Efficient Manipulation of Logical Formulas as Decision Diagrams

Milán Mondok, Vince Molnár
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Email: mondok@mit.bme.hu, molnarv@mit.bme.hu

***Abstract*—Constraint solving and the manipulation of Satisfiability Modulo Theories (SMT) formulas is a fundamental task in symbolic model checking. SMT solvers have proven to be efficient tools in exploiting the high expressive power and flexibility offered by SMT formulas. Decision diagram based approaches have also gained popularity for their capability to represent all solutions in a compact way and are used in numerous efficient algorithms. However, there is a gap between these two approaches.**

In this paper, we present a novel data structure that can combine the flexibility of SMT formulas and the power of SMT solvers with the efficient representation of the solutions. This data structure is a blend of decision diagrams and SMT formulas: it allows us to handle logical formulas as decision diagrams, leveraging both the power of SMT solvers and the advantages of diagram representation. The compatibility with decision diagrams allows the integration of efficient algorithms working on the two different representations. When discussing the benefits of this approach, we also emphasize how the intersection operation - a common problem in constraint solving - can be carried out more efficiently using lazy evaluation. We can also build on the same advantage in transitive closure calculations.

***Index Terms*—SMT, decision diagram, symbolic, model checking**

I. Introduction

Constraint solving and the manipulation of logical formulas is a fundamental task in symbolic model checking. First order logic [2] is an expressive language widely used in computer science, but its satisfiablity is undecidable in the general case. Satisfiability Modulo Theories (SMT) [1] solvers offer efficient algorithms for practical subsets (theories) [2] of first order logic and have proven to be efficient tools in exploiting the high expressive power and flexibility offered by first order logic formulas. Decision diagram based approaches have also gained popularity for their capability to represent all solutions in a compact way and efficient algorithms [3] are known that manipulate the sets directly in the encoded form. However, there is a gap between these two approaches.

In this paper, we present a novel data structure called *substitution diagram*, that can combine the flexibility of first order logic formulas and the power of SMT solvers with the efficient representation of the solutions. This data structure is a blend of decision diagrams and logical formulas: it allows us to handle logical formulas as decision diagrams, leveraging both the power of SMT solvers and the advantages of diagram representation. The compatibility with decision diagrams allows the integration of efficient algorithms working on the two different representations. When discussing the benefits of this approach, we also emphasize how the intersection operation - a common problem in constraint solving - can be carried out more efficiently using lazy evaluation. We can also build on the same advantage in transitive closure calculations.

Supported by the **ÚNKP-23-3-I-BME-8** New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund

II. Background

In this section, we briefly discuss the fundamentals of two different symbolic approaches, namely *satisifiability modulo theories* and *decision diagrams*, that we build on in the later sections of this paper.

A. Satisifiability Modulo Theories

First order logic (FOL) [2] is a rich language that is commonly used in the context of formal verification to capture constraints and reason about system designs. It extends propositional logic with predicates, functions and quantifiers. Consequently, FOL formulas can not only evaluate to truth values, but to any abstract concept, e.g., integers, animals, names. The satisfiability of first order logic formulas is undecidable in the general case.

First order theories [2] formalize the structures of a domain to enable reasoning about them formally. These theories can for example capture the concepts of equality ($v_1 = v_2$), linear arithmetic ($2x \leq (y + 5)$), bit vectors ($(a >> b)\ \&\ c$) or arrays ($a[i] = b[0]$). While satisfiability in FOL is undecidable in general, it is decidable in many practical first order theories (or in their fragments).

The satisfiability modulo theories (SMT) [1] problem is to decide if a given formula is satisfiable in a given theory (or combination of theories). Since Boolean satisfiablity is already NP-complete, the SMT problem is typically NP-hard even in decidable theories. Despite this, more and more SMT solvers are known that can efficiently solve the SMT problem for a practical set of inputs. Solvers like Z3 [4] and cvc5 [5] are used as foundational building blocks across a wide range of applications, ranging from theorem proving and model checking to software testing.

