Reachability set generation using hybrid relation compatible saturation

Shruti Biswal and Andrew S. Miner

Iowa State University, Ames IA 50011, USA

Abstract. Generating the state space of any finite discrete-state system using symbolic algorithms like saturation requires the use of decision diagrams or compatible structures for encoding its reachability set and transition relations. For systems that can be formally expressed using ordinary Petri Nets(PN), implicit relations, a static alternative to decision diagram-based representation of transition relations, can significantly improve the performance of saturation. However, in practice, some systems require more general models, such as self-modifying Petri nets, which cannot currently utilize implicit relations and thus use decision diagrams that are repeatedly rebuilt to accommodate the changing bounds of the system variables, potentially leading to overhead in saturation algorithm. This work introduces a hybrid representation for transition relations, that combines decision diagrams and implicit relations, to reduce the rebuilding overheads of the saturation algorithm for a general class of models. Experiments on several benchmark models across different tools demonstrate the efficiency of this representation.

Keywords: Petri nets, Self-modifying nets, Decision Diagrams, Saturation, Reachability set generation, Implicit relations

1 Introduction

With rapid progress in the development of automated systems, formal verification techniques have increasingly aided in building thorough and reliable automation. One such technique, model checking, involves an exhaustive analysis of all possible systemic behavior, conventionally defined in terms of its variables and events, for a design to be verified. This motivates the need for efficient state-space generation methods, in spite of the fact that the reachability set of a system can be extremely large due to state explosion problem.

State-of-the-art symbolic techniques for state-space generation, such as *saturation* [4], are time- and memory-efficient with respect to the explicit methods. Several implementations of the saturation algorithm demonstrate that the technique has been often fine-tuned to perform efficiently with different sub-classes of discrete-state systems. The most generic implementation of the saturation algorithm utilizes multi-valued decision-diagrams (MDDs)[13] as the underlying data structure for representation of reachable states and extensible matrix diagrams (MxDs) [4, 20] for representation of system events where the bounds of the states variables are not known a priori. For systems that can be represented in terms of Kronecker products, a static encoding for the events, such as *implicit relation forests* [3], is an efficient alternative that aids saturation by eliminating the need to rebuild transition relations as state variable sizes increase, thereby allowing the algorithm to focus completely on generation of reachability set.

In the real world, however, systems may have some events that are not expressible as Kronecker products, or are expressible as Kronecker products except for a few components. Such models would require either to use matrix diagrams entirely, or to split events until each event is expressible as a Kronecker product so that implicit relation forests may be used. The aim of the paper is to explore a *hybrid relation*, that allows the mixture of implicit nodes (from implicit relation forests) and matrix diagram nodes in the same forest. This makes it possible to simultaneously use implicit nodes for (portions of) events expressible as a Kronecker product, and matrix diagram nodes for the more general case.

The remainder of the paper is organized as follows. Section 2 describes the formalism for the class of models befitting the proposed methodology, the saturation algorithm and briefly discusses the various saturation-compatible data structures available in the literature. Section 3 introduces hybrid relation forests, describes their construction from a model, and formulates a modified saturation algorithm that works with events represented in a hybrid relation forest. Section 4 evaluates the proposed method experimentally by comparing its performance with existing techniques in the literature. Finally, Section 5 concludes the paper and presents some directions for future work.

2 Background and related work

In order to describe the problem domain, we use a standard definition of extended Petri nets, which are quite general and include inhibitor arcs and marking-dependent arc cardinalities, as a high-level formalism to define the class of finite discrete-state models.

Definition 1 (Extended Petri Net). The high-level formalism of a model $\mathcal{M}$ as an extended Petri net is defined as a tuple $(\mathcal{P}, \mathcal{T}, \mathcal{F}^-, \mathcal{F}^+, \mathcal{F}^o, \mathbf{i}_0)$ where:

- $\mathcal{P} = \{p_1, p_2, \ldots, p_{|\mathcal{P}|}\}$ is a finite set of *places* that encode the state variables of $\mathcal{M}$. Each place can contain a natural number of tokens. A marking $\mathbf{i} = (i_1, i_2, \ldots, i_{|\mathcal{P}|}) \in \mathbb{N}^{|\mathcal{P}|}$ represents the number of tokens in each place.
- $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ is a finite set of *transitions* of the net that represent the events of $\mathcal{M}$.
- $\mathcal{F}^- : \mathcal{P} \times \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N}$, $\mathcal{F}^+ : \mathcal{T} \times \mathcal{P} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N}$, and $\mathcal{F}^o : \mathcal{P} \times \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N} \cup \{\infty\}$ are the marking-dependent input, output, and inhibitor arcs, respectively.
- $\mathbf{i}_0 \in \mathbb{N}^{|\mathcal{P}|}$ is the initial marking of the net.

