Model Checking with Edge Valued Decision Diagrams

Pierre Roux, École Normale Supérieure de Lyon, France. pierre.roux@ens-lyon.fr
Radu I. Siminiceanu, National Institute of Aerospace, Hampton, Virginia, USA. radu@nianet.org

Abstract

We describe an algebra of Edge-Valued Decision Diagrams (EVMDDs) to encode arithmetic functions and its implementation in a model checking library. We provide efficient algorithms for manipulating EVMDDs and review the theoretical time complexity of these algorithms for all basic arithmetic and relational operators. We also demonstrate that the time complexity of the generic recursive algorithm for applying a binary operator on EVMDDs is no worse than that of Multi-Terminal Decision Diagrams.

We have implemented a new symbolic model checker with the intention to represent in one formalism the best techniques available at the moment across a spectrum of existing tools. Compared to the CUDD package, our tool is several orders of magnitude faster.

1 Introduction

Binary decision diagrams (BDD) [3] have revolutionized the reachability analysis and model checking technology. Arithmetic decision diagrams [2], also called Multi-Terminal Binary Decision Diagrams (MTBDD) [8] are the natural extension of regular BDDs to arithmetic functions. They take advantage of the symbolic encoding scheme of BDDs, but functions with large co-domains do not usually have a very compact representation because there are less chances for suffixes to be shared.

Edge-valued decision diagrams have been previously introduced, but only scarcely used. An early version, the edge valued binary decision diagrams (EVBDD) [11], is particularly useful when representing both arithmetic and logic functions, which is the case for discrete state model checking. However, EVBDD have only been applied to rather obscure applications: computing the probability spectrum and the Reed-Muller spectrum of (pseudo)-Boolean functions.

Binary Moment Diagrams [4] were designed to overcome the limitations of BDDs/EVBDDs when encoding multiplier functions. However, their efficiency seems to be limited only to this particular type of functions. A new canonization rule for edge-valued decision diagrams enabling them to encode functions in $\mathbb{Z} \cup \{+\infty\}$ was introduced in [6] along with EVMDDs, an extension to multi-way diagrams (MDD) [9], but, again, this was applied to a very specific task, of finding minimum length counterexamples for safety properties. Later, EVMDDs have been also used for partial reachability analysis.

In this paper we first present a theoretical comparison between EVMDDs and MTMDDs for building the transition relation of discrete state systems before dealing with an implementation in a model checker along with state-of-the-art algorithms for state space construction.

2 Background

2.1 Discrete-state Systems

A discrete–state model is a triple (S, S_0, T), where the discrete set S is the *potential state space* of the model; the set $S_0 \subseteq S$ contains the *initial states*; and $T : S \to 2^S$ is the *transition function* specifying which states can be reached from a given state in one step, which we extend to sets: $T(X) = \bigcup_{i \in X} T(i)$.

We consider structured systems modeled as a collection of K *submodels*. A (global) system state i is then a K-tuple $(i_K, \ldots, i_1)$, where i_k is the *local state* for submodel k, for $K \geq k \geq 1$, and S is given by $S_K \times \cdots \times S_1$, the cross–product of K local state spaces S_k, which we identify with $\{0, \ldots, n_k - 1\}$ since we assume that S is finite. The *(reachable) state space* $R \subseteq S$ is the smallest set containing S_0 and closed with respect to T, i.e. $R = S_0 \cup T(S_0) \cup T(T(S_0) \cup \cdots = T^*(S_0)$. Thus, R is the least fixpoint of function $X \mapsto S_0 \cup T(X)$.

2.2 Decision Diagrams

We discuss the extension of BDDs to integer variables, i.e., multi–valued decision diagrams (MDDs) [9]. We assume that the variables along any path from the root must follow the order $x_K, \ldots, x_1$. Ordered MDDs can be either *reduced* (no duplicate nodes and no node with all edges pointing to the same node, but edges possibly spanning multiple levels) or *quasi–reduced* (no duplicate nodes, and all edges spanning exactly one level), either form being *canonical*.

3 EVMDDs

Definition 1. *An* EVMDD *on a group* $(G,*)$*, is a pair* $A = \langle v, n\rangle$*, where* $v \in G$ *is the edge* value *also denoted as* $A.\mathrm{val}$ *and* n *is a* node *also denoted* $A.\mathrm{node}$.

