Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning*

Gianfranco Ciardo and Andy Jinqing Yu

Department of Computer Science and Engineering
University of California, Riverside
{ciardo,jqyu}@cs.ucr.edu

Abstract. We propose a new *saturation*-based symbolic state-space generation algorithm for finite discrete-state systems. Based on the structure of the high-level model specification, we first disjunctively partition the transition relation of the system, then conjunctively partition each disjunct. Our new encoding recognizes *identity transformations* of state variables and exploits *event locality*, enabling us to apply a recursive fixed-point image computation strategy completely different from the standard breadth-first approach employing a global fix-point image computation. Compared to breadth-first symbolic methods, saturation has already been empirically shown to be several orders more efficient in terms of runtime and peak memory requirements for asynchronous concurrent systems. With the new partitioning, the saturation algorithm can now be applied to completely general asynchronous systems, while requiring similar or better run-times and peak memory than previous saturation algorithms.

1 Introduction

Formal verification techniques have received much attention in the past decade. In particular, BDD-based [4] symbolic model checking [10, 17] has been successfully applied in industrial settings. However, even if BDDs can result in great efficiency, symbolic techniques remain a memory and time-intensive task.

We focus on symbolic state-space generation, a fundamental capability in symbolic model checking, and target asynchronous concurrent systems, including asynchronous circuits, distributed software systems, and globally-asynchronous locally-synchronous systems (GALSs), which are increasingly being used in complex hardware and embedded systems, such as System-on-Chip designs.

The standard approach to state-space generation uses a breadth-first strategy, where each iteration is an image computation. This corresponds to finding a "global" fixed-point of the transition relation. The *saturation* algorithm we introduced in [6] uses instead a completely different iteration strategy, which has been shown to excel when applied to asynchronous concurrent systems.

Saturation recognizes and exploits the presence of *event locality* and recursively applies multiple "local" fixed-point iterations, resulting in peak memory requirements, and consequently runtimes, often several orders of magnitude

* Work supported in part by the National Science Foundation under grants CNS-0501747 and CNS-0501748.

smaller than for traditional approaches. As introduced, however, saturation requires a *Kronecker-consistent* decomposition of the high-level model into *component models*. While this is not a restriction for some formalisms (e.g., ordinary Petri nets, even if extended with inhibitor and reset arcs), it does impose constraints on the decomposition granularity in others (e.g., Petri nets with arbitrary marking-dependent arc cardinalities or transition guards). For some models, these constraints may prevent us from generating the state space because each component model is too large. A particularly important example is the analysis of software.

In [20], Miner proposed a saturation algorithm applicable to models not satisfying Kronecker consistency, but its cost approaches that of an explicit generation in models where an event affects many state variables. In this paper, after giving an overview of state-space and transition relation encodings, we formalize previous saturation algorithms in a unifying framework (Sect. 2). Then, we present a new transition relation encoding based on a *disjunctive-conjunctive partition* and *identity-reduced* decision diagrams (Sect. 3). This allows us to define a new saturation algorithm that does not require Kronecker consistency, like [20], nor a priori knowledge of the state variable bounds, like [7], and is exponentially more efficient in certain models (Sect. 4). We present preliminary memory and runtime results for our approach and compare it to NuSMV [9] and SPIN [14] (Sect. 5). Finally, we report related work and our conclusions (Sect. 6).

2 Preliminaries

We consider a discrete-state model represented by a Kripke structure $M = (\widehat{\mathcal{S}}, \mathcal{S}_{init}, \mathcal{R}, L)$, where $\widehat{\mathcal{S}}$ is a finite set of states, $\mathcal{S}_{init} \subseteq \widehat{\mathcal{S}}$ is a set of initial states, and $\mathcal{R} \subseteq \widehat{\mathcal{S}} \times \widehat{\mathcal{S}}$ is a transition relation. We assume the (*global*) model state to be a sequence of K *local state* variables, $(x_K, ..., x_1)$, where, for $K \geq l \geq 1$, $x_l \in \{0, 1, ..., n_l - 1\} = \mathcal{S}_l$, for some $n_l \in \mathbb{N}$. Thus, $\widehat{\mathcal{S}} = \mathcal{S}_K \times \cdots \times \mathcal{S}_1$ and we write $\mathcal{R}(i_K, ..., i_1, i'_K, ..., i'_1)$, or $\mathcal{R}(\mathbf{i}, \mathbf{i}')$, if the model can transition from the *current state* $\mathbf{i}$ to a *next state* $\mathbf{i}'$ in one step (unprimed symbols denote current states, primed symbols denote next states). We let $\mathbf{x}_{(l,k)}$ denote the (sub)state $(x_l, ..., x_k)$, for $K \geq l \geq k \geq 1$. Given a function f on the domain $\widehat{\mathcal{S}}$, $Supp(f)$ denotes the set of variables in its support. Formally, $x_l \in Supp(f)$ if there are states $\mathbf{i}, \mathbf{j} \in \widehat{\mathcal{S}}$, differing only in component l, such that $f(\mathbf{i}) \neq f(\mathbf{j})$.

2.1 Symbolic encoding of state space and transition relation

State space generation consists of building the smallest set of states $\mathcal{S} \subseteq \widehat{\mathcal{S}}$ satisfying (1) $\mathcal{S} \supseteq \mathcal{S}_{init}$ and (2) $\mathcal{S} \supseteq Img(\mathcal{S})$, where the *image computation* function gives the set of successor states: $Img(\mathcal{X}) = \{\mathbf{x}' : \exists \mathbf{x} \in \mathcal{X}, (\mathbf{x}, \mathbf{x}') \in \mathcal{R}\}$. The most common symbolic approach to store the state space uses $\lceil n_l \rceil$ boolean variables to encode each state variable x_l, thus it encodes a set of states $\mathcal{Z}$ through its *characteristic function* $f_{\mathcal{Z}}$, using a BDD with $\sum_{K \geq l \geq 1} \lceil n_l \rceil$ levels.

Instead of BDDs, we prefer *ordered multi-way decision diagrams* (MDDs) [18] to encode sets of states, where each variable x_l is directly encoded in a single

level, using a node with n_l outgoing edges. Not only this results in a simpler discussion of our technique, but it also allows us to more clearly pinpoint an important property we exploit, *event locality*.

Definition 1 An MDD over $\widehat{\mathcal{S}}$ is an acyclic edge-labeled multi-graph where:

- Each node p belongs to a *level* in $\{K, ..., 1, 0\}$, denoted $p.lvl$.
- There is a single *root* node.
- Level 0 contains the only two *terminal* nodes, *Zero* and *One*.
- A node p at level $l > 0$ has n_l outgoing edges, labeled from 0 to $n_l - 1$. The edge labeled by i_l points to node q, with $p.lvl > q.lvl$; we write $p[i_l] = q$.

Finally, one of two reductions ensures canonicity. Both forbid *duplicate* nodes:

- Given nodes p and q at level l, if $p[i_l] = q[i_l]$ for all $i_l \in \mathcal{S}_l$, then $p = q$.

Then, the *fully-reduced* version [4] forbids *redundant* nodes:

- No node p at level l can exist such that, $p[i_l] = q$ for all $i_l \in \mathcal{S}_l$.