A transition t is *enabled* on a marking $\mathbf{i}$ iff $\forall p \in \mathcal{P}$, $\mathcal{F}^-(p, t, \mathbf{i}) \leq i_p < \mathcal{F}^o(p, t, \mathbf{i})$. When multiple transitions are enabled on a marking, the choice for

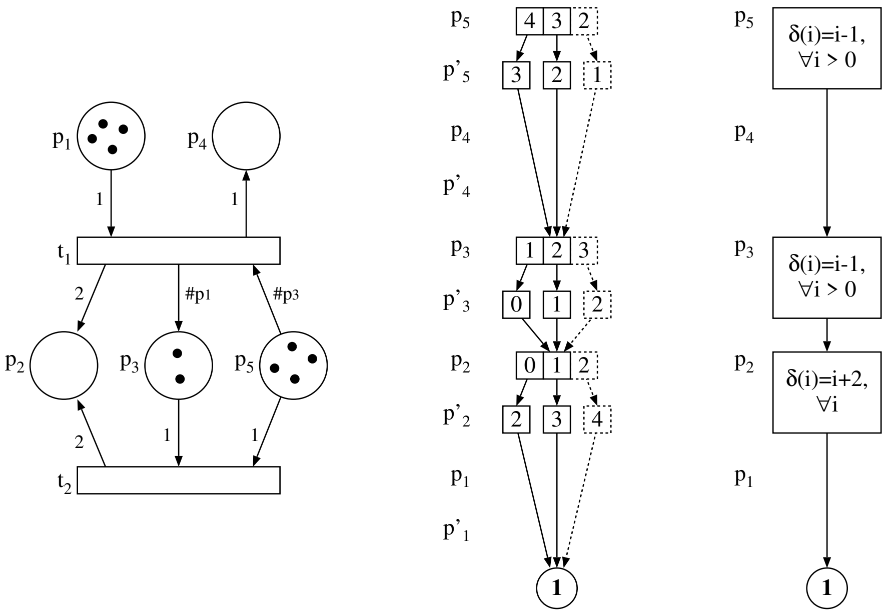

Fig. 1: A Petri net model with marking-dependent arcs, with 2L-level MDD and implicit relation representation of transition t_2.

firing a transition is made non-deterministically. On firing an enabled transition t on $\mathbf{i}$, a new marking $\mathbf{j}$ is obtained, where $\mathbf{j}_p = \mathbf{i}_p - \mathcal{F}^-(p,t,\mathbf{i}) + \mathcal{F}^+(t,p,\mathbf{i}), \forall p \in \mathcal{P}$.

The next-state function $\mathcal{N}_t(\mathcal{I}) = \{\mathbf{j} : \mathbf{i} \in \mathcal{I} \text{ and } \mathbf{i} \xrightarrow{t} \mathbf{j}\}$ represents the next set of markings obtained when transition t is fired on a given set of markings, $\mathcal{I}$. The support-set of a next-state function, defined as $supp(\mathcal{N}_t) = \{p \in \mathcal{P} : \exists \mathbf{i} \in \mathbb{N}^{|\mathcal{P}|}, \mathcal{F}^-(p,t,\mathbf{i}) > 0 \vee \mathcal{F}^+(t,p,\mathbf{i}) > 0 \vee \mathcal{F}^o(p,t,\mathbf{i}) < \infty\}$, is the set of places that could affect the enabling of t, or could affect or be affected by the firing of t.

Given the initial marking $\mathbf{i}_0$, a marking $\mathbf{i}'$ is said to be reachable if there exists a finite sequence of transitions which when fired in succession starting from $\mathbf{i}_0$ can lead to $\mathbf{i}'$. The set of all possible reachable markings of a Petri net, denoted by $\mathcal{S} \subseteq \mathbb{N}^{|\mathcal{P}|}$, is the least fixed point such that $\mathcal{S} = \{\mathbf{i}_0\} \cup \bigcup_{t \in \mathcal{T}} \mathcal{N}_t(\mathcal{S})$

The objective of this work is to generate the set $\mathcal{S}$ using saturation-based symbolic state-space generation. The algorithms in the paper are based on the assumption that $\mathcal{S}$ is finite. Therefore, all state variables of the model are bounded, but these bounds do not need to be known a priori as they are discovered during execution of the algorithm.

2.1 Symbolic representation of state space and transition relations

The state-space of a model is a set of all reachable states, represented symbolically as an indicator function encoded using decision diagrams [19].

Definition 2 (Multi-valued Decision Diagram [13]). Defined over an ordered sequence of L domain variables $(v_L, \ldots, v_1)$ such that $v_{k+1} \succ v_k$ and the variable-specific domains are denoted as $\mathcal{D}_k = \{0, 1, \ldots, n_k\}$, an ordered MDD is a directed acyclic graph where :

- *Terminal* nodes are labeled **0** or **1** and are associated with a special variable v_0 such that any domain variable $v_k \succ v_0$.
- Every *non-terminal* node p is associated with some domain variable $\mathsf{p}.var = v_k \succ v_0$ and has $\mathcal{D}_k$ child pointers denoted by $\mathsf{p}[i_k]$ such that $\mathsf{p}.var \succ \mathsf{p}[i_k].var$.

Two nodes, p and q, in an MDD are said to be *duplicates* if $\mathsf{p}.var = \mathsf{q}.var = v_k$ and $\forall i_k \in \mathcal{D}_k, \mathsf{p}[i_k] = \mathsf{q}[i_k]$. If all pointers of a node point to the same child, then the parent node is termed as *redundant node.* To correctly interpret the unknown but finite variable bounds which are explored during generation, *on-the-fly* saturation requires the use of *quasi-reduced* MDDs, which forbid duplicate nodes but require redundant nodes except for all **0** children: for every non-terminal node p and for every child i_k, either $\mathsf{p}[i_k] = \mathbf{0}$ or $\mathsf{p}[i_k].var = \mathsf{p}.var - 1$.

