. In contrast with the methods used in NuSMV and VIS which are synchronous oriented, and thus, use conjunctively partitioned transition relations, transition relations of concurrent systems are more easily represented with disjunctive partitioning. When disjunctive partitioning is used it is also important to schedule the application of the transition relations, specially when the chaining technique is used. Recall that the chaining technique is conceptually very simple: instead of firing from the same origin set (usually named as from set) each one of the transitions, all the states produced by the application of each transition is accumulated into the origin set as soon as they are produced. The methods described below are just heuristics to try to derive a good scheduling order for this technique in order to minimize the number of events needed, thus the possibility of a BDD operation that produces a blow up.

Basic chaining

This method uses chaining, but uses the same static event scheduling as plain BFS, that is, the order of event definition in the input file. This method was introduced in [36] and it is useful as a ground basis to compare the amount of reduction due to the chaining technique itself and the amount due to the particular scheduling. In some examples (notably buffers or FIFOs) it may achieve very good results as the order of event definition in the input file is already almost optimal.

Token traversal

Before starting the traversal, a causality analysis between events is performed. An event $e_i$ is said to be a causal successor of event $e_j$ if potentially there exists some state where the activation of $e_j$ enables $e_i$. This information may be derived directly from the transition relations of the events and is stored in a sparse matrix, as usually one event is a causal predecessor of very few events.

A vector of integers is used to determine which event would be fired next. Each event is assigned a place in the vector, and the places of events that are initially fireable are set to 1, the others are set to 0. The algorithm choses the event with the highest value and fires it. In case of a tie, it nondeterministically choses one of them.

After the event is fired, it adds the resulting new states to the reached set and the from set (chaining technique) and clears the position of the event. If no new states were produced no further action is done, otherwise, the vector places of the events that are causal successors of the one fired, increment their value by one.

When all the vector has zeros, the frontier set is computed and all the process starts again (we check which events are fireable, and put a one in its place, ...). This process continues until the frontier set becomes the empty set, which means that a fixpoint has been reached.

We must stress that this approach (as well as the others presented in this section) are dynamic, in the sense that the scheduling may change during the traversal process in response to the states discovered so far.

Weighed token traversal

This is a variation of the former method. It has only two minor changes: (1) instead of placing a one when the event is fireable, we put the exact number of states in which it is fireable; (2) when an event is fired, we compute the new states it produced, and increment the vector places of its causal successor events by the same amount as the number of these new states in which this particular causal event is fireable. For
instance, imagine that we fire event \( a \) that has two successors: events \( b \) and \( c \). The firing of \( a \) produced four new states, three in which \( b \) is fireable and two in which \( c \) is fireable. Then we will increment by three the vector place of \( b \) and by two the vector place of \( c \).

This method selects more carefully the next firing event at the cost of performing more BDD operations.

**Dynamic event-clustered traversal**

This method keeps track, for each event, of how many new states have been discovered in which it can be fired. The algorithm selects always the event with more states to fire from. This states can be stored in a particular from set for the event or, if its BDD size is bigger than the reached set, we use this latter set as the from set, in order to save space. The selected event is fired, its from set is cleared and the new states produced are distributed over the from sets of the rest of the events. When all the from sets are empty, a fixpoint has been reached.

**Transition relation cluster-closure traversal**

The idea of this method is similar to the concepts of NuSMV methods, but adapted to concurrent systems. First of all, we cluster events under some criteria (usually shared variables, but manual configuration is also possible) until a threshold is reached, then we apply iterative squaring to the cluster. This allows to reach states with distance from the frontier greater than one, with just one application of the transition.

This method it is not a traversal method in itself, instead is a way to manage the transition relations trying to keep the transition BDDs small, while getting more states per firing. It can be combined with any of the other three methods. For this reason, we did chose not to present results on it in Section 6.

### 6 Comparative Evaluation

#### 6.1 Comparison Parameters

We conducted a number of traversal experiments with and without dynamic reordering activated. In all cases a suitable variable order was derived. We used the same initial variable order and dynamic reordering parameters in all tools. We ran several sets of benchmarks. Some of them are scalable, in the sense that they are built by adding instances of the same component. This kind of examples are useful to have a hindsight on the asymptotic behavior of the methods used.