In this work, we consider SMT formulas that can contain Boolean literals ($\top, \bot$), Boolean connectives ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$),

integer variables and literals, linear arithmetic ($+, -, *, <, >, =$), but the presented approach can work with any first order theory given a suitable SMT solver. Let the syntax of quantifier-free first order formulas be the following:

- the Boolean literals $\top$ and $\bot$ are formulas;
- the natural numbers $\mathbb{N}$ are formulas;
- variables $x \in V$ are formulas;
- if φ, ψ are formulas, then $\varphi \wedge \psi, \varphi \vee \psi, \neg\varphi, \varphi \rightarrow \psi, \varphi \leftrightarrow \psi$ are also formulas;
- if φ, ψ are formulas, then $\varphi + \psi, \varphi - \psi, \varphi * \psi, \varphi < \psi, \varphi > \psi, \varphi \leq \psi, \varphi \geq \psi, \varphi = \psi$ are also formulas;

Let $\mathcal{L}$ denote the set of first order logic formulas that can be generated with the rules above. Let $\varphi[v/\psi]$ denote the operation of substituting all appearances of the variable $v \in V$ in the formula $\varphi \in \mathcal{L}$ with the formula $\psi \in \mathcal{L}$. For example, $(x < 2 \wedge y = x + 1)[x/0]$ is $(0 < 2 \wedge y = 0 + 1)$.

B. Decision Diagrams

Decision diagrams offer a compressed representation for sets and relations. Binary decision diagrams (BDD) [3], which are essentially binary decision trees with all the identical subtrees merged are commonly used to represent Boolean functions. Multi-valued decision diagrams [7] generalize this idea and can be used to reason about variables with larger discrete domains, e.g. integers.

An ordered quasi-reduced multi-valued decision diagram (MDD) [8] over a set of variables V ($|V| = K$), a variable ordering, and domains D is a tuple $\mathit{MDD} = (\mathcal{V}, \mathit{lvl}, \mathit{children})$ where:

- $\mathcal{V} = \bigcup_{k=0}^{K} \mathcal{V}_k$ is the set of *nodes*, where items of $\mathcal{V}_0$ are the *terminal* nodes $\mathbf{1}$ and $\mathbf{0}$, the rest ($\mathcal{V}_{>0} = \mathcal{V} \setminus \mathcal{V}_0$) are *internal* nodes ($\mathcal{V}_i \cap \mathcal{V}_j = \emptyset$ if $i \neq j$);
- $\mathit{lvl} : \mathcal{V} \rightarrow \{0, 1, \ldots, K\}$ assigns non-negative *level numbers* to each node, associating them with variables according to the variable ordering (nodes in $\mathcal{V}_k = \{n \in \mathcal{V} \mid \mathit{lvl}(n) = k\}$ belong to variable x_k for $1 \leq k \leq K$ and are terminal nodes for $k = 0$);
- $\mathit{children} : \mathcal{V}_{>0} \times \mathbb{N} \rightarrow \mathcal{V}$ defines edges between nodes labeled with elements of $\mathbb{N}$ (denoted by $n[i] = \mathit{children}(n, i)$, $n[i]$ is left-associative), such that for each node $n \in \mathcal{V}_k$ ($k > 0$) and value $i \in D(x_k) : \mathit{lvl}(n[i]) = \mathit{lvl}(n) - 1$ or $n[i] = \mathbf{0}$; as well as $n[i] = \mathbf{0}$ if $i \notin D(x_k)$;
- for every pair of nodes $n_1, n_2 \in \mathcal{V}_{>0}$, if for all $i \in \mathbb{N} : n_1[i] = n_2[i]$, then $n_1 = n_2$, meaning the equivalence of two nodes is decided recursively through their outgoing edges and the equivalence of their children.

An MDD node $n \in \mathcal{V}_k$ encodes a set of vectors $S(n)$ over variables $V_{\leq k}$ such that for each $\vec{s} \in S(n)$ the value of $n[\vec{s}[k]] \cdots [\vec{s}[k]]$ (recursively indexing n with components of $\vec{s}$) is $\mathbf{1}$ and for all $\vec{s} \notin S(n)$ it is $\mathbf{0}$. When speaking about an MDD encoding the set $S(n)$, we mean the pair (MDD, n) and we refer to n as the root node of the MDD. The size of such an MDD is defined as the number of nodes reachable from the root node and is denoted by $|n|$.

Figure 1 shows an example for the graphical representation of an MDD. The circles represent internal nodes, while the squares represent terminal nodes. The edges in the figure correspond to the *children* edges. The terminal $\mathbf{0}$ node and all edges leading to it are omitted for readability. The MDD on the left encodes 3 vectors: $(x = 0, y = 0)$, $(x = 1, y = 1)$ and $(x = 2, y = 2)$.