Similar to MDDs, a transition relation of a model can be encoded in the form of a set of pairs of states and their next-states using 2L-level MDDs where an indicator function is defined over $(\mathcal{D}_k \times \mathcal{D}_k) \times \ldots \times (\mathcal{D}_1 \times \mathcal{D}_1) \rightarrow \{\mathbf{0}, \mathbf{1}\}$. The first set in each pair corresponds to unprimed or "from" state of variable and the second set refers to primed or "to" state of variable.

Given a Petri net encoding $(\mathcal{P}, \mathcal{T}, \mathcal{F}^-, \mathcal{F}^+, \mathcal{F}^o, \mathbf{i}_0)$ of a model, its reachability set, $\mathcal{S}$ and transition relation, $\mathcal{N}$ can be encoded using MDDs :

- The domain variables $(v_L, \ldots, v_1)$ of the MDD correspond to the Petri net $\mathcal{P} = (p_1, p_2, \ldots p_{|\mathcal{P}|})$ such that each variable v_k is associated with one p_i, for simplicity.
- The next-state function, $\mathcal{N}_t(\mathbf{i}) = \mathbf{j}$ can be represented by a 2L-level MDD where nodes corresponding to unprimed level and primed level of variable v_k encode the pair (i_k, j_k).

Nodes in a 2L-level MDD can be redundant or duplicate under same definitions as above. While a 2L-level MDD can be *reduced* as per any choice of reduction rule, the proposed work requires the use of *Fully-Fully* [7] reduced 2L-level MDD which results in skipping of redundant nodes.

Figure 1 shows a small Petri net (left), and symbolic encodings of the relation for transition t_2 using a both a 2L-level MDD (middle) and an implicit relation (right). It is noticed that for a Petri net transition where the modified value of a state variable is a function of the variable itself, the 2L-level MDD of the transition assumes a peculiar shape, wherein every edge from the nodes associated with some primed variable v'_k points to the same child node. This structure creates an unnecessary overhead for on-the-fly saturation as it requires to be rebuilt each time the variable bound expands. In order to encode $\mathcal{N}_t$ of such PNs efficiently, [3] proposes an alternate representation, *implicit relation.* The full encoding of $\mathcal{N}$ contains $|\mathcal{T}|$ implicit relations uniquely identified by their topmost nodes, forming an *implicit relation forest.*

Definition 3 (Implicit Relation). An implicit relation defined over the sequence of L domain variables $(v_L, \ldots, v_1)$ is a compact version of a 2L-level MDD where the unprimed and primed levels of a variable are combined into a single node, *relation node* which encodes the local effect of the transition on that variable. Each relation node r has a single downward pointer $r.ptr$ to another node such that $r.var \succ r.ptr.var$.

Previous works of Couvreur et. al [9] have proposed *homomorphisms* as an alternative approach to symbolically encode transition relations that works well with Data Decision Diagrams (DDD) [9] and Hierarchical Set Decision Diagrams (SDD) [11] for reachability generation. Homomorphisms are a class of operators which are used on DDDs to encode the function represented by the transition relation. Additionally, this representation is generic and is designed well [10] for extended Petri nets like 2L-level MDDs. Molnar et. al in [14] proposed *abstract next-state diagrams* that encode the transition relations via the use of level-wise *descriptors* to capture the local effect of the transitions. While the domain of models and the key idea of abstracting Petri net transitions is similar to that of implicit relations, the facility with implicit relations to skip the encoding for non-support variables of transitions provides a computational advantage during saturation. The various symbolic representations of transition relations discussed in this section are used as benchmark techniques for performance comparison of the proposed work.

2.2 Saturation

Saturation is a symbolic algorithm for state-space generation that uses DAG-type data structures like decision diagrams or implicit relations for storing and computing operations on the data. In order to calculate the state-space of any system, a breadth first iteration on the fixed point equation $\mathcal{S} = \{\mathbf{i}_0\} \cup \mathcal{N}(\mathcal{S})$, where $\mathcal{N}(\mathcal{S}) = \bigcup\limits_{t \in \mathcal{T}} \mathcal{N}_t(\mathcal{S})$ is an explicit method and is computationally inefficient. However, the strategy of *node saturation* computes the fixed point recursively by exploring reachable states through continuous computation of *relational product* between $\mathcal{S}$ and $\mathcal{N}$ in a bottom-up fashion. The algorithm is described in procedure Saturate of Fig 4. On-the-fly saturation [5, 15, 16] is a variation that explores the variable bounds during reachability set generation and eliminates the need for knowing the variable bounds a priori. However, this dynamic expansion of variable bounds involves certain overhead due to re-construction of $\mathcal{N}$. *Extensible decision diagrams* [20] were introduced to alleviate some of these overheads.

3 Hybrid Relations

Taking into consideration the generality of real-world discrete-state systems and the need to encode these models for efficient execution of on-the-fly saturation techniques, this section formulates an ensemble of decision diagrams and implicit relations for representation of next-state functions where the existence of dependency among variables in the functions and the choice of partitioning technique for the transition relations determine the data structure to be used. This representation combines the power of generalization from 2L-level MDDs and the efficiency of static description from implicit relations. We also modify the on-the-fly saturation algorithm to work with the hybrid relations.