While the *quasi-reduced* version [19] forbids arcs from spanning multiple levels:

- The root is at level K.
- Given a node p at level l, $p[i_l].lvl = l - 1$ for all $i_l \in \mathcal{S}_l$. □

Definition 2 The set encoded by MDD node p at level k w.r.t. level $l \geq k$ is

$$\mathcal{B}(l,p) = \begin{cases} \mathcal{S}_l \times \mathcal{B}(l-1,p) & \text{if } l > 0 \wedge l > k \\ \bigcup_{i_l \in \mathcal{S}_l} \{i_l\} \times \mathcal{B}(l-1,p[i_l]) & \text{if } l > 0 \wedge l = k \end{cases},$$

with the convention that $\mathcal{X} \times \mathcal{B}(0, Zero) = \emptyset$ and $\mathcal{X} \times \mathcal{B}(0, One) = \mathcal{X}$. □

Most symbolic model checkers, e.g., NuSMV [9], generate the state space with breadth-first iterations, each consisting of an image computation. At the dth iteration, $\mathcal{Z}$ contains all the states at distance exactly d, or at distance up to d (either approach can be the most efficient, depending on the model). When using MDDs, we encode $\mathcal{Z}(\mathbf{x})$ as a K-level MDD and $\mathcal{R}(\mathbf{x}, \mathbf{x}')$ as a $2K$-level MDD whose unprimed and primed variables are normally interleaved for efficiency. Furthermore, the transition relation can be conjunctively partitioned into a set of *conjuncts*, $\mathcal{R}(\mathbf{x}, \mathbf{x}') = \bigwedge_\alpha \mathcal{C}_\alpha(\mathbf{x}, \mathbf{x}')$, or disjunctively partitioned into a set of *disjuncts*, $\mathcal{R}(\mathbf{x}, \mathbf{x}') = \bigvee_\alpha \mathcal{D}_\alpha(\mathbf{x}, \mathbf{x}')$ [16], stored as a set of MDDs, instead of a single monolithic MDD. Heuristically, such partitioned relations have been shown effective for synchronous and asynchronous systems, respectively.

2.2 A general partitioning methodology for the transition relation

In general discrete-state systems, both asynchronous and synchronous behavior can be present. Thus, given a model expressed in a high-level formalism, we first exploit the asynchronous aspects, by first disjunctively partitioning the transition relation $\mathcal{R}$ into a set of disjuncts, where each disjunct $\mathcal{D}_\alpha$ corresponds to a different *event* α in the set $\mathcal{E}$ of system events, i.e., $\mathcal{R}(\mathbf{x}, \mathbf{x}') \equiv \bigvee_{\alpha \in \mathcal{E}} \mathcal{D}_\alpha(\mathbf{x}, \mathbf{x}')$.

Then, each event can synchronously update several state variables. We assume that, for each disjunct $\mathcal{D}_\alpha$, the high-level model description specifies both:

- A set of *enabling conjuncts* specifying when event α can occur, or *fire*. The support of conjunct $Enable_{\alpha,m}$ is a subset of $\{x_K, ..., x_1\}$.
- A set of *updating conjuncts* describing how the state variables are updated when α fires. The support of conjunct $Upd_{\alpha,n}$ is a subset of $\{x_K, x'_K, ..., x_1, x'_1\}$.

Thus, the partitioned transition relation can be represented as:

$$\mathcal{R}(\mathbf{x}, \mathbf{x}') \equiv \bigvee_{\alpha \in \mathcal{E}} \mathcal{D}_\alpha(\mathbf{x}, \mathbf{x}') \equiv \bigvee_{\alpha \in \mathcal{E}} \left(\bigwedge_m Enable_{\alpha,m}(\mathbf{x}) \wedge \bigwedge_n Upd_{\alpha,n}(\mathbf{x}, \mathbf{x}') \right).$$

We assume a particularly important class of models, where each updating conjunct only updates one primed variables, so that we can write:

$$\mathcal{R}(\mathbf{x}, \mathbf{x}') \equiv \bigvee_{\alpha \in \mathcal{E}} \mathcal{D}_\alpha(\mathbf{x}, \mathbf{x}') \equiv \bigvee_{\alpha \in \mathcal{E}} \left(\bigwedge_m Enable_{\alpha,m}(\mathbf{x}) \wedge \bigwedge_{K \geq l \geq 1} Upd_{\alpha,l}(\mathbf{x}, x'_l) \right). \quad (1)$$

As a running example, we consider an event α corresponding to the following pseudocode statement in a larger program:

if $x_5 > 2$ and $x_6 \leq 1$ then $\langle x_3, x_6 \rangle \leftarrow \langle x_4, (x_7 + x_6) \bmod 6 \rangle$;

where the state variables are $x_7, ..., x_1 \in [0..5]$ and the "$\langle\ \rangle$" pairs enclose v (two, in our case) distinct variables to be simultaneously assigned, and the corresponding v expressions, which are evaluated before performing any assignment. The disjunct $\mathcal{D}_\alpha$ has then two enabling conjuncts, $Enable_{\alpha,1} \equiv [x_5 > 2]$ and $Enable_{\alpha,2} \equiv [x_6 \leq 1]$, and seven updating conjuncts, one for each variable x_k, $k \in [7, ..., 1]$, $Upd_{\alpha,3} \equiv [x'_3 = x_4]$, $Upd_{\alpha,6} \equiv [x'_6 = (x_7 + x_6) \bmod 6]$, and $Upd_{\alpha,k} \equiv [x'_k = x_k]$, for $k \in \{7, 5, 4, 2, 1\}$.

2.3 Event locality

We now examine the ways an event α can be "independent" of a state variable and show how the standard concept of support for a function is inadequate when applied to the disjuncts of the transition relation. Recalling that $\mathcal{D}_\alpha$ is just a function of the form $\widehat{\mathcal{S}} \times \widehat{\mathcal{S}} \to \mathbb{B}$, we can consider the following cases:

- If $x_l \notin Supp(\mathcal{D}_\alpha)$, the value of x_l affects neither the enabling of event α nor the value of any x'_k, for $K \geq k \geq 1$, including $k = l$, when α fires. In our running example, this is the case for x_3.
- If $x'_l \notin Supp(\mathcal{D}_\alpha)$, the value of x'_l is independent of that of x_k, for $K \geq k \geq 1$, including $k = l$, when α fires. This corresponds to nondeterministically setting x'_l to any value in $\mathcal{S}_l$. Of course, given the expression of Eq. 1, we already know that x'_l affects neither the enabling of α nor the values of x'_k, for $k \neq l$.

When encoding $\mathcal{D}_\alpha$ with a fully-reduced $2K$-level MDD, the above two cases are reflected in the absence of node at level l, or l', respectively. Indeed, it is even possible that both $x_l \notin Supp(\mathcal{D}_\alpha)$ and $x'_l \notin Supp(\mathcal{D}_\alpha)$ hold, thus neither l nor l' would contain any node. However, these two cases are neither as important nor as common as the following type of "independence":

– If $x_l \notin Supp(\bigwedge_m Enable_{\alpha,m} \wedge \bigwedge_{k \neq l} Upd_{\alpha,k})$ and $Upd_{\alpha,l} \equiv [x'_l = x_l]$, the value of x_l affects neither the enabling of event α nor the value of any x'_k, for $k \neq l$, while the firing of α does not change the value of x_l.

This common situation is not exploited by ordinary MDD reductions; rather, it results in the presence of (possibly many) *identity patterns*: a node p at level l such that, foreach $i_l \in \mathcal{S}_l$, $p[i_l] = q_{i_l}$, and $q_{i_l}[j_l] = Zero$ for all $j_l \in \mathcal{S}_l$, except for $q_{i_l}[i_l] = r$, where node $r \neq Zero$ does not depend on i_l (the gray pattern in Fig. 1). It is instead exploited by Kronecker encodings of transition matrices [6, 8].