For each set of benchmarks we have three tables: a general table where information about the benchmarks is given, and two result tables, one for traversal methods without reordering and one with dynamic reordering on.

In Table 1 we give the number of variables needed to encode the benchmark and the number of reachable states. We also give the number of parts (TR P) in which the TR was split when using NuSMV, VIS or TranSyT. The number of parts using TranSyT corresponds exactly to the number of events defined in the system (as well as the number of processes in NuSMV). Also information about the TR sizes for both partitionings is given (total non-shared number of BDD nodes of all clusters or events) (TR S). The variable order used to gather this statistics was the same for all the tools and was equal to the order used in fixed order traversal. Note that we used several partitioning schemes in VIS, so this tool has two entries. The Hybrid and IWLS methods share the same partitioning, as we used IWLS method to conjoin in the Hybrid. Remember that TranSyT partitioning is disjunctive while in NuSMV and VIS is conjunctive.

For the result tables, the factors compared were the CPU time needed to complete the reachability analysis and the maximum representation size needed by the tool, in terms of BDD nodes. This corresponds to the peak number of live nodes (in thousands of nodes) in the BDD manager.

In the fixed order tables, we also include the iterations needed by the algorithm to finish the traverse, but this numbers are really only directly comparable between BFS and BFS with chaining methods, as other methods have different concepts of “iteration”. For this reason we also included the number of events fired.
This is the number of parts of a TR that have been applied to complete the process. For instance, in NuSMV and VIS this corresponds to the number of iterations multiplied by the number of clusters created. This value is not exact for the Hybrid method, as several conjunctions are counted as one conjunction depending on the number of splits, but we decided to keep the value to have a reference.

For the tables where dynamic reordering was allowed, we also evaluated the impact of reordering in the results. Obviously some of the parameters were exactly the same using reordering or not (i.e. the number of iterations and number of events fired) so instead of these sections we include the number of reorderings triggered during traversal process as well as the total time consumed in this task.

The experiments were conducted on a Pentium IV with 1 Gb of memory, using Linux. The processes were limited to use an hour of elapsed time. If in this time, the verification process was not finished this is marked by a dash in the appropriate entry of the table.

6.2 Experimental results

We conducted an extensive set of experiments with more than a hundred of different files. However for obvious space reasons we do only give some of the most significant results. Tables 1, 2, 3, 4 and 5 show the results obtained. We analyzed two parameterizable benchmarks: a tree arbiter and the well known Muller’s C-element pipeline. Both benchmarks are signal transition graphs. Additional benchmarks are briefly described here:

- **PCC** Pausible clock controller for heterogeneous systems in [43].
- **GALS** Globally-Asynchronous Locally-Synchronous design in [28].
- **RGD-arbiter** asP*, RGD arbiter in [19] described at transistor level.
- **IPCMOS** A pulse-based controller for asynchronous pipelines in [37].
- **STARI** A self-timed pipeline in [18].

To make result tables as small as possible we will use the following codes to identify the traversal methods:

- N for NuSMV using threshold and affinity.
- B for TranSyT BFS.
- C for TranSyT BFS with chaining.
- T for TranSyT Token traverse.
- W for TranSyT Weighed Token traverse.
- D for TranSyT Dynamic Event Clustered traverse.
- I for VIS IWLS.
- H for VIS Hybrid method.
- M for VIS MLP method.
- IH for VIS IWLS with high density.
- HH for VIS Hybrid method with high density.
- MH for VIS MLP method with high density.
<table>
<thead>
<tr>
<th>Name</th>
<th>Vars</th>
<th>Reachable</th>
<th>TR P</th>
<th>TR S</th>
<th>TR P</th>
<th>TR S</th>
<th>TR P</th>
<th>TR S</th>
<th>TR P</th>
<th>TR S</th>
</tr>
</thead>
<tbody>
<tr>
<td>tree5</td>
<td>74</td>
<td>1620</td>
<td>24</td>
<td>159</td>
<td>5</td>
<td>1871</td>
<td>17</td>
<td>4540</td>
<td>4</td>
<td>2886</td>
</tr>
<tr>
<td>tree10</td>
<td>134</td>
<td>492075</td>
<td>44</td>
<td>294</td>
<td>5</td>
<td>45714</td>
<td>17</td>
<td>17996</td>
<td>9</td>
<td>14226</td>
</tr>
<tr>
<td>tree15</td>
<td>194</td>
<td>4.13489e+08</td>
<td>64</td>
<td>429</td>
<td>6</td>
<td>389779</td>
<td>–</td>
<td>–</td>
<td>11</td>
<td>334233</td>
</tr>
<tr>
<td>tree20</td>
<td>254</td>
<td>4.06792e+10</td>
<td>84</td>
<td>565</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tree30</td>
<td>374</td>
<td>3.08837e+15</td>
<td>124</td>
<td>834</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tree40</td>
<td>494</td>
<td>2.22891e+20</td>
<td>164</td>
<td>1105</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tree50</td>
<td>614</td>
<td>1.55545e+25</td>
<td>204</td>
<td>1642</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tree60</td>
<td>734</td>
<td>7.09227e+34</td>
<td>284</td>
<td>1915</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>muller_5</td>
<td>50</td>
<td>20</td>
<td>10</td>
<td>100</td>
<td>2</td>
<td>2081</td>
<td>3</td>
<td>2465</td>
<td>2</td>
<td>1155</td>
</tr>
<tr>
<td>muller_10</td>
<td>100</td>
<td>420</td>
<td>20</td>
<td>200</td>
<td>3</td>
<td>4198</td>
<td>7</td>
<td>4693</td>
<td>5</td>
<td>2364</td>
</tr>
<tr>
<td>muller_15</td>
<td>150</td>
<td>6006</td>
<td>30</td>
<td>300</td>
<td>4</td>
<td>7640</td>
<td>11</td>
<td>7585</td>
<td>10</td>
<td>4191</td>
</tr>
<tr>
<td>muller_20</td>
<td>200</td>
<td>251940</td>
<td>40</td>
<td>400</td>
<td>5</td>
<td>10309</td>
<td>14</td>
<td>9400</td>
<td>9</td>
<td>4873</td>
</tr>
<tr>
<td>muller_25</td>
<td>250</td>
<td>4.08595e+06</td>
<td>50</td>
<td>500</td>
<td>6</td>
<td>15398</td>
<td>17</td>
<td>11718</td>
<td>11</td>
<td>6103</td>
</tr>
<tr>
<td>muller_30</td>
<td>300</td>
<td>6.099e+07</td>
<td>60</td>
<td>600</td>
<td>7</td>
<td>27668</td>
<td>20</td>
<td>14734</td>
<td>13</td>
<td>7527</td>
</tr>
<tr>
<td>muller_40</td>
<td>400</td>
<td>4.64139e+10</td>
<td>80</td>
<td>800</td>
<td>–</td>
<td>–</td>
<td>25</td>
<td>19352</td>
<td>19</td>
<td>9946</td>
</tr>
<tr>
<td>muller_50</td>
<td>500</td>
<td>3.61071e+13</td>
<td>100</td>
<td>1000</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>muller_60</td>
<td>600</td>
<td>8.38369e+15</td>
<td>120</td>
<td>1200</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>muller_70</td>
<td>700</td>
<td>7.01713e+18</td>
<td>140</td>
<td>1400</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>muller_80</td>
<td>800</td>
<td>5.82061e+21</td>
<td>160</td>
<td>1600</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>muller_90</td>
<td>900</td>
<td>1.34627e+24</td>
<td>180</td>
<td>1800</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>muller_100</td>
<td>1000</td>
<td>1.16143e+27</td>
<td>200</td>
<td>2000</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IPCMOS-C1</td>
<td>26</td>
<td>2516</td>
<td>21</td>
<td>71</td>
<td>1</td>
<td>8378</td>
<td>2</td>
<td>1474</td>
<td>2</td>
<td>795</td>
</tr>
<tr>
<td>IPCMOS-C2</td>
<td>42</td>
<td>372368</td>
<td>35</td>
<td>121</td>
<td>2</td>
<td>6361</td>
<td>4</td>
<td>2806</td>
<td>4</td>
<td>1130</td>
</tr>
<tr>
<td>IPCMOS-C4</td>
<td>74</td>
<td>8.15635e+9</td>
<td>63</td>
<td>220</td>
<td>4</td>
<td>8649</td>
<td>4</td>
<td>2579</td>
<td>4</td>
<td>1103</td>
</tr>
<tr>
<td>IPCMOS-A1</td>
<td>52</td>
<td>532</td>
<td>16</td>
<td>107</td>
<td>2</td>
<td>1959</td>
<td>3</td>
<td>2081</td>
<td>3</td>
<td>1070</td>
</tr>
<tr>
<td>IPCMOS-A2</td>
<td>80</td>
<td>14896</td>
<td>24</td>
<td>165</td>
<td>2</td>
<td>24478</td>
<td>5</td>
<td>3460</td>
<td>5</td>
<td>1446</td>
</tr>
<tr>
<td>IPCMOS-A4</td>
<td>136</td>
<td>1.16785e+12</td>
<td>40</td>
<td>281</td>
<td>3</td>
<td>45314</td>
<td>11</td>
<td>6910</td>
<td>7</td>
<td>3077</td>
</tr>
<tr>
<td>IPCMOS-A8</td>
<td>248</td>
<td>7.17824e+12</td>
<td>72</td>
<td>513</td>
<td>5</td>
<td>95867</td>
<td>20</td>
<td>13163</td>
<td>11</td>
<td>6336</td>
</tr>
<tr>
<td>IPCMOS-A16</td>
<td>472</td>
<td>2.71195e+24</td>
<td>136</td>
<td>977</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>star14</td>
<td>56</td>
<td>5.40504e+06</td>
<td>56</td>
<td>182</td>
<td>4</td>
<td>7327</td>
<td>6</td>
<td>3331</td>
<td>5</td>
<td>3403</td>
</tr>
<tr>
<td>star18</td>
<td>104</td>
<td>1.07225e+12</td>
<td>104</td>
<td>358</td>
<td>6</td>
<td>43982</td>
<td>11</td>
<td>8065</td>
<td>9</td>
<td>6806</td>
</tr>
<tr>
<td>star12</td>
<td>152</td>
<td>2.12662e+17</td>
<td>152</td>
<td>535</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>star16</td>
<td>200</td>
<td>4.21776e+22</td>
<td>200</td>
<td>707</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>RGD-A</td>
<td>98</td>
<td>3.35313e+09</td>
<td>70</td>
<td>280</td>
<td>4</td>
<td>50216</td>
<td>11</td>
<td>6638</td>
<td>8</td>
<td>3854</td>
</tr>
<tr>
<td>RGD-C</td>
<td>126</td>
<td>3.49018e+13</td>
<td>98</td>
<td>342</td>
<td>6</td>
<td>53859</td>
<td>12</td>
<td>8689</td>
<td>12</td>
<td>5563</td>
</tr>
<tr>
<td>GALS</td>
<td>84</td>
<td>1232</td>
<td>15</td>
<td>369</td>
<td>2</td>
<td>5437</td>
<td>6</td>
<td>4525</td>
<td>5</td>
<td>1788</td>
</tr>
<tr>
<td>PCC</td>
<td>68</td>
<td>989184</td>
<td>30</td>
<td>221</td>
<td>3</td>
<td>25972</td>
<td>5</td>
<td>2688</td>
<td>5</td>
<td>2158</td>
</tr>
</tbody>
</table>