Definition 4 (Hybrid relation). Given a transition $t \in \mathcal{T}$ of a Petri net with a conjunctively-partitioned next-state function $\mathcal{N}_t = \bigcap_{i=1}^{m_t} \mathcal{N}_{t,c_i}$, a hybrid relation encoding of t has a corresponding set of m_t DAGs as follows.

- Each DAG encodes one of the sub-relations $\mathcal{N}_{t,c}$, using either an implicit relation or a 2L-level MDD. Each DAG is uniquely identified by its topmost node.
- Every 2L-level MDD obeys FF-reduction rule for skipping nodes associated with variables $v \notin \mathit{supp}(\mathcal{N}_{t,c})$.
- Every implicit relation contains a single non-terminal relation node with pointer to the terminal **1**.
- The interpretation of skipped levels is based on the overall hybrid relation of the transition: if a skipped variable v affects the transition at all, then $v \in \mathit{supp}(\mathcal{N}_t)$, and thus $v \in \mathit{supp}(\mathcal{N}_{t,c})$ for some c. In this case, the FF-reduction ensures that the intersection is handled appropriately. If instead v does not affect the transition ad $v \notin \mathit{supp}(\mathcal{N}_t)$, then $v \notin \mathit{supp}(\mathcal{N}_{t,c})$ for any c and the interpretation is "v is not changed by the firing of t".

3.1 Constructing next-state functions

We construct the DAG for each sub-relation $\mathcal{N}_{t,c}$ based on the number of variables in its support. The case $\mathit{supp}(\mathcal{N}_{t,c}) = \emptyset$ corresponds to state variables that do not affect, and are not affected by, the firing of t in the sub-relation; we assume that this case is already handled (because each such variable v will either not affect t at all because $v \notin \mathit{supp}(\mathcal{N}_t)$, or will affect t but then $v \in \mathit{supp}(\mathcal{N}_{t,c'})$ for some other c'). When $\mathit{supp}(\mathcal{N}_{t,c}) = \{v\}$, $\mathcal{N}_{t,c}$ can be represented using an implicit relation node r where $r.\mathit{var} = v$, the partial function $r.\delta$ encodes the sub-relation and $r.\mathit{ptr} = \mathbf{1}$. In practice, the implicit relation nodes can be compacted (conjuncted) to form a single implicit relation, which will be the original implicit nodes chained together. This has a performance advantage since it decreases the number of sub-relations that needs to be kept track of during saturation for their *symbolic conjunction*. When $|\mathit{supp}(\mathcal{N}_{t,c})| > 1$, $\mathcal{N}_{t,c}$ must be encoded using a 2L-level MDD over the domain variables in $\mathit{supp}(\mathcal{N}_{t,c})$.

Different methods may be used for partitioning a transition relation $\mathcal{N}_t$ into its sub-relations. One method [16] is to use the finest possible partition, such that each state variable belongs to the support of at most one sub-relation. Another method [7] separates the transition enabling and firing conditions, and then uses the finest possible partition so that each variable is *modified* in at most one sub-relation (but may appear, read-only, in other sub-relations). These methods are illustrated in Figure 2 for transition t_1 from the PN in Figure 1. The left side uses

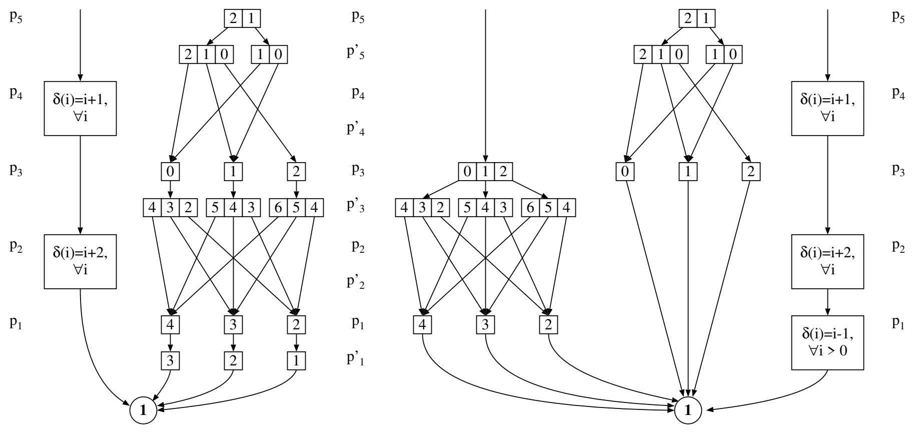

Fig. 2: Hybrid relation-based encodings of transition t_1 from Figure 1 when partitioning techniques $\Theta_{\circ}$ (left) and $\Theta_{\square}$ (right) are used.

the method from [16], and obtains two sub-relations: one as a chain of implicit nodes, representing $p'_4 = p_4+1$ and $p'_2 = p_2+2$, and one as an MDD, representing $(p_5 \geq p_3) \wedge (p'_5 = p_5 - p_3) \wedge (p'_3 = p_3 + p_1) \wedge (p_1 \geq 1) \wedge (p'_1 = p_1 - 1)$. The right side uses the method from [7], and obtains three sub-relations: one representing $p'_3 = p_3 + p_1$, one representing $(p_5 \geq p_3) \wedge (p'_5 = p_5 - p_3)$, and a chain of implicit nodes representing $p'_4 = p_4 + 1$, $p'_2 = p_2 + 2$, and $(p_1 \geq 1) \wedge (p'_1 = p_1 - 1)$.