A node *n is either the unique terminal node $\langle 0, e\rangle$ where e is the identity element of G, or a pair $\langle k, p\rangle$ where $1 \le k \le K$ and p is an array of edges of size n_k (cardinality of S_k). The first element of the pair will be denoted* $n.\mathrm{level}$ *and, when relevant, the i-th element in the array will be denoted by* $n[i]$.

Definition 2. *For a node n with* $n.\mathrm{level} = k$ *and* $(i_k, \ldots, i_1) \in S_k \times \cdots \times S_1$*, we define* $n(i_k, \ldots, i_1)$ *as* $n[i_k].\mathrm{val}$ *if* $n[i_k].\mathrm{node.level} = 0$ *and* $n[i_k].\mathrm{val} * n[i_k].\mathrm{node}(i_{n[i_k].\mathrm{node.level}}, \ldots, i_1)$ *otherwise.*

The function encoded *by an EVMDD A,* $f : S \to G, (i_K, \ldots, i_1) \mapsto A.\mathrm{val} * A.\mathrm{node}(i_{A.\mathrm{node.level}}, \ldots, i_1)$ *is the repetitive application of law $*$ on the edge values along the path from the root to the terminal node, corresponding to arcs i_k, for $K \ge k \ge 1$:*

Definition 3. *A* canonical node *is either the terminal node or a node n such that* $n[0].\mathrm{val} = e$.

A canonical *EVMDD contains only canonical nodes.*

It can be proved that any function f has a unique canonical EVMDD representation [6].

Examples of graph representations of EVMDDs are given in Figure 1.

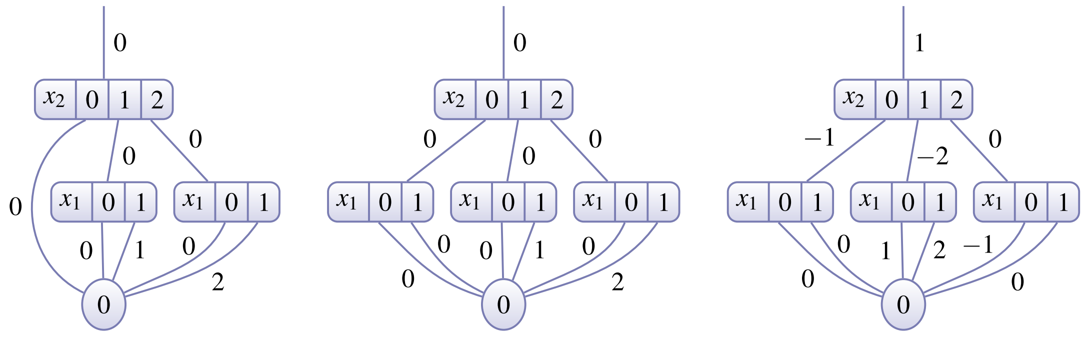

Figure 1: EVMDDs on $(\mathbb{Z},+)$ representing the same function $f : \{0,1,2\} \times \{0,1\} \to \mathbb{Z}, (x_2,x_1) \mapsto x_2 \cdot x_1$. The leftmost EVMDD is reduced while the others are quasi–reduced. The rightmost EVMDD is not canonical.

EVMDDs can be used when even the algebraic structure G is not a group. For example, [6] offers a canonization rule for $\mathbb{N} \cup \{+\infty\}$. Also, $(\mathbb{Z}, \times)$ that can be handled with the canonization rule "$\gcd\{n[i].\mathrm{val} \mid i \in S_{n.\mathrm{level}}\} = 1$ and $(n[0].\mathrm{val}, \ldots, n[n_{n.\mathrm{level}}].\mathrm{val}) \ge_{\mathrm{lex}} 0$".

4 EVMDDs compared to MTMDDs

MTBDDs are commonly used in model checking to build the transition relation of discrete-state systems. In this section we show that EVMDDs are at least as suited to that purpose and oftentimes significantly better. In the following, we choose, without loss of generality, $(G,*) = (\mathbb{Z},+)$.