Table 1: General tables
<table>
<thead>
<tr>
<th>n</th>
<th>Iterations</th>
<th>Events fired</th>
<th>Peak BDD</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>B</td>
<td>C</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>37</td>
<td>37</td>
<td>17</td>
<td>114</td>
</tr>
<tr>
<td>10</td>
<td>72</td>
<td>72</td>
<td>32</td>
<td>365</td>
</tr>
<tr>
<td>15</td>
<td>107</td>
<td>47</td>
<td>6912</td>
<td>3072</td>
</tr>
<tr>
<td>20</td>
<td>-142</td>
<td>62</td>
<td>-12012</td>
<td>5292</td>
</tr>
<tr>
<td>30</td>
<td>-212</td>
<td>92</td>
<td>-26412</td>
<td>11532</td>
</tr>
<tr>
<td>40</td>
<td>-282</td>
<td>122</td>
<td>-46412</td>
<td>20172</td>
</tr>
<tr>
<td>50</td>
<td>- -</td>
<td>-152</td>
<td>-31212</td>
<td>-</td>
</tr>
<tr>
<td>60</td>
<td>- -</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>70</td>
<td>- -</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 2: Fixed order tables (NuSMV & Transyt)**
<table>
<thead>
<tr>
<th>n</th>
<th>Iterations</th>
<th>Events fired</th>
<th>Peak BDD</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IH</td>
<td>HH</td>
<td>MH</td>
<td>I</td>
</tr>
<tr>
<td>5</td>
<td>39</td>
<td>39</td>
<td>39</td>
<td>304</td>
</tr>
<tr>
<td>10</td>
<td>86</td>
<td>74</td>
<td>102</td>
<td>1241</td>
</tr>
<tr>
<td>15</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>30</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>35</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>40</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>45</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>50</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>55</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>60</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>65</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>70</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