Algorithm BuildHybrid, shown in Figure 3, constructs an overall hybrid relation for a given transition $t \in \mathcal{T}$, as a set of DAGs. This should be called once, for each transition, before saturation is invoked. The set of sub-relation DAGs for transition t is denoted by $\mathcal{N}_t.nodes$. Note that the overall transition relation $\mathcal{N}_t$ is not computed, but rather is evaluated during the saturation algorithm on the fly. The hybrid relations are updated during saturation as new local states are discovered, using algorithm Confirm, also shown in Figure 3.

3.2 Saturation using hybrid relations

Saturating a node for variable v_k requires repeatedly firing transitions in $\mathcal{T}_k$, the set of transitions whose top-most variable is v_k (i.e., $supp(\mathcal{N}_t) \cap \{v_L, \ldots, v_k\} = \{v_k\}$), until a fixed point is reached. The algorithm is shown in Figure 4. Procedures RecFire and Saturate are similar to those of saturation using MxDs [16] (using either a single DAG per transition, or a single DAG for all of $\mathcal{T}_k$), except all nodes of the hybrid relation for a transition are traversed simultaneously. Thus, in addition to the MDD node encoding the set of states, procedure RecFire requires a set of hybrid relation nodes, rather than a single MxD node (as used in [16]) or a transition identifier (as used in [5]). This also means that the compute table entries for RecFire must include the set of hybrid relation nodes (c.f. lines 2 and 15).

```
BuildHybrid($transition\ t$)
• Builds hybrid relation for Petri net transition $t$
1: Partition $\mathcal{N}_t$ into $\mathcal{N}_{t,c_1}, \mathcal{N}_{t,c_2}, \ldots \mathcal{N}_{t,c_{m_t}}$
2: $n \leftarrow \mathbf{1}$;
3: for each $\mathcal{N}_{t,c}$ s.t. $|supp(\mathcal{N}_{t,c})| = 1$, from bottom up, do
4:     $r \leftarrow$ new relation node;
5:     $r.var \leftarrow \mathcal{N}_{t,c}.var$ ;
6:     $r.\delta \leftarrow \mathcal{N}_{t,c}.\Delta$ ;
7:     $r.ptr \leftarrow n$;
8:     $n \leftarrow r.ptr$;
9: $\mathcal{N}_t.nodes \leftarrow \{n\} \setminus \{\mathbf{1}\}$;
10: for each $\mathcal{N}_{t,c}$ s.t $|supp(\mathcal{N}_{t,c})| > 1$ do
11:     $\mathbf{s}_{from} \leftarrow \mathbf{s}_0$;
12:     $\mathbf{s}_{to} \leftarrow \mathcal{N}_{t,c}(\mathbf{s}_{from})$;
13:     $\mathcal{N}_{t,c}$.updateHybrid($\mathbf{s}_{from}, \mathbf{s}_{to}$);
14:     $\mathcal{N}_t.nodes \leftarrow \mathcal{N}_t.nodes \cup \mathcal{N}_{t,c}.node$;
```

```
Confirm($level\ k$, $index\ i$)
• Updates relevant relations $\mathcal{N}_t$, $\mathcal{N}_{t,c}$ if local state $v_k = i$ is new
1: if $i \notin \mathcal{D}_k$ then
2:     $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \{i\}$;
3:     for each $t \in \mathcal{T}$ do
4:         for each $\mathcal{N}_{t,c}$ s.t. $|supp(\mathcal{N}_{t,c})| > 1$ and $k \in supp(\mathcal{N}_{t,c})$ do
5:             $\mathcal{N}_t.nodes \leftarrow \mathcal{N}_t.nodes \setminus \mathcal{N}_{t,c}.node$;
6:             for each $\mathbf{s}_{from} \in \mathcal{N}_{t,c}$.buildState($k, i$);
7:                 $\mathbf{s}_{to} \leftarrow \mathcal{N}_{t,c}(\mathbf{s}_{from})$ ;
8:                 $\mathcal{N}_{t,c}$.updateHybrid($\mathbf{s}_{from}, \mathbf{s}_{to}$)
9:             $\mathcal{N}_t.nodes \leftarrow \mathcal{N}_t.nodes \cup \{\mathcal{N}_{t,c}.node\}$;
```

Fig. 3: Algorithms for building a Hybrid Relation.