An interesting property of decisions diagrams is that the number of the nodes does not grow proportionally with the size of the encoded set. The size of the MDD can even decrease when a new state is added because of the exploited regularities. This is illustrated in Fig. 1, where the MDD on the left encodes 3 vectors with 4 nodes, while the MDD on the right encodes 9 vectors with only 2 nodes.

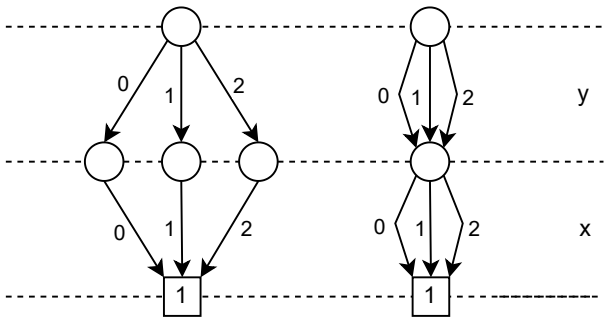

Fig. 1. An MDD with 4 nodes encoding 3 vectors (left) and and MDD with 2 nodes encoding 9 vectors (right).

III. Substitution Diagrams

We present a novel data structure called *substitution diagram*, which allows for lazy evaluation of SMT formulas in a decision diagram like representation. The idea behind substitution diagrams comes from the observation that first order logic formulas and the variable substitution operation ($\varphi[x/\psi]$) span a structure that is very similar to decision diagrams. A substitution diagram (SMDD) over a set of variables V ($|V| = K$), a variable ordering, and domains D is a tuple $\mathit{SMDD} = (\mathcal{V}, \mathit{lvl}, \mathit{children}, \mathit{semantics})$, where:

- $\mathcal{V} \subseteq \bigcup_{k=0}^{K} \mathcal{V}_k \subseteq (\mathcal{L} \cup (\mathcal{L} \times V))$ is the set of *nodes*, where $\mathcal{L}$ denotes the set of quantifier-free first order logic formulas over V', where $V \subseteq V'$ (this means that the formulas are allowed to contain additional variables compared to V). Items of $\mathcal{V}_0 \subseteq \mathcal{L}$ are the *terminal* nodes, which are first order logic formulas. *Internal* nodes $(\mathcal{V}_{>0} = \mathcal{V} \setminus \mathcal{V}_0) \subseteq \mathcal{L} \times V$ are formula-variable pairs;
- $\mathit{lvl} : \mathcal{V} \rightarrow \{0, 1, \ldots, K\}$ assigns non-negative *level numbers* to each node, associating them with variables according to the variable ordering. The nodes of level k are denoted with $\mathcal{V}_k = \{n \in \mathcal{V} \mid \mathit{lvl}(n) = k\}$. For all $1 \leq k \leq K$ and $n = (\varphi, x) \in \mathcal{V}_k$, $x = x_k$, meaning that the nodes of level k can only contain the variable associated with their level;
- $\mathit{semantics} : \mathcal{L} \rightarrow \mathcal{L}$ assigns semantically equivalent first order logic formulas to first order logic formulas. This can be thought of as an operation that brings the formulas to some normal form;

- *children* : $\mathcal{V}_{>0} \times \mathbb{N} \to \mathcal{V}$, define edges between nodes labeled with elements of $\mathbb{N}$ (denoted by $n[i] = \textit{children}(n, i)$, $n[i]$ is left-associative). The presence of the edges is defined as $n_1[i] = n_2$, iff $\textit{semantics}(\varphi_1[x_k/i]) = \varphi_2$, where $n_1 = (\varphi_1, x_k) \in \mathcal{V}_k$, $n_2 = (\varphi_2, x_{k-1}) \in \mathcal{V}_{k-1}$, $k > 1$, $i \in D(x_k)$, meaning an edge points from the node n_1 to the node n_2 with the label i if and only if substituting all appearances of the variable x_k in the expression of n_1 with the literal i and bringing this expression to a normal form results in the expression of n_2. In the case of $k = 1$, $n_1[i] = n_2$ iff $\textit{semantics}(\varphi_1[x_1/i]) = n_2$, where $n_1 = (\varphi_1, x_1) \in \mathcal{V}_1$, $n_2 \in \mathcal{V}_0$, $i \in D(x_1)$, the edge points to a terminal node in this case;
- for every terminal node $\varphi \in \mathcal{V}_0$, $\textit{semantics}(\varphi) = \varphi$, and for every internal node $(\varphi, x) \in \mathcal{V}_{>0}$, $\textit{semantics}(\varphi) = \varphi$, meaning the formulas of the nodes must be in the normal form;
- for every pair of nodes of the same level $n_1 = (\varphi_1, x_k) \in \mathcal{V}_k$, $n_2 = (\varphi_2, x_k) \in \mathcal{V}_k$, if $\textit{semantics}(\varphi_1) = \textit{semantics}(\varphi_2)$, then $\varphi_1 = \varphi_2$, meaning two nodes with different formulas that evaluate to the same normal form are not allowed on the same level. This means that contrary to decision diagrams, the equivalence of nodes in substitution diagrams is decided through the *semantics* function.