muller (VIS)

<table>
<thead>
<tr>
<th>n</th>
<th>Iterations</th>
<th>Events fired</th>
<th>Peak BDD</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>39</td>
<td>39</td>
<td>39</td>
<td>304</td>
</tr>
<tr>
<td>10</td>
<td>86</td>
<td>74</td>
<td>102</td>
<td>1241</td>
</tr>
<tr>
<td>15</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>30</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>35</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>40</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>45</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>50</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>55</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>60</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>65</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>70</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

IPCMOS-C (VIS)

<table>
<thead>
<tr>
<th>n</th>
<th>Iterations</th>
<th>Events fired</th>
<th>Peak BDD</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>84</td>
</tr>
<tr>
<td>2</td>
<td>69</td>
<td>69</td>
<td>69</td>
<td>276</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

IPCMOS-A (VIS)

<table>
<thead>
<tr>
<th>n</th>
<th>Iterations</th>
<th>Events fired</th>
<th>Peak BDD</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29</td>
<td>29</td>
<td>29</td>
<td>84</td>
</tr>
<tr>
<td>2</td>
<td>55</td>
<td>55</td>
<td>55</td>
<td>270</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

stari (VIS)

<table>
<thead>
<tr>
<th>n</th>
<th>Iterations</th>
<th>Events fired</th>
<th>Peak BDD</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>251</td>
<td>213</td>
<td>253</td>
<td>1212</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

misc (VIS)

<table>
<thead>
<tr>
<th>n</th>
<th>Iterations</th>
<th>Events fired</th>
<th>Peak BDD</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGD-A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RGD-C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GALS</td>
<td>66</td>
<td>66</td>
<td>66</td>
<td>390</td>
</tr>
<tr>
<td>PCC</td>
<td>97</td>
<td>83</td>
<td>93</td>
<td>345</td>
</tr>
</tbody>
</table>