We define $\mathcal{V}_\alpha^{indep}$ to be the set containing any such (unprimed) variable, and $\mathcal{V}_\alpha^{dep} = \{x_K, ..., x_1\} \setminus \mathcal{V}_\alpha^{indep}$, and say that *event locality* is present in a system when $\mathcal{V}_\alpha^{dep}$ is a strict subset of $\{x_K, ..., x_1\}$. We further split $\mathcal{V}_\alpha^{dep}$ into $\mathcal{V}_\alpha^{upd} = \{x_l : Upd_{\alpha,l} \not\equiv [x'_l = x_l]\}$, the set of unprimed variables whose corresponding primed variable can be *updated* by the firing of α, and $\mathcal{V}_\alpha^{unchanged} = \mathcal{V}_\alpha^{dep} \setminus \mathcal{V}_\alpha^{upd}$, the set of unprimed variables that affect the enabling of α or the value of some primed variable, but whose corresponding primed variable is *not updated* by the firing of α. For our running example, $\mathcal{V}_\alpha^{upd} = \{x_6, x_3\}$, $\mathcal{V}_\alpha^{unchanged} = \{x_7, x_5, x_4\}$ and $\mathcal{V}_\alpha^{indep} = \{x_2, x_1\}$. By definition, $\mathcal{V}_\alpha^{upd} \cup \mathcal{V}_\alpha^{unchanged} \cup \mathcal{V}_\alpha^{indep} = \{x_K, ..., x_1\}$ and, based on these sets, we can partition the transition relation as:

$$\mathcal{R}(\mathbf{x}, \mathbf{x}') \equiv \bigvee_{\alpha \in \mathcal{E}} \left(\bigwedge_m Enable_{\alpha,m}(\mathbf{x}) \wedge \bigwedge_{x_k \in \mathcal{V}_\alpha^{upd}} Upd_{\alpha,k}(\mathbf{x}, x'_k) \wedge \bigwedge_{x_k \notin \mathcal{V}_\alpha^{upd}} [x'_k = x_k] \right).$$

Definition 3 Let $Top(\alpha)$ and $Bot(\alpha)$ be the highest and lowest variable indices in $\mathcal{V}_\alpha^{dep}$: $Top(\alpha) = \max\{k : x_k \in \mathcal{V}_\alpha^{dep}\}$, $Bot(\alpha) = \min\{k : x_k \in \mathcal{V}_\alpha^{dep}\}$. □

We can then group the disjuncts $\mathcal{D}_\alpha$ according to the value of $Top(\alpha)$:

$$\mathcal{R}(\mathbf{x}, \mathbf{x}') \equiv \bigvee_{K \geq k \geq 1} \mathcal{R}_k(\mathbf{x}, \mathbf{x}') \equiv \bigvee_{K \geq k \geq 1} \left(\bigvee_{\alpha : Top(\alpha) = k} \mathcal{D}_\alpha(\mathbf{x}, \mathbf{x}') \right). \tag{2}$$

2.4 Kronecker encoding of the transition relation to exploit locality

The performance evaluation community working on Markov chains has long recognized that Kronecker techniques can be used to encode large (real) transition matrices while naturally exploiting the presence of identity transformations for state variables [25]. However, such an encoding requires a *Kronecker consistent* model. In our setting, this means that the following two properties must hold:

– The support of each enabling conjunct contains only one unprimed variable. Thus, $Enable_{\alpha,l}(x_l)$ simply lists the set of values $\mathcal{S}_{\alpha,l} \subseteq \mathcal{S}_l$ for x_l in which the event may be enabled: $Enable_{\alpha,l}(x_l) \equiv [x_l \in \mathcal{S}_{\alpha,l}]$. Of course, when $\mathcal{S}_{\alpha,l} = \mathcal{S}_l$, the conjunct does not enforce any restriction on the enabling of event α.
– The support of the updating conjunct for x'_l contains only x_l, in addition to x'_l: $Upd_{\alpha,l}(x_l, x'_l) \equiv [x'_l \in \mathcal{N}_{\alpha,l}(x_l)]$, where $\mathcal{N}_{\alpha,l}(x_l) \subseteq \mathcal{S}_l$.

Such a model is called Kronecker consistent because, letting $\mathcal{R}_{\alpha,l}(x_l, x'_l) \equiv [x_l \in \mathcal{S}_{\alpha,l} \wedge x'_l \in \mathcal{N}_{\alpha,l}(x_l)]$, and storing it as an $n_l \times n_l$ boolean matrix $\mathbf{R}_{\alpha,l}$, we can write $\mathcal{R}(\mathbf{x}, \mathbf{x}') \equiv \bigvee_{\alpha\in\mathcal{E}} \bigwedge_{K\geq l\geq 1} \mathcal{R}_{\alpha,l}(x_l, x'_l)$ and the matrix $\mathbf{R}$ corresponding to the overall transition relation $\mathcal{R}$ can be expressed as $\mathbf{R} = \sum_{\alpha\in\mathcal{E}} \bigotimes_{K\geq l\geq 1} \mathbf{R}_{\alpha,l}$, where "$\sum$" indicates boolean sum and "$\otimes$" Kronecker product of matrices. Our notion of locality becomes apparent: for $x_l \in \mathcal{V}^{indep}_{\alpha}$, $\mathcal{R}_{\alpha,l}(x_l, x'_l) \equiv [x'_l = x_l]$, thus $\mathbf{R}_{\alpha,l}$ is an identity matrix, which of course is not explicitly stored.

A model can be made Kronecker consistency in two ways. We can *merge* state variables into new "larger" variables, so that each new variable can depend only on the original state variables that were merged into it; in our running example, we could merge variables x_4 and x_3 into a new variable, and variables x_7 and x_6 into another new variable. Or we can *split* a disjunct $\mathcal{D}_\alpha$ based on a Shannon expansion, so that each new "smaller" disjunct satisfies Kronecker consistency; in our example, we can split along variables x_7 and x_4 and write $\mathcal{D}_\alpha = \bigvee_{i_7\in\{0,\ldots,5\},i_4\in\{0,\ldots,5\}} \mathcal{D}_{\alpha,x_7=i_7,x_4=i_4}$, where each $\mathcal{D}_{\alpha,x_7=i_7,x_4=i_4}$ satisfies Kronecker consistency. However, neither approach is satisfactory for models with intricate dependencies; excessive merging results in few or even just a single state variable, i.e., an explicit approach, while excessive splitting causes an exponential growth in the number of events, i.e., the storage for the $\mathbf{R}_{\alpha,l}$ matrices.