4.1 Space Complexity

Theorem 1. *For any function f, the number of nodes of the EVMDD representing f is at most the number of nodes of the MTMDD representing the same function f.*[1]

4.2 Time complexity

Section 2 of [8] gives an algorithm to compute any binary operation on BDDs. The *apply* algorithm can be easily generalized to MDDs for any n-ary operator $\square_n$. It computes its result in time $\mathrm{O}\left(\prod_{i=1}^{n}|f_i|\right)$, where $|f_i|$ is the size (in nodes) of the MTMDD representing operand i.

Section 2.2 of [10] gives the equivalent *apply* algorithm for edge-valued decision diagrams.

Theorem 2. *The number of recursive calls of the generic* apply *algorithm for MTMDDs is equal to that for EVMDDs representing the same function [10].*

Hence, EVMDD computations are at least not worse than the MTMDD counterpart. However, particular operators $\square_n$ may enable much better algorithms on EVMDDs. Below is a synopsis of the basic algorithms to manipulate EVMDDs.

- Addition of constant ($f+c$): $\mathrm{O}(1)$.
- Multiplication with scalar ($f \times c$): $\mathrm{O}(|f|)$ [10].
- Addition ($f+g$): $\mathrm{O}(|f|\,|g|)$ [10].
- Remainder and Euclidean Division: $\mathrm{O}(|f|c)$.
- Minimum and Maximum: $O(|f|)$.
- Relational Operator with constant ($f < c$): not better in the worst case,
  but in practice the complexity can be improved, by using min and max.
- Relational Operators ($f < g$): can be computed as ($f - g < 0$).

4.3 Multiplication

As stated in [10], the result of a multiplication can have an EVMDD representation of exponential size in terms of the operands. For example, let S be $\{0,1\}^K$, $f : (x_K,\ldots,x_1) \mapsto \sum_{k=2}^{K} x_k 2^{k-2}$ and $g : (x_K,\ldots,x_1) \mapsto x_1$, f and g both have an EVMDD representation with $K+1$ nodes whereas $f{\cdot}g$ has 2^K nodes. Therefore, we cannot expect to find an algorithm with better worst-case complexity. However, the following equation, coming from the decomposition of $\langle v,n\rangle$ in $v+\langle 0,n\rangle$ and $\langle v',n'\rangle$ in $v'+\langle 0,n'\rangle$

$$\langle v,n\rangle \times \langle v',n'\rangle = vv' + v\langle 0,n'\rangle + v'\langle 0,n\rangle + \langle 0,n\rangle \times \langle 0,n'\rangle$$

[1]All proofs and algorithms are given in a technical report [12], to appear.

suggests an alternative algorithm.

The first product is an integer multiplication done in constant time. The next two are multiplications by a constant done in $\mathrm{O}(|f|)$ and $\mathrm{O}(|g|)$, respectively. The last one is done through recursive calls. The first addition takes constant time, the second one takes $\mathrm{O}(|f|\,|g|)$ and produce a result of size at most $|f|\,|g|$, hence a cost of $\mathrm{O}(|f|\,|g|\,|fg|)$ for the last addition. The recursive function is called $\mathrm{O}(|f|\,|g|)$ times, hence a final complexity of $\mathrm{O}\left(|f|^2\,|g|^2\,|fg|\right)$.

Although we were unable to theoretically compare this algorithm to the generic *apply* algorithm, it seems to perform far better on practical cases.

5 Implementation

Model size	Reachable states	CUDD (in s)	SMART (in s)	EVMDD (in s)
Dining philosophers				
100	4×10^{62}	11.42	1.49	0.03
200	2×10^{125}	3054.69	3.03	0.07
15000	2×10^{9404}	—	—	195.29
Round robin mutual exclusion protocol				
40	9×10^{13}	4.44	0.44	0.08
100	2×10^{32}	—	2.84	1.17
200	7×10^{62}	—	20.02	9.14
Slotted ring protocol				
10	8×10^{9}	1.16	0.19	0.01
20	2×10^{20}	—	0.71	0.04
200	8×10^{211}	—	412.27	25.97
Kanban assembly line				
15	4×10^{10}	80.43	3.41	0.01
20	8×10^{11}	2071.58	8.23	0.02
400	6×10^{25}	—	—	74.89
Knights problem				
5	6×10^{7}	1024.42	5.29	0.27
7	1×10^{15}	—	167.41	3.46
9	8×10^{24}	—	—	32.20
Randomized leader election protocol				
6	2×10^{6}	4.22	8.42	0.86
9	5×10^{9}	—	954.81	18.89
11	9×10^{11}	—	—	109.25

Table 1: Execution times for building state space using our library or CUDD (“—” means “$>$ 1hour”).