A node $n = (\varphi, x_k) \in \mathcal{V}_k$ encodes the set of satisfying partial assignments to its formula φ, such that for an assignment $(i_k, i_{k-1}, \ldots, i_0) \in D(x_k) \times D(x_{k-1}) \times \ldots \times D(x_0)$, $\varphi[x_k/i_k] \cdots [x_0/i_0] = \varphi_\top$ (i.e. substituting the values of the assignment into φ results in the formula $\varphi_\top$) iff the value of $n[i_k] \cdots [i_0]$ (recursively indexing n with the values of the assignment) is $\varphi_\top$.

Consider the substitution diagram in Fig. 2 as an example. We start the construction of the diagram top-down with the formula $(y = 0 \wedge x = y) \vee (y = 1 \wedge x \geq 0 \wedge x < y)$ and the top-level variable y. This formula can only be satisfied with two values for the variable y, 0 and 1 (which can be enumerated using an SMT solver), so the node is going to have two children. If we substitute the appearances of y with the literal 0, we get the formula $(0 = 0 \wedge x = 0) \vee (0 = 1 \wedge x \geq 0 \wedge x < 0)$. We can replace $0 = 0$ with $\top$ and remove it from the conjunction. Similarly, on the right side of the disjunction, we can replace $0 = 1$ with $\bot$ and replace the conjunction with $\bot$. After removing the right operand $\bot$ from the disjunction, we arrive at the formula $x = 0$ and create a new node for it. This formula can only be satisfied with the assignment $x = 0$, which results in the formula $\top$, so the node only has one outgoing edge that leads to the terminal $\top$ node.

Semantic or Syntactic Equivalence

Deciding whether two nodes are equivalent and can be merged is not straightforward in a substitution diagram. Consider the two nodes $x = 0$ and $x \geq 0 \wedge x < 1$ in Fig. 2. The two formulas are syntactically different, but both can only be satisfied with the model (assignment) $x = 0$. In case of an MDD, equivalence is decided semantically, i.e., through the outgoing edges and children of the two nodes. We can see that in the equivalent MDD, the two nodes are merged.

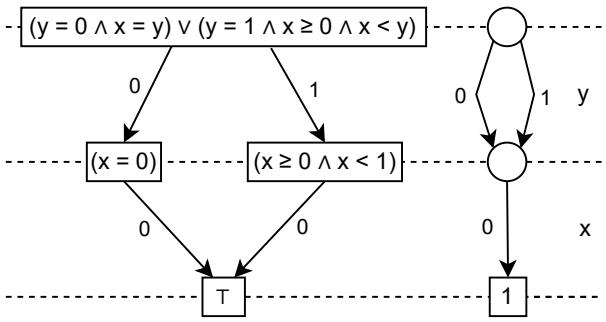

Fig. 2. Substitution diagram (left) and the equivalent MDD (right).

Semantic equivalence in the case of two SMT formulas means that exactly the same models satisfy them. However, exact semantic uniqueness among the nodes of the substitution would be too costly to maintain because it would require a solver call for each existing node on a level of the diagram whenever a new node is added. A good practical compromise could be to rely on syntactic equivalence and a low-cost transformation that brings the formulas to some sort of normal form before comparing them. We denote this transformation with the name *semantics*. The size of a substitution diagram is drastically affected by the precision of this transformation, however, in most cases the more precise the transformation is, the costlier it is to calculate it. In the extreme case where the *semantics* function is maximally precise (meaning it maps all semantically equivalent formulas to the same formula), the constructed substitution diagrams have the same number of nodes as the equivalent decision diagrams.