The actual firing with a set of hybrid relation nodes is handled in helper procedure Fire, also shown in Figure 4. This takes in a set of hybrid relation nodes, and determines the set of nodes to follow one level below, by examining each node in the set and following its downward pointer ($h[i][j]$ in line 4), unless the current level is skipped, in which case the current node is used again (line 3). Note that $h[i][j]$ must be determined differently, based on if h is an implicit node, or an MDD node. If the set of downward pointers contains terminal node $\mathbf{0}$, then the intersection of the sub-relations will be empty, and in this case no states will be reached (line 5). Otherwise, the traversal continues by calling RecFire recursively (line 6). If the reached set of states is non-empty, then local state variable j must be confirmed [5] if it is new (line 7).

```
$mdd$ Fire($mdd\ n, nodeset\ \mathcal{H}, index\ i, j$)
• Fire hybrid relation nodes $\mathcal{H}$ on node $n$, from $i$ to $j$.
1: $\mathcal{H}' \leftarrow \emptyset$;                          • Determine next-level relation nodes
2: for each $h \in \mathcal{H}$ do
3:     if $n.var > h.var$ then   $\mathcal{H}' \leftarrow \mathcal{H}' \cup \{h\}$;
4:     else                      $\mathcal{H}' \leftarrow \mathcal{H}' \cup \{h[i][j]\}$;
5: if $\mathcal{H}' = \emptyset$ or $\mathbf{0} \in \mathcal{H}'$ then return $\mathbf{0}$;
6: $f \leftarrow$ RecFire($n[i], \mathcal{H}'$);
7: if $f \neq \mathbf{0}$ then Confirm($n.var, j$);
8: return $f$;
```

```
Saturate($level\ k, mdd\ n$)
• Saturate node $n$ at level $k$ using $\mathcal{T}_k$.
1: $\mathcal{Q} \leftarrow \{i : n[i] \neq \mathbf{0}\}$;
2: while $\mathcal{Q} \neq \emptyset$ do
3:     $i \leftarrow$ Choose($\mathcal{Q}$);
4:     $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{i\}$;
5:     for each $t \in \mathcal{T}_k$
6:         $h^t \leftarrow$ topmost($\mathcal{N}_t.nodes$);
7:         for each $j$ s.t. $h^t[i][j] \neq \mathbf{0}$ do
8:             $f \leftarrow$ Fire($n, \mathcal{N}_t.nodes, i, j$);
9:             $u \leftarrow$ Union($f, n[j]$)
10:            if $u \neq n[j]$ then
11:                $n[j] \leftarrow u$;
12:                $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{j\}$;
```

```
$mdd$ RecFire($mdd\ n, nodeset\ \mathcal{H}$)
• Fire $\mathcal{H}$ on node $n$ and then saturate
1: if $n = \mathbf{0}$ or $\mathcal{H} = \{\mathbf{1}\}$ then return $n$;
2: if (RecF, $n, \mathcal{H}, m$) $\in$ CT then return $m$;
3: $h^t \leftarrow$ topmost($\mathcal{H}$);
4: $k \leftarrow$ max($n.var, h^t.var$);
5: $m \leftarrow$ new MDD node for variable $v_k$;
6: if $n.var > h^t.var$ then
7:     for each $i \in \mathcal{D}_k$ do
8:         $m[i] \leftarrow$ RecFire($n[i], \mathcal{H}$);
9: else
10:    for each $i, j$ s.t. $n[i] \neq \mathbf{0}$, $h^t[i][j] \neq \mathbf{0}$
11:        $f \leftarrow$ Fire($n, \mathcal{H}, i, j$);
12:        $m[j] \leftarrow$ Union($m[j], f$);
13: Saturate($k, m$);
14: $m \leftarrow$ UniqueInsert($m$);
15: CT $\leftarrow$ CT $\cup$ (RecF, $n, \mathcal{H}, m$);
16: return $m$;
```

Fig. 4: Saturation using hybrid relations

4 Experimental Results

4.1 Setup

We implemented the hybrid relation-based saturation algorithm, HybSat, using the relation partitioning method of [16] in SMART [6], using Meddly [2] as the underlying decision diagram library. We conducted two groups of experiments to compare the performance of HybSat,

- **Group I**, with the existing "on-the-fly saturation with matrix diagrams" approach (OtfSat) from SMART and with DDD/SDD-based reachability set generation from ITS-Tools (ITSTools) for extended PN models on identical static variable order.
- **Group II**, with the saturation algorithm based on a generalized representation of next-state functions (AbstractDesc) from [17] for ordinary PN models obtained from [1] using SOUPS [18] ordering for variables.

All experiments are run on a server of Intel Xeon CPU 2.13GHz with 48G RAM under Linux Kernel 4.9.9 with timeout for each run set to 20 minutes. The suite of extended PNs used are listed in the Appendix.

4.2 Observations

Table 1 and Table 2 summarize the experiments run on extended PNs and ordinary PNs respectively. The metrics used for performance comparison include the time taken by the reachability generation algorithm and the total number of peak nodes required by the execution for encoding the transition relations. For experiments conducted under Group I (see Table 1), the observations from OtfSat and HybSat reveal that models that have transitions with tightly-bound sub-relation effects, force the HybSat to build 2L-level MDDs leading to similar metric values and often these values are worse with respect to ITSTools. For example, models like Stack, the sub-relations display a chain-dependency among the variables leading to creation of a full 2L-level MDD for encoding the relation. For models such as FMS, where all but four transitions have constant arc cardinalities, HybSat performs better than ITSTools but shows similar performance with OtfSat. The overall results in these experiments indicate that higher the number of relation nodes in the encoding, better is the performance of HybSat. However, the ability of HybSat to encode general transitions make it as versatile as OtfSat and ITSTools

The observations of Group II experiments that are summarized in Table 2, confirm that within a given model, the ratio of execution times of saturation algorithm in the Java implementation of AbstractDesc to that of HybSat in SMART is fairly constant. While these encodings and algorithms are conceptually similar, the significant yet constant difference in the execution times can be attributed to the implementational details and underlying language. Note that the number of nodes in case of AbstractDesc is higher compared to HybSat because the former does not skip descriptors that encode identity relations and it counts every pair of "from" and "to" local states as defined by the descriptors. The number of nodes encoded for $\mathcal{N}$ by HybSat for is at most equal to the number of arcs in the Petri net definition. Since HybSat pre-determines the encoding strategy for each transition, for all models under this experiment group, their transitions are represented as relation nodes. Hence, the comparison metrics for reachability generation in HybSat and ImplSat are exactly the same and therefore, the observations of ImplSat are omitted. The missing fields refer to timeout of the experimental run.