Symbolic model checkers, such as (Nu)SMV or SAL, are based on the library CUDD[1] which offers an efficient implementation of BDDs and MTBDDs. Our goal was to implement a new symbolic model checking library featuring EVMDDs for the transition relation construction and saturation[5] for state space generation. We also developed a basic model checking front-end to test the library and compare it to CUDD. Binaries and source code for both the EVMDD library and the model checker are available at `http://research.nianet.org/~radu/evmdd/`.

5.1 Encoding the Transition Relation

We represent the transition relation T as a disjunction of events which is well suited for globally–asynchronous locally–synchronous systems, where each event encodes some local transition. To avoid the expensive coding of lot of identities, we use the *full-identity reduction* from [7].

5.2 State Space Construction

For state space construction, we use the Saturation algorithm [5] instead of the classical breadth first search exploration. This heuristic often gives spectacular improvements when building the state spaces of globally–asynchronous locally–synchronous systems. This is certainly the major source of improvement of our implementation over existing BDD libraries.

5.3 Experimental Results

Our new tool comprises 7K lines of ANSI-C code for the library and 4K lines for the simple model checker that provides a common interface to both our library and CUDD. Table 1 shows execution times for building the state space on a suite of classical models. Programs to generate all models can be found in the `examples` folder of our source code distribution.

We collected the results on a Linux machine with Intel Core 2 processor, 1.2GHz, 1.5GB of memory.

Note that using other existing tools, such as NuSMV or SAL on these models, we get execution times of the same order of magnitude as with the CUDD interface of our tool.

Compared to the first implementation of saturation algorithm [5] in the tool SMART, our new implementation is always several (up to a few dozens) times faster. This is due to both the encoding of the transition relation and our simple C implementation in comparison to the object-oriented C++ version.

6 Conclusions and Future Work

We have studied the advantages of the EVMDD data structure over the widely used MTBDDs for the construction of transition relations of finite state systems and implemented them in a library, along with state-of-the-art algorithms for state space generation. We obtained execution times several orders of magnitude faster than the CUDD library and classical algorithms, with a reduced memory usage enabling to handle extremely large systems. Future work should focus primarily on integrating our library into the SAL model checker.

Our results show that symbolic model checking remains an efficient technique for analyzing globally–asynchronous locally–synchronous systems and significant improvements are still possible.

References

[1] Colorado University Decision Diagram library. `http://vlsi.colorado.edu/~fabio/CUDD/`.

[2] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *ICCAD*, pages 188–191, 1993.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

[4] R. E. Bryant and Y.-A. Chen. Verification of arithmetic functions with binary moment diagrams. Technical Report CS-94-160, CMU, 1994.

[5] 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, Genova, Italy, Apr. 2001. Springer.

[6] G. Ciardo and R. Siminiceanu. Using edge-valued decision diagrams for symbolic generation of shortest paths. In *Proc. FMCAD*, LNCS 2517, pages 256–273, Portland, OR, USA, Nov. 2002. Springer.

[7] G. Ciardo and J. Yu. Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In *Proc. CHARME*, October 2005.

[8] E. Clarke, M. Fujita, and X. Zhao. Application of multi-terminal binary decision diagrams. Technical report, IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, 1995.

[9] 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.

[10] Y.-T. Lai, M. Pedram, and B. K. Vrudhula. Formal verification using edge-valued binary decision diagrams. *IEEE Trans. Comput.*, 45:247–255, 1996.

[11] Y.-T. Lai and S. Sastry. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *DAC*, pages 608–613, 1992.

[12] P. Roux and R. Siminiceanu. Model checking with edge-valued decision diagrams. CR submitted, NASA, Langley Research Center, Hampton VA 23681-2199, USA, March 2010.