2.5 Previously proposed variants of the saturation algorithm

The saturation algorithm exploits event locality through lightweight recursive fixed point image computations where the disjunctive partitioning of the transition relation is organized according to the value of $Top(\alpha)$, for each $\alpha \in \mathcal{E}$:

$$\mathcal{R}(\mathbf{x}, \mathbf{x}') \equiv \bigvee_{K\geq l\geq 1} \bigvee_{\alpha:Top(\alpha)=l} \mathcal{D}_\alpha(\mathbf{x}, \mathbf{x}') \equiv \bigvee_{K\geq l\geq 1} \mathcal{R}_l(\mathbf{x}_{(l,1)}, \mathbf{x}'_{(l,1)}) \wedge [\mathbf{x}'_{(K,l+1)} = \mathbf{x}_{(K,l+1)}].$$

A node p at level l is saturated if the set of states it encodes cannot be enlarged by applying events α such that $Top(\alpha) \leq l$, i.e., $\mathcal{B}(l,p) \supseteq Img_{\leq l}(\mathcal{B}(l,p))$, where $Img_{\leq l}$ is the image computation restricted to $\mathcal{R}_k$, for $l \geq k \geq 1$:

$$Img_{\leq l}(\mathcal{X}) = \{\mathbf{x}'_{(l,1)} : \exists \mathbf{x}_{(l,1)} \in \mathcal{X}, \exists k \leq l, \mathcal{R}_k(\mathbf{x}_{(k,1)}, \mathbf{x}'_{(k,1)}) \wedge \mathbf{x}_{(l,k+1)} = \mathbf{x}'_{(l,k+1)}\}.$$

Thus, if, before saturating it, p encoded the set $\mathcal{B}(l,p) = \mathcal{X}_0$, at the end of its saturation, p encodes the least fixed point $\mathcal{B}(l,p) = \mu\mathcal{X}.(\mathcal{X}_0 \cup Img_{\leq l}(\mathcal{X}))$. To encode this fixed point, p is modified *in place*, i.e., the pointers $p[i_l]$ are changed, so that they point to nodes that encode increasingly larger sets, while the pointers to p from nodes above it are unchanged.

Starting from the MDD encoding the initial state(s), the nodes of this MDD are *saturated* in bottom-up order. In other words, whenever the application of $\mathcal{R}_l$ causes the creation of a node at a level $k < l$, this new node must be immediately saturated, by applying $\mathcal{R}_k$ to it. Thus, during the bottom-up process of saturating all nodes, only $\mathcal{R}_l$ must be applied to a node p at level l, since all $\mathcal{R}_k$, for $k < l$, have been already applied to saturate its children.

In the original saturation algorithm for Kronecker-consistent models [6], we store each $\mathcal{D}_\alpha$ as the set of matrices $\mathbf{R}_{\alpha,l}$, for $Top(\alpha) \geq l \geq Bot(\alpha)$; of course,

for $Top(\alpha) > l > Bot(\alpha)$, we might have $x_l \in \mathcal{V}_\alpha^{indep}$, in which case $\mathbf{R}_{\alpha,l}$ is the identity and is not stored. For state-space generation of GALS, we showed how the peak memory and runtime requirements of saturation can be several orders of magnitude better than a traditional breadth-first iteration.

In [7], we extended the algorithm to models where the state variables have unknown bounds, which must then be discovered "on-the-fly". During the generation process, each matrix $\mathbf{R}_{\alpha,l}$ contains rows and columns corresponding to the *confirmed* values for $\mathbf{x}_l$, i.e., values i_l that appear as the lth component in at least one global state $\mathbf{i}$ known to be reachable, but also columns corresponding to *unconfirmed* values for $\mathbf{x}_l$, i.e., values j_l such that $\mathcal{R}_{\alpha,l}(i_l, j_l)$ is a possible transition in isolation from a confirmed state i_l, but we don't yet know whether α is enabled in a global state $\mathbf{i}$ whose l^{th} component is i_l. Thus, the algorithm interleaves the building of rows of $\mathbf{R}_{\alpha,l}$, obtained through an explicit (local) state-space exploration of the model restricted to variable x_l, with the (global) symbolic exploration of the state space.

In [20], Miner showed how to deal with models that do not satisfy the Kronecker-consistency requirement. The transition relation is encoded using K-level *matrix diagrams (MxDs)*, which we introduced in [8]. Essentially, these are $2K$-level MDDs where the nodes of levels x_l and x'_l are merged into "matrix" nodes having $n_l \times n_l$ edges, but, unlike ordinary decision diagrams, the reduction rule requires to remove a node p if it describes an identity, i.e., if $p[i_l, j_l] = Zero$ for $i_l \neq j_l$ and $p[i_l, i_l] = q$ for all $i_l \in \mathcal{S}_l$. Thus, MxDs combine the generality of decision diagrams (they can represent any relation over $\widehat{\mathcal{S}}$) with the advantages of a Kronecker representation (they can reveal and exploit event locality).

A single MxD can encode $\mathcal{R}_l$, but [20] requires $\mathcal{R}_l(\mathbf{x}, \mathbf{x}')$ to be expressed as

$$\mathcal{R}_l(\mathbf{x}_{(l,1)}, \mathbf{x}'_{(l,1)}) \equiv \bigvee_{\alpha:\, Top(\alpha)=l} \left(\bigwedge_g Group_{\alpha,g}(\mathbf{x}_{(l,1)}, \mathbf{x}'_{(l,1)}) \wedge \bigwedge_{x_k \in \mathcal{V}_\alpha^{indep}, k<l} [x'_k = x_k] \right),$$

where $Supp(Group_{\alpha,g})$ is a set of "unprimed-primed" variable pairs, and groups have disjoint supports: $g \neq h$ implies $Supp(Group_{\alpha,g}) \wedge Supp(Group_{\alpha,h}) = \emptyset$. Thus, each "coarse-grain" $Group_{\alpha,g}$ corresponds to the intersection of all the enabling and updating conjuncts that depend on its support, including the updating conjuncts of the form $[x'_k = x_k]$, for $x_k \in \mathcal{V}_\alpha^{unchanged} \cap Supp(Group_{\alpha,g})$.

Then, [20] maintains an MxD for each $Group_{\alpha,g}$, and updates it every time a new local state $i_k \in \mathcal{S}_k$ is confirmed, if $x_k \in Supp(Group_{\alpha,g})$. In turn, this triggers the rebuilding of the MxD for $\mathcal{R}_l$, by performing the appropriate MxD intersection and union operations. Just like in [7], these updates following the confirmation of a local state require one step of explicit state space exploration in a portion of the model. However, instead of considering a single variable x_k, we must now consider **all** the unprimed variables in $Supp(Group_{\alpha,g})$ whenever the set of possible values for **any** of these variables is extended. For example, if $Supp(Group_{\alpha,g}) = \{x_7, x'_7, x_5, x'_5, x_2, x'_2\}$ and $i_5 \in \mathcal{S}_5$ is confirmed, [20] explicitly explores the possible transitions from each state in $\mathcal{S}_7 \times \{i_5\} \times \mathcal{S}_2$.

3 Fine-grain partitioning with MDD-based encodings

The cost of building the coarse-grain disjoint partitioned groups $Group_{\alpha,g}$ of [20] can be large, since each group $Group_{\alpha,g}$ is built explicitly, at an exploration cost $O(\prod_{x_k \in Supp(Group_{\alpha,g})} |\mathcal{S}_k|)$. The disjoint partitioning requirement may result in too coarse a conjunctive-partitioning for event α or even, in the worst case, in a single group, as in the shift register example of Sect. 5.