Lazy Evaluation

The most promising feature of substitution diagrams is that they allow for lazy evaluation of paths (solutions). When the diagram is initialized, only the root node (the node on the highest level) is stored, further nodes can be obtained by either enumerating children using an SMT solver or by querying the presence of either singular edges or paths consisting of multiple edges. Whenever the presence of an edge is confirmed, it can be cached, so that querying an SMT solver will not be required if the edge is needed again.

Operations on Substitution Diagrams

Substitution diagrams are intentionally compatible with decision diagrams in the sense that a substitution diagram $\textit{SMDD} = (\mathcal{V}, \textit{lvl}, \textit{children}, \textit{semantics})$ can be interpreted as a relaxed decision diagram $\textit{MDD} = (\mathcal{V}', \textit{lvl}', \textit{children}')$, where:

- $\mathcal{V}'_{>0} = \mathcal{V}_{>0}$, $\textit{lvl}' = \textit{lvl}$, $\textit{children}' = \textit{children}$, i.e. the internal nodes, the level numbering and the edges are analogous in the two representations;
- the terminal node $\mathbf{0} = \bot$, and all satisfiable terminal nodes $\varphi_\top \in \mathcal{V}_0$, $\varphi_\top \nrightarrow \bot$ can be interpreted as the terminal node $\mathbf{1}$;

- the semantic uniqueness of the nodes is not guaranteed, this depends on the precision (i.e. if it maps all semantically equivalent formulas to the same formula) of the *semantics* function.

This means that substitution diagrams can be used in operations that are defined on MDDs. This compatibility also allows us to mix the two representation modes in these operations such that one of the operands is an MDD, and the other operand is a substitution diagram. We can exploit the advantages offered by the lazy evaluation of substitution diagrams in these operations. A fundamental operation of constraint solving, the intersection operation returns the intersection of the sets encoded by two diagrams (i.e. the vectors that are contained by both diagrams). Typically, this is done by iterating over the edges of one of the diagrams and checking if the edge is present in the other diagram as well. When calculating the intersection of an SMDD and an MDD, by consciously choosing the MDD as the diagram that we iterate through and only checking the presence of those edges in the SMDD that are present in the MDD, we can avoid iterating over the potentially infinite solutions of the SMDD.

IV. Evaluation

We implemented a normal form transformation (*semantics* function) that (1) is carried out together with the variable substitution ($\varphi[x/\psi]$) operation such that it has extra information about the substituted variable, (2) uses simple rewriting rules to remove unnecessary operands from the expressions (for example replaces conjunctions like $\bot \wedge \varphi$ with $\bot$, or disjunctions like $\bot \vee \varphi$ with φ), (3) removes operators that can be expressed with other operators (for example replaces $\leq$ with a negation and $>$, or replaces implications of the form $\varphi \rightarrow \psi$ with $\varphi \vee \psi$), (4) is carried out purely syntactically without the use of an SMT solver.

To evaluate the precision of our transformation, we assembled a benchmark set of 10000 SMT formulas created from the transition relations of randomly generated symbolic transition system [6] models, which describe the relationship between the values of the variables before and after executing transitions, describing operations including but not limited to assignments, guards and various control structures (e.g. if-else, nondeterministic choice, sequence). The expressions can refer to 10 variables and the values of the variables are bound between 0 and 5 (which means that a formula can have at most $6^{10} = 60466176$ satisfying assignments). The expressions are generated such that the maximum depth of expression embedding is 13.