5 Conclusions

In this paper, we introduced hybrid relations, an ensemble data-structure, for symbolic encoding of transition relations in a general class of discrete-state systems. The translation of a system event into its corresponding hybrid relation, Performance comparison of reachability generation among SMART (OtfSat, HybSat) and ITSTools from ITS-tools over benchmark of extended PN models

Instance	$\lvert\mathcal{S}\rvert$	Time (in sec)			Peak Nodes: $\mathcal{N}$		
		OtfSat	ITSTools	HybSat	OtfSat	ITSTools	HybSat
Swap							
10	3.6×10^{6}	2.7×10^{-01}	9×10^{-01}	3.3×10^{-01}	1149	58	1091
12	4.8×10^{8}	3.0×10^{00}	9.5×10^{00}	3.5×10^{00}	1890	70	1861
Stack							
D8 V5	3.9×10^{5}	2.7×10^{00}	6.0×10^{-02}	2.7×10^{00}	290	41	242
D10 V3	5.2×10^{47}	6.3×10^{2}	5.0×10^{00}	5.7×10^{2}	2763	409	2120
Leader							
6	2.39×10^{7}	4.0×10^{01}	5.8×10^{01}	6.1×10^{01}	1109	79	476
8	3.0×10^{8}	1.9×10^{02}	2.3×10^{02}	2.7×10^{02}	1676	105	735
Flexible Manufacturing System							
15	2.7×10^{11}	2.9×10^{-01}	5.3×10^{-01}	1.7×10^{-01}	1544	118	355
20	6.0×10^{12}	4.2×10^{-01}	1.29×10^{00}	3.9×10^{-01}	1660	118	505
30	7.7×10^{14}	9.1×10^{-01}	5.51×10^{00}	1.1×10^{00}	2800	118	880
Tiles							
N3M3	1.8×10^{05}	1.4×10^{00}	8.9×10^{00}	2.2×10^{00}	952	154	362
N5M5	1.8×10^{06}	1.5×10^{01}	5.5×10^{01}	1.3×10^{01}	1223	168	433
Tower of Hanoi							
D8	6.5×10^{03}	3.7×10^{00}	5.9×10^{00}	4.3×10^{00}	4566	40	1219
D10	5.9×10^{04}	5.4×10^{01}	7.0×10^{01}	5.9×10^{01}	9837	40	2481
Satellite							
X 100 Y 3	7.6×10^{04}	2.1×10^{01}	1.1×10^{01}	1.9×10^{01}	3207	98	737

involves structural analysis of the event in terms of dependencies among the variables of the event. This prior system review to assign a suitable data-structure for encoding each (sub)event rewards during saturation, since unnecessary overhead costs of managing the compute-table entries for 2L-level MDD operations and saturation are eliminated. The efficiency and validity of the proposed saturation algorithm is empirically evaluated with respect to several versions of saturation algorithm.

Although this paper briefly discusses the use of event partitioning techniques for building hybrid relations, the consequence of each technique on the efficiency of hybrid relations based saturation algorithm is yet to be explored. This can pave way for applying desirable partitioning methods for each event instead of making an apriori choice for all events. Building and incorporating compatible data structures into hybrid relations that can optimally utilize certain event behaviors can make the representational formalism more powerful and efficient for saturation-based symbolic reachability generation.

Performance comparison of reachability generation among SMART (ImpSat, HybSat) and AbstractDesc over benchmark of ordinary PN models

Model	$\lvert\mathcal{S}\rvert$	Time (in sec)		Peak Nodes for encoding $\mathcal{N}$	
		AbstractDesc	HybSat	AbstractDesc	HybSat
Eratosthenes					
200	1.1×10^{46}	4.09×10^{-01}	2.27×10^{-02}	441	603
500	4.1×10^{121}	2.32×10^{00}	2.35×10^{-01}	2820	1875
Philosophers					
1000	1.3×10^{477}	2.24×10^{01}	3.98×10^{00}	62410	15000
2000	4.0×10^{2385}	7.03×10^{01}	2.41×10^{01}	124917	30000
SwimmingPool					
10	3.4×10^{10}	5.87×10^{01}	3.97×10^{01}	721877	20
Kanban					
200	3.2×10^{22}	1.26×10^{01}	1.92×10^{00}	44028	39
500	7.1×10^{26}	1.51×10^{02}	2.94×10^{01}	260027	39
HouseConstruction					
20	1.4×10^{13}	1.27×10^{01}	3.06×10^{00}	279043	51
50	1.6×10^{19}	1.74×10^{02}	4.37×10^{01}	2280222	51
SmallOperatingSystem					
MT512 DC128	1.0×10^{11}	1.51×10^{02}	4.05×10^{01}	348039	22
MT512 DC256	2.5×10^{11}	3.13×10^{02}	1.03×10^{02}	594183	22
GPPP					
C10 N100	1.8×10^{11}	7.25×10^{00}	1.34×10^{01}	194454	79
Referendum					
100	5.2×10^{47}	4.48×10^{-01}	1.57×10^{-02}	2701	460
200	2.7×10^{95}	3.14×10^{01}	9.07×10^{00}	529255	940
TCPcondis					
15	5.4×10^{12}	$--$	8.48×10^{01}	4519532	90
20	5.6×10^{14}	$--$	3.03×10^{02}	$--$	90
Raft					
6	2.9×10^{26}	$--$	1.87×10^{01}	$--$	620
7	3.6×10^{35}	$--$	1.46×10^{02}	$--$	829

Acknowledgment

This work was supported in part by the National Science Foundation under grant ACI-1642327.