We propose a fine-grain partitioned approach, using the more familiar $2K$-level MDDs to encode the transition relation. We express $\mathcal{R}_l(\mathbf{x}_{(l,1)}, \mathbf{x}'_{(l,1)})$ as:

$$\mathcal{R}_l(\mathbf{x}_{(l,1)}, \mathbf{x}'_{(l,1)}) = \bigvee_{\alpha: Top(\alpha)=l} \left(\mathcal{D}_\alpha^{Part}\left(\mathbf{x}_{(l,1)}, \mathbf{x}'_{(l,1)}\right) \wedge \bigwedge_{x_k \notin \mathcal{V}_\alpha^{upd}, k \leq l} [x'_k = x_k] \right), \quad (3)$$

where the "partial" relation $\mathcal{D}_\alpha^{Part}\left(\mathbf{x}_{(l,1)}, \mathbf{x}'_{(l,1)}\right)$ is defined as:

$$\mathcal{D}_\alpha^{Part}\left(\mathbf{x}_{(l,1)}, \mathbf{x}'_{(l,1)}\right) = \bigwedge_m Enable_{\alpha,m}\left(\mathbf{x}_{(l,1)}\right) \wedge \bigwedge_{x_k \in \mathcal{V}_\alpha^{upd}} Upd_{\alpha,k}\left(\mathbf{x}_{(l,1)}, x'_k\right).$$

We use a fully-reduced MDD for each enabling conjunct $Enable_{\alpha,m}$ and each updating conjunct $Upd_{\alpha,k}$ of each event α with $Top(\alpha) = l$, where $x_k \in \mathcal{V}_\alpha^{upd}$. The variables of each such MDD are only those in the support of the encoded conjunct; because of our new encoding technique, we do not store the updating conjuncts of the form $[x'_k = x_k]$, for $x_k \notin \mathcal{V}_\alpha^{upd}, k \leq l$, even if $x_k \notin \mathcal{V}_\alpha^{indep}$.

The fully-reduced MDD encoding $\mathcal{D}_\alpha^{Part}$ is then obtained as the intersection (boolean conjunction) of the MDDs of all its $Enable_{\alpha,m}$ and $Upd_{\alpha,k}$ conjuncts, thus $Supp(\mathcal{D}_\alpha^{Part})$ is the union of the supports of these conjuncts.

3.1 Fully-identity reduced $2K$-level MDD encoding of the disjuncts

To efficiently build $\mathcal{R}_l$ from Eq. 3 and exploit event locality in the MDD, we introduce a new canonicity-preserving *identity* reduction rule. In our particular application, we use a *fully-identity* reduced $2K$-level MDD to encode each $\mathcal{R}_l$:

- Each unprimed level, l (for variable x_l) $K \geq l \geq 1$, is fully-reduced, i.e., no node at level l can be redundant, and is immediately followed by the corresponding primed level l' (for variable x'_l).
- Each level l' is *identity-reduced* w.r.t. to level l: (1) if node p is at level l and $p[i_l]$ reaches q at level l', then q is not a *singular*-i_l node, i.e., it is not a node with $q[i_l] \neq Zero$ and $q[j_l] = Zero$ for all $j_l \in \mathcal{S}_l \setminus \{i_l\}$; (2) a singular-$i_l$ node at level l', for any $i_l \in \mathcal{S}_l$, must be pointed by a node at level l.

Fig. 1 shows three examples of MDDs that are either fully-fully (left) or fully-identity (right) reduced for the unprimed and primed levels, respectively. In the first example, the entire identity pattern clearly visible in the fully-fully case is absent in the fully-identity case, because nodes q_0 and q_1 are eliminated first (due to the identity-reduced rule for level l') and the remaining node p is now redundant and eliminated (due to the fully-reduced rule for level l). In the second example, singular-0 node q_0 is eliminated, but redundant node q is added

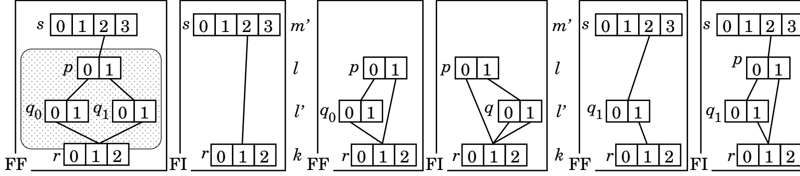
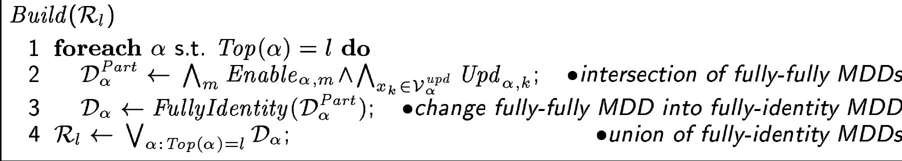


Fig. 1: Comparing fully-fully (FF) and fully-identity (FI) reductions for MDDs.



Build($\mathcal{R}_l$)
1 **foreach** α s.t. $Top(\alpha) = l$ **do**
2 $\quad \mathcal{D}_\alpha^{Part} \leftarrow \bigwedge_m Enable_{\alpha,m} \wedge \bigwedge_{x_k \in \mathcal{V}_\alpha^{upd}} Upd_{\alpha,k}$; •*intersection of fully-fully MDDs*
3 $\quad \mathcal{D}_\alpha \leftarrow FullyIdentity(\mathcal{D}_\alpha^{Part})$; •*change fully-fully MDD into fully-identity MDD*
4 $\mathcal{R}_l \leftarrow \bigvee_{\alpha:Top(\alpha)=l} \mathcal{D}_\alpha$; •*union of fully-identity MDDs*

Fig. 2: Algorithm to build $\mathcal{R}_l$ from the disjuncts and conjuncts.

while, in the third example, singular-1 node q_1 requires the introduction of node p, which is not redundant in the fully-identity reduction case.

From the fully-fully reduced MDD for $\mathcal{D}_\alpha^{Part}$, the fully-identity reduced $2K$-level MDD for $\mathcal{D}_\alpha$ is built using a recursive procedure. Then, the $2K$-level fully-identity reduced MDD encoding of $\mathcal{R}_l$ is built using a recursive *union* for fully-identity reduced MDD on the disjuncts $\mathcal{D}_\alpha$ for which $Top(\alpha) = l$ (Fig. 2).

Our fully-identity reduced $2K$-level MDDs, while strongly related to MxDs used in [20] to encode disjuncts, can be even more compact. Only matrix nodes corresponding to levels $l \in \mathcal{V}_\alpha^{indep}$ can be eliminated in MxDs, while, in addition to eliminating these entire identity patterns, our fully-identity reduced MDDs also eliminate nodes at primed levels $k \in \mathcal{V}_\alpha^{unchanged}$.

4 A new saturation algorithm for state-space generation

Based on the new encoding technique for partitioned transition relations, we propose new saturation algorithms for models with known or unknown variable bounds, respectively (Fig. 3 and 4). As in previous saturation algorithms, the state space is encoded in a *quasi-reduced* K-level MDD, since $\mathbf{R}_l$ must be applied also to redundant nodes at level l; if fully-reduced MDDs were used, these nodes would have to be re-inserted to saturate them during bottom-up saturation. For simplicity, the pseudocode of Fig. 3 4 assumes that either both or neither levels k and k' are skipped in the MDD of the disjuncts $\mathcal{R}_l$, for $K \geq l \geq 1$; our actual implementation also manages the case when only one of them is skipped.