Out of the 10000 generated formulas, 3789 were satisfiable. For each satisfiable formula, we generated a random variable ordering and constructed a substitution diagram (SMDD) and a decision diagram (MDD) using the same variable ordering. We plotted the size difference between the two diagram representations on the scatterplot in Fig. 3. Each dot of the diagram corresponds to a satisfiable formula, with the number of nodes in the MDD representation plotted on the x axis and the number of nodes in the SMDD representation plotted on the y axis. Both axes are plotted using a logarithmic scale for better readability. The color of the dots corresponds to the number of vectors encoded by the diagrams (i.e. the number of possible satisfying assignments to the formula). The first, second and third quartiles of the number of vectors encoded by the formulas are, respectively, $q_1 = 1714$, $q_2 = 5184$ and $q_3 = 17140$, meaning more than half of the diagrams encoded more than 1700 and less than 17200 vectors. The largest number of vectors encoded by a diagram was 4.5 million. We can see that while the MDD is representation is always smaller or equal in size to the SMDD representation, the latter is not significantly larger in most cases. This indicates that our normal form transformation can provide a good approximation of semantic equality and should be studied further. Note that the transformation to an MDD representation is only possible in case of a finite number of solutions, but an SMDD can encode infinite solutions.

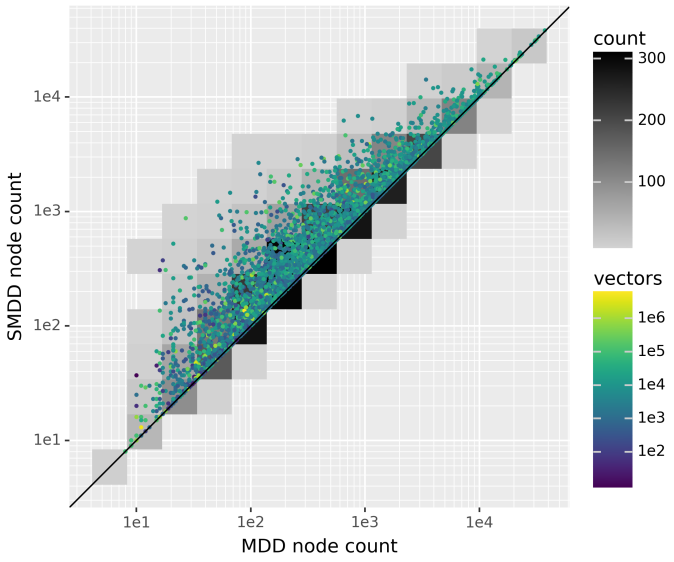

Fig. 3. SMDD and MDD node count for 3789 randomly generated SMT formulas. The color of the dots corresponds to the number of vectors encoded by the diagrams.

V. Conclusion

In this paper, after introducing the fundamentals of satisfiability modulo theories (SMT) and decision diagrams (MDD), we presented a novel data structure called substitution diagram (SMDD). This data structure combines the advantages of decision diagram representations with the power offered by SMT solvers to allow for efficient manipulation of logical formulas. We evaluated to compactness of substitution diagram representation on randomly generated SMT formulas and found that our approach can approximate decision diagrams promisingly.

As future work, we plan on exploring the possibilities of applying substitution diagrams in formal verification, more specifically in decision diagram based symbolic model check-

ing algorithms, as these could benefit from the flexibility offered by SMT formulas.

VI. Acknowledgements

We would like to thank Nóra Almási for her contributions to the development of substitution diagrams and Dániel Szekeres for the help he provided with the practical evaluation of our approach.

References

[1] Clark Barrett and Cesare Tinelli. *Satisfiability Modulo Theories*, pages 305–343. Springer International Publishing, Cham, 2018.

[2] Aaron R Bradley and Zohar Manna. *The calculus of computation: decision procedures with applications to verification*. Springer Science & Business Media, 2007.

[3] Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[4] Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient smt solver. In *2008 Tools and Algorithms for Construction and Analysis of Systems*, pages 337–340. Springer, Berlin, Heidelberg, March 2008.

[5] Haniel Barbosa et al. cvc5: A versatile and industrial-strength SMT solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.

[6] Ákos Hajdu, Tamás Tóth, András Vörös, and István Majzik. A configurable CEGAR framework with interpolation-based refinements. In Elvira Albert and Ivan Lanese, editors, *Formal Techniques for Distributed Objects, Components and Systems*, volume 9688 of *Lecture Notes in Computer Science*, pages 158–174. Springer, 2016.

[7] D.M. Miller and R. Drechsler. Implementing a multiple-valued decision diagram package. In *Proceedings. 1998 28th IEEE International Symposium on Multiple- Valued Logic (Cat. No.98CB36138)*, pages 52–57, 1998.

[8] Vince Molnár. *Extensions and Generalization of the Saturation Algorithm in Model Checking*. PhD thesis, Budapest University of Technology and Economics, February 2020.