References

1. MCC : Model Checking Competition @ Petri Nets. https://mcc.lip6.fr.
2. J. Babar and A. S. Miner. Meddly: Multi-terminal and Edge-valued Decision Diagram LibrarY. In *Proc. QEST*, pages 195–196. IEEE Computer Society, 2010.
3. S. Biswal and A. S. Miner. Improving saturation efficiency with implicit relations. In *Application and Theory of Petri Nets and Concurrency*, pages 301–320. Springer International Publishing, 2019.

4. G. Ciardo, G. Lüttgen, and R. Siminiceanu. Saturation: An efficient iteration strategy for symbolic state space generation. In *Proc. TACAS*, LNCS 2031, pages 328–342. Springer, 2001.
5. G. Ciardo, R. Marmorstein, and R. Siminiceanu. Saturation unbound. In *Proc. TACAS*, LNCS 2619, pages 379–393. Springer, 2003.
6. G. Ciardo and A. S. Miner. SMART: Stochastic Model checking Analyzer for Reliability and Timing, User Manual. Available at http://smart.cs.iastate.edu.
7. G. Ciardo and A. J. Yu. Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In *Proc. CHARME*, LNCS 3725, pages 146–161. Springer, 2005.
8. A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model verifier. In *Proc. CAV*, LNCS 1633, pages 495–499. Springer, 1999.
9. J.-M. Couvreur, E. Encrenaz, E. Paviot-Adet, D. Poitrenaud, and P.-A. Wacrenier. Data decision diagrams for petri net analysis. In *Proc. ATPN*, pages 101–120. Springer, 2002.
10. J.-M. Couvreur and Y. Thierry-Mieg. Hierarchical decision diagrams to exploit model structure. In *Proc. Formal Description Techniques, FORTE95*, volume 3731 of *LNCS*, pages 443–4572, 2005.
11. A. Hamez, Y. Thierry-Mieg, and F. Kordon. Hierarchical set decision diagrams and automatic saturation. In *Proc. ATPN*, LNCS 5062, pages 211–230. Springer, June 2008.
12. A. Itai and M. Rodeh. Symmetry breaking in distributed networks. In *22th Annual Symp. on Foundations of Computer Science*, pages 150–158. IEEE Comp. Soc. Press, Oct. 1981.
13. T. Kam, T. Villa, R. K. Brayton, and A. Sangiovanni-Vincentelli. Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic*, 4(1–2):9–62, 1998.
14. K. Marussy, V. Molnár, A. Vörös, and I. Majzik. Getting the priorities right: Saturation for prioritised petri nets. In W. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets and Concurrency*, pages 223–242, Cham, 2017. Springer International Publishing.
15. S.-i. Minato. Zero-suppressed BDDs and their applications. *Software Tools for Technology Transfer*, 3:156–170, 2001.
16. A. S. Miner. Saturation for a general class of models. *IEEE Trans. Softw. Eng.*, 32(8):559–570, Aug. 2006.
17. V. Molnár and I. Majzik. Saturation enhanced with conditional locality: Application to petri nets. In S. Donatelli and S. Haar, editors, *Application and Theory of Petri Nets and Concurrency*, pages 342–361, Cham, 2019. Springer International Publishing.
18. B. Smith and G. Ciardo. Soups: A variable ordering metric for the saturation algorithm. In *2018 18th International Conference on Application of Concurrency to System Design (ACSD)*, pages 1–10, 2018.
19. F. Somenzi. Binary decision diagrams. In *Calculational System Design, volume 173 of NATO Science Series F: Computer and Systems Sciences*, pages 303–366. IOS Press, 1999.
20. M. Wan and G. Ciardo. Symbolic state-space generation of asynchronous systems using extensible decision diagrams. In *Proc. SOFSEM*, LNCS 5404, pages 582–594. Springer, 2009.

Appendix : List of Extended Petri Nets

- *Swap* model where given a list of N distinct integers, operations are to exchange two neighboring integers.
- *Stack* model that generates the count of possible combination of D objects that can be have values in the range 1 to N.
- A simplified version of *Leader* election protocol [12] that designates a single process as the leader among a ring of N processes by determining the maximum of the *unique_id*s transmitted by the processes.
- A non-stochastic version of *FMS* (Flexible Manufacturing System) from [8] that has N number of 3 different kinds of parts fitted on three machines.
- *Tiles* that models a N $\times$ M rectangular puzzle with movable tiles.
- *Tower of Hanoi* models D disks of different sizes that can move across three rods with constraints on sizes of the disks that can be placed on a rod.
- A modified version of *SatelliteMemory*, an ordinary PN from [1], where the constant arc cardinalities are converted to constant places and markings of these places are used to define the arc cardinalities.