4.1 Saturation when state variables have known bounds

For models where state variables have known bounds, e.g., circuits and other hardware models, each conjunct can be built separately a priori, considering all the possible transitions when the conjunct is considered in isolation. Then, the MDD encodings of the disjunctive partition $\mathcal{R}_K, ..., \mathcal{R}_1$, can be built by the *Build*($\mathcal{R}_l$) procedure of Fig. 2, prior to state-space generation. The saturation algorithm for the case when $\mathcal{R}_K, ..., \mathcal{R}_1$ are built this way is shown in Fig. 3.

```
Saturate(MDD p)
1 l ← p.lvl;
2 repeat
3    B(l,p) ← B(l,p) ∪ ⋃_{i_l ∈ S_l, i'_l ∈ S_l} {i'_l} × ImgSat(p[i_l], R_l[i_l][i'_l])
4 until B(l,p) is not changed
```

```
ImgSat(MDD q, MDD2 f)
1 if q = Zero or f = Zero then return ∅;
2 k ← q.lvl;   m ← f.lvl;   s ← a new MDD node s at level k;
3 if k > m then
4    B(k,s) = ⋃_{i_k ∈ S_k} {i_k} × ImgSat(q[i_k], f);
5 else                                                    •k is equal to m
6    B(k,s) = ⋃_{i_k ∈ S_k, i'_k ∈ S_k} {i'_k} × ImgSat(q[i'_k], f[i_k][i'_k]);
7 Saturate(s);
8 return B(k,s).
```

Fig. 3: A saturation algorithm for models with known variable bounds.

Saturate(p) recursively compute a fixed-point on node p at level l. It iteratively selects a child $p[i_l]$; for each $i'_l \in \mathcal{S}_l$, it calls *ImgSat*$(p[i_l], \mathcal{R}_l[i_l][i'_l])$ to compute the (possibly new) reachable states in $\mathcal{S}_{l-1} \times ... \times \mathcal{S}_1$; finally, it adds the states $\{i'_l\} \times$ *ImgSat*$(p[i_l], \mathcal{R}_l[i_l, i'_l])$, a subset of $\mathcal{S}_l \times ... \times \mathcal{S}_1$, to $\mathcal{B}(l, p)$.

The *ImgSat*(q, f) procedure takes a K-level MDD node q at level k and a $2K$-level MDD node f at level m as inputs, where $m \leq k$, since the K-level MDD for the state space is quasi-reduced. If either node q or node f is $Zero$, the empty set is returned, since no transitions are possible in this case. If node f is at a level m below k, our fully-identity reduction implies the identity-transformation of variable x_k. If node f is instead at level k, for each possible transition of variable x_k from i_k to i'_k, a recursive call on the children nodes is made. These (possibly) new states are encoded in a MDD node s at level k, which is *Saturated* prior to returning it to the calling procedure.

The state-space generation starts with an MDD encoding for the initial state(s), then follows with a bottom-up saturation of these initial MDD nodes, until all of them are saturated. The final result is the encoding of the state space.

4.2 Saturation when state variables have unknown bounds

For systems with unknown variable bounds, the partitioned transition relations cannot be built prior to state-space generation. We must instead interleave building partitioned transition relation i.e., calls to the *Confirm* procedure, with symbolic state-space generation (Fig. 4).

We define the *confirmed* set $\mathcal{S}^c_l \subseteq \mathcal{S}_l$ for variable x_l as the values of variable x_l that appear in a global state currently encoded by the MDD. $\mathcal{S}_l \backslash \mathcal{S}^c_l$ contains instead the *unconfirmed* states that appear only in the l' level of the transition relation; these are "locally" but not necessarily "globally" reachable. For any node p at level l, $p[i_l] = Zero$ for any such unconfirmed local state i_l.

Saturate is then modified so that, at each iteration, any new values for x_l that are now known to be reachable (appears in a path leading to *One* in the MDD encoding the state-space) are confirmed by calling *Confirm*(x_l, i'_l), and $\mathcal{R}_l$

```
Saturate(MDD p)
1 l ← p.lvl;
2 repeat
3     Confirm(x_l, i_l) for any state i_l ∈ S_l \ S_l^c s.t. p[i_l] ≠ Zero;
4     Build(R_l);
5     pick i_l ∈ S_l^c, i'_l ∈ S_l s.t. R_l[i_l][i'_l] ≠ Zero;
6     B(l, p) ← B(l, p) ∪ {i'_l} × ImgSat (p[i_l], R_l[i_l][i'_l])
7 until B(l, p) is not changed
```

```
Confirm(x_l, i_l)
1 S_l^c ← S_l^c ∪ {i_l};
2 foreach enabling conjunct Enable_{α,m}, s.t. x_l ∈ Supp(Enable_{α,m}) do
3     foreach i_sub ∈ {i_l} × X_{x_k ∈ Supp(Enable_{α,m}) \ {x_l}} S_k^c do
4         if ModelEnable_{α,m}(i_sub) then Enable_{α,m} ← Enable_{α,m} ∪ {i_sub};
5 foreach updating conjunct Upd_{α,n}, s.t. x_l ∈ Supp(Upd_{α,n}) do
6     foreach i_sub ∈ {i_l} × X_{x_k ∈ {x_K,...,x_1} ∩ Supp(Upd_{α,n}) \ {x_l}} S_k^c do
7         I'_n ← ModelUpd_{α,n}(i_sub);                • states reachable from i_sub in one step
8         Upd_{α,n} ← Upd_{α,n} ∪ {i_sub} × I'_n;
9         S_n ← S_n ∪ I'_n;
```

Fig. 4: A saturation algorithm for models with unknown variable bounds.

is rebuilt if needed, i.e., if any of its conjuncts has changed. The selection of (i_l, i'_l) in statement 5 of course avoids repeating a pair unless $p[i_l]$ or $\mathcal{R}_l[i_l][i'_l]$ have changed; this check is omitted for clarity.

Procedure *Confirm* takes a new value i_l for variable x_l and updates each conjunct with x_l in its support. This requires to explicitly query the high-level model for each (sub)state $\mathbf{i}_{sub}$ that can be formed using the new value i_l and any of the values in the confirmed set $\mathcal{S}_k^c$ of any unprimed variable x_k in the support of such a conjunct. Functions $ModelEnable_{\alpha,m}$ and $ModelUpd_{\alpha,n}$ are analogous to $Enable_{\alpha,m}$ and $Upd_{\alpha,n}$, but they are assumed to be "black-boxes" that can be queried only explicitly; they return, respectively, whether α is enabled in $\mathbf{i}_{sub}$, and what is the set of possible values for x'_n when α fires in $\mathbf{i}_{sub}$ in isolation, i.e., considering only the restriction of the model to the particular conjunct. Thus, our cost is still of the form $O(\prod_{x_k \in Supp(f)} |\mathcal{S}_k|)$, as in [20], but f is now $Enable_{\alpha,m}$ or $Upd_{\alpha,n}$, which can have a much smaller support than $Group_{\alpha,g}$.

We stress that, while our presentation assumes that the support of each updating conjunct contains a single primed variable, this is is not required by our approach. Thus, a situation where an event α "nondeterministically either increments both x_5 and x_4 or decrements both x_5 and x_4" is captured with an updating conjunct of the form $Upd_{\alpha,\{4,5\}}$ having both x'_5 and x'_4 in its support. Indeed, our implementation heuristically merges enabling or updating conjuncts for efficiency reasons: if $Supp(Enable_{\alpha,m}) \subset Supp(Upd_{\alpha,l})$, we can merge the effect of $Enable_{\alpha,m}$ in the definition of $Upd_{\alpha,l}$; if $Supp(Upd_{\alpha,l}) \setminus \{x'_l\} \subseteq Supp(Upd_{\alpha,k}) \setminus \{x'_k\}$, we can merge the two updating conjuncts, into a conjunct $Upd_{\alpha,\{l,k\}}$ having the union of the supports. As long as no new unprimed variable is added to the support of a conjunct, the enumeration cost of explicitly building the conjunct is not affected, but the number of conjuncts is reduced.