Table 3: Fixed order tables (VIS)
<table>
<thead>
<tr>
<th>n</th>
<th>Reorderings</th>
<th>Reordering Time</th>
<th>Peak BDD</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N B C T W D</td>
<td>N B C T W D</td>
<td>N B C T W D</td>
<td>N B C T W D</td>
</tr>
<tr>
<td>5</td>
<td>2 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
<td>9 4 3 3 3 3</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>10</td>
<td>9 2 2 1 1 1</td>
<td>26 1 2 1 1 1</td>
<td>166 16 12 11 7 8</td>
<td>36 3 3 0 1 1</td>
</tr>
<tr>
<td>15</td>
<td>11 2 2 2 1 1</td>
<td>105 5 7 4 2 2</td>
<td>505 43 31 38 15 15</td>
<td>512 20 10 6 2 3</td>
</tr>
<tr>
<td>20</td>
<td>3 3 3 1 1 1</td>
<td>– 0 0 0 0 0</td>
<td>– 85 68 105 24 24</td>
<td>– 102 17 7 0 1</td>
</tr>
<tr>
<td>30</td>
<td>3 3 5 1 1 1</td>
<td>– 74 96 268 17 17</td>
<td>– 204 193 470 51 51</td>
<td>– 808 224 337 17 21</td>
</tr>
<tr>
<td>40</td>
<td>3 3 5 1 1 1</td>
<td>– 191 246 610 41 40</td>
<td>– 385 432 717 87 87</td>
<td>– 3234 752 1002 42 51</td>
</tr>
<tr>
<td>50</td>
<td>4 1 1 1 1</td>
<td>– 1044 – 81 82</td>
<td>– 702 – 132 132</td>
<td>– 2504 – 85 108</td>
</tr>
<tr>
<td>60</td>
<td>– – 1 1 1</td>
<td>– – – – 145 147</td>
<td>– – – – 188 188</td>
<td>– – – – 151 197</td>
</tr>
<tr>
<td>70</td>
<td>– – – – 1 1</td>
<td>– – – – – – 252 252</td>
<td>– – – – – – 9 97</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4: Dynamic reordering tables (NuSMV & Transyt)**
<table>
<thead>
<tr>
<th>n</th>
<th>Reordering</th>
<th>Reordering Time</th>
<th>Peak BDD</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>H</td>
<td>M</td>
<td>IH</td>
</tr>
<tr>
<td>----</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>9</td>
<td>13</td>
<td>–</td>
</tr>
<tr>
<td>20</td>
<td>–</td>
<td>10</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>30</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>40</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>50</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>60</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>70</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>muller (VIS)</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>25</td>
<td>13</td>
<td>21</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>30</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>40</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>50</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>60</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>70</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>80</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>90</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>100</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IPCMOS-C (VIS)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IPCMOS-A (VIS)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>16</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>start (VIS)</td>
<td>4</td>
<td>9</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>16</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>misc (VIS)</td>
<td>RGD-A</td>
<td>13</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>RGD-C</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>GALS</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>PCC</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 5: Dynamic reordering tables (VIS)
6.3 Discussion

Available results demonstrates that for asynchronous systems the selection of a conjunctive partitioned TR is clearly inappropriate. Table 1 shows several orders of magnitude in the number of BDD nodes required to represent TRs. This is true because in a disjunctive partition all variables that should not change the state can be eliminated from the support of the TR, and also from the set of variables that should be quantified. Adding these “equivalence” factors in the conjunctive representation complicates the BDD representation.

Tables 2, 3, 4, 5 show that even a BFS traversal scheme can be reasonably efficient if used in combination with a disjunctive representation of the TR. CPU times are far from being efficient, but at least mid-size examples can be completed. Using the chaining technique drastically improves the traversal process consistently across all benchmarks. It is not clear, however, the particular heuristic that will provide better result on each example. Further analysis is required in order to try to classify systems according the traversal strategy to be applied.

It is interesting to note that TranSyT performs consistently well with a good starting variable order and with or without automatic reorder activated. However, NuSMV and VIS perform significantly better when automatic reordering is available. We believe that this is because the initial variable orders were designed to minimize the size of the reachability sets, and in both NuSMV and VIS the size of the TR is the dominant BDD factor.

7 Conclusions

This paper justifies the significant differences that exists between synchronous and asynchronous systems and their impact during BDD-based symbolic traversal. After describing the most relevant traversal strategies that exists nowadays, we have applied one of them (built inside NuSMV) on several asynchronous benchmarks. We have shown that the results are extremely disappointing, with huge BDD size and extremely long CPU times even for small systems. The paper has also review some of the existing traversal strategies specially designed for asynchronous systems. Comparison on the same set of benchmarks is clearly favorable on the specialized traversal strategies, hence justifying their development. This study also rises two additional questions that should be studied in the future. Is the chaining strategy applicable to synchronous system with coarse-grained disjunctive TRs? And, when analyzing GALS systems (Globally Asynchronous Locally Synchronous), how should be combined synchronous and asynchronous oriented traversal strategies?

References