5 Experimental results

We now show some experimental results for our new technique, run on a 3 Ghz Pentium IV workstation with 1GB memory, on a set of models whose state-space size can be controlled by a parameter N. We compare the new proposed saturation algorithm for the unknown bound case (for ease of model specification) with the saturation algorithm in [20], the symbolic model checker NuSMV [9], and the explicit model checker SPIN [14], which targets asynchronous software verification and applies partial-order reduction and other optimizations such as minimized automaton storage for reachable states and hash compaction.

Table 1 reports the parameter N, the size $|\mathcal{S}|$ of the original and the reduced (by partial-order reduction) state space, and the runtimes and peak memory requirements for the four approaches, SPIN, SMV (NuSMV), QEST (the approach in [20]), and NEW (our new encoding technique). Both QEST and NEW are implemented in our tool SMART [5]. We studied the following models:

- A *Slotted-ring* network protocol [24] where N processors can access the medium only during the slot allocated to them.
- The classical *Queen* problem where the N queens must be placed on an $N \times N$ chessboard, in sequential order from row 1 to N without attacking each other.
- A *Fault-tolerant* multiprocessor system described in [11]. While [11] requires 1200 seconds for $N = 5$, we require 0.07 seconds for $N = 5$ and we can generate the state space for $N = 100$ in 485 seconds. [20] reports similar improvements, but uses over twice as much memory.
- A *Leader* election protocol among N processes, a simplified version of [12], where the broadcasting of the winner identity is omitted. Messages are sent asynchronously via FIFO message queues.
- A *Bubble-sort* algorithm where an array of N numbers initialized to $N, ..., 1$ need to be sorted into the result $1, ..., N$.
- A *Swapper* program [1] where a boolean array of size $2N$ is initialized with 0's in the first half and 1's in the second half, and the two halves are swapped through neighbor-only exchanges. While the best tool considered in [1] requires 90 seconds for $N = 40$, we require 0.03 seconds for $N = 40$ and can generate the state space for $N = 2,000$ in 50 seconds.
- A *Round-robin* mutex protocol [13] where N processes require exclusive access to a resource.
- A *Bit-shifter* where, at each step, a new random bit b_0 is generated, and bit b_k is set to bit b_{k-1}, for $N \geq k \geq 1$.
- An analogous *Int-shifter*, which shifts values in the range $\{1, ..., N\}$.

Defining equivalent models (with the same number of states) was a challenge. For *Leader*, from the SPIN distribution, we were able to define equivalent models for NuSMV and SMART. For all other models, initially defined in SMART, we defined equivalent NuSMV and SPIN models, except that, for SPIN, our *Queen* model has approximately 1/3 more states, and we have no *Fault-tolerant* model.

From Table 1, we can observe that, compared with QEST, NEW has better runtime and memory consumptions for essentially all models and parameter combinations. In the only two cases where QEST is (negligibly) faster than NEW, *Bubble-sort* and *Fault-tolerant*, NEW's memory consumption is much better.

Table 1: Experimental results.

N	Original $\|\mathcal{S}\|$	Reduced $\|\mathcal{S}\|$	CPU time (sec)				Peak memory (KB)			
			SPIN	SMV	QEST	NEW	SPIN	SMV	QEST	NEW
Slotted-ring: $K = N$, $\|\mathcal{S}_k\| = 15$ for all k										
6	575,296	575,296	8.2	0.13	0.03	0.03	401,130.5	5,660.6	33.3	36.8
15	1.5×10^{15}	—	—	1285.1	0.17	0.15	—	21,564.2	262.7	155.8
200	8.4×10^{211}	—	—	—	204.8	153.6	—	—	362,175.8	176,934.1
Queen: $K = N$, $\|\mathcal{S}_k\| = N + 1$ for all k										
11	166,926	228,004	0.7	14.0	3.9	1.8	21,308.4	17,018.4	14,078.6	5,302.2
12	856,189	1.2×10^{6}	3.8	105.9	19.5	8.9	103,307.3	53,218.3	63,828.6	23,276.7
Fault-tolerant: $K = 10N + 1$, $\|\mathcal{S}_k\| \leq 4$ for all k except $\|\mathcal{S}_1\| = N + 1$										
5	2.4×10^{13}	n/a	n/a	152.0	0.14	0.07	n/a	18,576.8	171.4	120.2
100	1.0×10^{262}	n/a	n/a	—	480.2	485.0	n/a	—	52,438.7	23,406.1
Leader: $K = 2N$, $\|\mathcal{S}_k\| \leq 19$ for all k										
7	1.5×10^{6}	70	0.013	491.7	1.6	0.7	7,521.3	91,270.2	607.2	42.1
20	3.3×10^{17}	200	0.024	—	93.4	28.2	7,521.3	—	7,281.3	86.6
30	1.8×10^{26}	300	0.047	—	568.9	145.6	9,618.4	—	20,271.7	114.3
Bubble-sort: $\|\mathcal{S}\| = N!$, $K = N$, $\|\mathcal{S}_k\| = N$ for all k										
11	4.0×10^{7}	4.0×10^{7}	1,042.8	125.3	1.7	1.7	848,498.7	19,704.4	7,217.6	2,505.7
12	4.8×10^{8}	—	—	859.9	5.3	5.5	—	43,948.3	21,680.7	7,438.5
Swapper: $\|\mathcal{S}\| = N!/((N/2)!)^2$, $K = N$, $\|\mathcal{S}_k\| = 2$ for all k										
20	184,756	184,756	0.9	0.2	0.01	0.01	37,006.3	6,831.6	30.8	23.5
40	1.4×10^{11}	—	—	241.6	0.06	0.03	—	14,795.2	100.4	34.3
2,000	2.0×10^{600}	—	—	—	742.3	49.8	—	—	187,266.0	58,970.9
Round-robin: $K = N + 1$, $\|\mathcal{S}_k\| = 10$ for all k except $\|\mathcal{S}_K\| = N + 1$										
16	2.3×10^{6}	2.3×10^{6}	43.0	0.34	0.11	0.07	1,046,004.7	7,060.3	290.0	125.5
50	1.3×10^{17}	—	—	11.7	2.1	1.2	—	61,239.0	6,041.3	1,299.2
200	7.2×10^{62}	—	—	—	336.3	77.0	—	—	351,625.7	47,833.1
Bit-shifter: $\|\mathcal{S}\| = 2^{N+2}$, $K = N + 3$, $\|\mathcal{S}_k\| = 2$ for all k except $\|\mathcal{S}_{N+3}\| = \|\mathcal{S}_{N+2}\| = 3$										
16	262,144	262,144	1.2	0.03	12.7	0.01	306,798.6	4,449.4	8,326.6	44.2
256	4.6×10^{77}	—	—	447.7	—	2.63	—	16,723.4	—	4,988.2
1,000	4.3×10^{301}	—	—	—	—	64.52	—	—	—	97,060.2
Int-shifter: $\|\mathcal{S}\| = 2N^{N+1}$, $K = N + 3$, $\|\mathcal{S}_k\| = N$ for all k except $\|\mathcal{S}_{N+3}\| = \|\mathcal{S}_{N+2}\| = 3$										
7	1.2×10^{7}	1.2×10^{7}	137.6	0.05	281.4	0.04	680,857.6	4,767.9	157,344.0	120.1
24	6.4×10^{34}	—	—	29.2	—	2.87	—	14,827.8	—	8,096.1
32	9.4×10^{49}	—	—	—	—	11.3	—	—	—	24,221.1

Especially for *Bit-shifter* and *Int-shifter*, NEW has enormously better performance. This is because QEST requires conjuncts (groups) with non-intersecting supports; for these two models, the resulting groups are very large and prevent QEST from analyzing a 256-bit shifter or an 8-int shifter.

Compared with SMV, both QEST and NEW achieve much better runtime and memory consumptions for all models except *Bit-shifter* and *Int-shifter*, where QEST reports much worse results than SMV, while NEW still greatly outperforms SMV in both time and memory.

Considering now SPIN, clearly, all symbolic approaches greatly outperform SPIN unless its partial-order reduction techniques are applicable. One case where this happen is *Leader*, for which only $10N$ states are explored in the reduced state space, while the actual state space $\mathcal{S}$ grows exponentially. Nevertheless, for *Leader*, NEW can generate the state space for $N = 30$, greatly outperforming SMV and QEST in time and memory; indeed NEW's peak number of MDD nodes for $\mathcal{S}$ is $22N - 23$, just one more than the final number. For all other models, instead, SPIN fails to reduce any states, thus its explicit exploration is limited to small parameter values.

6 Related work and conclusions

For traditional breadth-first state space generation, the efficiency of image computation has been extensively studied. A *conjunctive partition* of the transition relation is the dominant approach for synchronous systems; the *conjunctive scheduling problem* [21] consider efficient clustering of the conjuncts and ordering of the clusters to minimize the size of the intermediate results during image computation. Traditionally, coarse-grain conjunctive partitioning is used to build the transition relation, and conjuncts are split only as necessary. A fine-grain conjunctive partition is instead used in [15], where bit-level conjuncts are conjoined into clusters. A *disjunctive partition* of the transition relation is instead naturally applied to asynchronous systems, but also to synchronous systems based on a Shannon's expansion [23]. [22] proposes an approach combining conjunctive partition with disjunctive recursive splitting; this differs from our approach which performs a conjunction on the results of the disjunction by events.

The presence of identities in disjunctive partitions of asynchronous circuits is suggested in [10], by limiting the image computation using disjuncts to the *dependent* variables. For software models, [2] shows how to translate conjuncts into disjuncts and vice versa. Their disjuncts modify only one state variable and the program counter, while we allow disjuncts to concurrently modify any number of state variables. Thus, [2] uses conjuncts not to "decompose" the disjuncts but to perform a pre-model-checking reduction. Furthermore, the image computation uses partial disjuncts, as in [10], but there is no merging of the partial-disjuncts while still exploiting the identity transformations, as allowed by our fully-identity reduced $2K$-MDDs.

Finally, regarding iteration orders other than breadth-first, only [20] uses a saturation-based approach; [3] uses a guided search in symbolic CTL model checking, in the hope to obtain a witness or counterexample without exploring the entire state space; and [26] uses a mixed breadth-first/depth-first approach in state-space generation based on the idea of *chaining*, a precursor to saturation.

To summarize our contribution, we introduced a new encoding for the transition relation of a discrete-state model, based on a new disjunctive-conjunctive partition and a new fully-identity reduction rule for MDDs. With this encoding, we perform symbolic state-space generation using the efficient *saturation* algorithm without having to satisfy the *Kronecker consistency* requirement. This new algorithm retains the efficiency of the original version, but has general applicability. In particular, it can be used to study models of software, for which the consistency requirement hindered the use of previous versions of saturation.

Remarkably, for saturation, encoding the transition relation with (at most) one MDD for each state variable turns out to be more efficient than the finer encoding with one MDD for each event. This suggests that **a disjunctive partition improves efficiency as long as it enables the recognition of event locality, but exploiting identity transformations is what truly matters**.

References

1. P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT-solvers. *TACAS*, LNCS 1785, pp.411–425, 2000.
2. S.Barner and I.Rabinovitz. Efficient symbolic model checking of software using partial disjunctive partitioning. *CHARME*, LNCS 2860, pp.35–50, 2003.
3. R.Bloem, K.Ravi, and F.Somenzi. Symbolic guided search for CTL model checking. *DAC*, pp.29–34, 2000.
4. R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comp.*, 35(8):677–691, 1986.
5. G.Ciardo, R.L. Jones, A.S. Miner, and R.Siminiceanu. Logical and stochastic modeling with SMART. *Tools*, LNCS 2794, pp.78–97, 2003.
6. G.Ciardo, G.Luettgen, and R.Siminiceanu. Saturation: An efficient iteration strategy for symbolic state space generation. *TACAS*, LNCS 2031, pp.328–342, 2001.
7. G.Ciardo, R.Marmorstein, and R.Siminiceanu. Saturation unbound. *TACAS*, LNCS 2619, pp.379–393, 2003.
8. G.Ciardo and A.S. Miner. A data structure for the efficient Kronecker solution of GSPNs. *PNPM*, pp.22–31, 1999.
9. A.Cimatti, E.Clarke, F.Giunchiglia, and M.Roveri. NuSMV: A new symbolic model verifier. *CAV*, LNCS 1633, pp.495–499, 1999.
10. E.M. Clarke, O.Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 1999.
11. S.Derisavi, P.Kemper, and W.H. Sanders. Symbolic state-space exploration and numerical analysis of state-sharing composed models. *NSMC*, 167–189, 2003.
12. D.Dolev, M.Klawe, and M.Rodeh. An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle. *J. of Algorithms*, 3(3):245–260, 1982.
13. S.Graf, B.Steffen, and G.Lüttgen. Compositional minimisation of finite state systems using interface specifications. *Formal Asp. of Comp.*, 8(5):607–616, 1996.
14. G.J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2003.
15. H.Jin, A.Kuehlmann, and F.Somenzi. Fine-grain conjunction scheduling for symbolic reachability analysis. *TACAS*, LNCS 2280, pp.312–326, 2002.
16. J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic model checking with partitioned transition relations. *VLSI*, 49–58, 1991. IFIP.
17. K. L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.
18. T.Kam, T.Villa, R.Brayton, and A.Sangiovanni-Vincentelli. Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic*, 4(1–2):9–62, 1998.
19. S.Kimura and E.M. Clarke. A parallel algorithm for constructing binary decision diagrams. *ICCD*, pp.220–223, 1990.
20. A.S. Miner. Saturation for a general class of models. *QEST*, pp.282–291, 2004.
21. I.-H. Moon, G.D. Hachtel, and F.Somenzi. Border-block triangular form and conjunction schedule in image computation. *FMCAD*, LNCS 1954, pp.73–90, 2000.
22. I.-H. Moon, J.H. Kukula, K.Ravi, and F.Somenzi. To split or to conjoin: the question in image computation. *DAC*, pp.23–28, 2000.
23. A.Narayan, A.J. Isles, J.Jain, R.K. Brayton, and A.Sangiovanni-Vincentelli. Reachability analysis using partitioned-ROBDDs. *ICCAD*, pp.388–393, 1997.
24. E.Pastor, O.Roig, J.Cortadella, and R.Badia. Petri net analysis using boolean manipulation. *ATPN*, LNCS 815, pp.416–435, 1994.
25. B.Plateau. On the stochastic structure of parallelism and synchronisation models for distributed algorithms. *SIGMETRICS*, pp.147–153, 1985.
26. M.Solé and E.Pastor. Traversal techniques for concurrent systems. *FMCAD*, LNCS 2517, pp.220–237, 2002.