Notes on Validated Model Counting
Version of March 24, 2022

Randal E. Bryant

Computer Science Department
Carnegie Mellon University, Pittsburgh, PA, United States

1 Notation

Consider Boolean formulas over a set of variables X. An assignment α is a function mapping each variable to truth value $\top$ (true) or $\bot$ (false). We can extend α to Boolean formulas in the normal way, such that $\alpha(F)$ will be $\top$ (respectively, $\bot$) if the evaluation of formula F yields $\top$ (resp., $\bot$) when its variables are assigned values according to α.

For Boolean formula F, we define its set of *models* $\mathcal{M}(F)$ as

$$\mathcal{M}(F) = \{\alpha | \alpha(F) = \top\} \tag{1}$$

The task of model counting is, given formula F, to determine the size of its set of models $\mu(F) = |\mathcal{M}(F)|$. Ideally, this should be done without actually enumerating the set.

For assignment α and a Boolean formula E over X, we use the notation $\alpha[E/x]$ to denote the assignment α', such that $\alpha'(y) = \alpha(y)$ for all $y \neq x$ and $\alpha'(x) = \alpha(E)$. In particular, the notation $\alpha[\overline{x}/x]$ indicates the assignment in which the value assigned to x is complemented, while others remain unchanged.

A Boolean formula F is said to be *independent* of variable x if every $\alpha \in \mathcal{M}(F)$ has $\alpha[\overline{x}/x] \in \mathcal{M}(F)$.

The following lemmas describe the primary abstractions used to perform model counting without enumerating all solutions.

Lemma 1. *For any formula F:*

$$\mu(\neg F) = 2^n - \mu(F)$$

Lemma 2. *For a partitioning of the variables X into disjoint sets X_1 and X_2, and for formulas F_1 defined over X_1 and F_2 defined over X_2:*

$$\mu(F_1 \wedge F_2) = \mu(F_1) \cdot \mu(F_2)$$

Lemma 3. *For formulas F_1 and F_2 such that $\mathcal{M}(F_1) \cap \mathcal{M}(F_2) = \emptyset$:*

$$\mu(F_1 \vee F_2) = \mu(F_1) + \mu(F_2)$$

2 ITE Operation

We consider a single Boolean operation, known as "If-Then-Else," or simply "ITE." For Boolean values a, b, c, the operation is defined as $\mathit{ITE}(a, b, c) = (a \wedge b) \vee (\neg a \wedge c)$. This single operation can be used to express several common Boolean operations:

$$\begin{aligned} \neg a &= \mathit{ITE}(a, \bot, \top) \\ a \wedge b &= \mathit{ITE}(a, b, \bot) \\ a \vee b &= \mathit{ITE}(a, \top, b) \\ a \rightarrow b &= \mathit{ITE}(a, b, \top) \end{aligned}$$

For variable x and Boolean formulas F and G, the set of models for $\mathit{ITE}(x, F, G)$ is given by the formula:

$$\mathcal{M}(\mathit{ITE}(x, F, G)) = \{\alpha \in \mathcal{M}(F) | \alpha(x) = \top\} \cup \{\alpha \in \mathcal{M}(G) | \alpha(x) = \bot\} \tag{2}$$

3 ITE Graphs

An *ITE graph* describes a combinational logic circuit constructed from two-input multiplexors. It is defined as a directed acyclic graph with three node types:

Constant: Corresponds to value $\top$ or $\bot$. A constant node has no incoming arcs.
Input: Corresponds to one of the variables in X. A variable node has no incoming arcs.
Operator: Represents an application of the *ITE* operation. An operator node has three incoming arcs, labeled **I**, **T**, and **E**, corresponding to the three arguments of the *ITE* operation.

When describing ITE graphs, we refer to constant and input nodes by their associated value or variable. An operator node is described by an expression of the form $\mathit{ITE}(v_i, v_t, v_e)$, where v_i, v_t, and v_e are the nodes corresponding to the **I**, **T**, and **E** inputs, respectively.

We can define the function D mapping each node v in an ITE graph to the set of variables on which it logically depends. This can be expressed recursively as

$$D(\top) = \emptyset \tag{3}$$
$$D(\bot) = \emptyset \tag{4}$$
$$D(x) = \{x\} \tag{5}$$
$$D(\mathit{ITE}(x, v_t, v_e)) = D(v_i) \cup D(v_t) \cup D(v_e) \tag{6}$$

Lemma 4. *Any node v in an ITE graph is independent of any variable $y \in X$ such that $y \notin D(v)$.*

Proof: Follows by the recursive definition of D (3–6) and by (2).

A *variable-partitioned* ITE (VPITE) graph is one satisfying the property that for every operator node $v = \mathit{ITE}(v_i, v_t, v_e)$, the dependency set of its **I** input most be disjoint from those of its **T** and **E** inputs. That is, both $D(v_i) \cap D(v_t) = \emptyset$ and $D(v_i) \cap D(v_e) = \emptyset$.

4 Model Counting VPITE Graphs

While model counting is a difficult problem for arbitrary formulas, it readily be computed when F takes the form of a VPITE graph.

Theorem 1. *Letting $n = |X|$, the following recursive formula holds for any node in a VPITE graph:*

$$\mu(\top) = 2^n \tag{7}$$

$$\mu(\bot) = 0 \tag{8}$$

$$\mu(x) = 2^{n-1} \tag{9}$$

$$\mu[\mathit{ITE}(v_i, v_t, v_e)] = \frac{\mu(v_i) \cdot \mu(v_t) + [2^n - \mu(v_i)] \cdot \mu(v_e)}{2^n} \tag{10}$$

Proof: The key result here is (10). It follows from Lemmas 1–3, along with the definition of the ITE operator.

5 Relation to Other Representations

In general, there is a trade-off between the compactness of different Boolean function representations versus the ease with which different properties can readily be computed. For example, Boolean formulas and Boolean circuits can be very compact, but basic properties of these representations, such as satisfiability, equivalence testing, and model counting are NP-hard. On the other hand all of these properties can be computed in polynomial time for reduced, orderd binary decision diagrams (ROBDDs) [3], but they can grow exponentially in relation to the representation of the function as a formula or circuit. ITE graphs lie somewhere between these two extremes. Here we examine how they relate to other representations and the implications of these relations.

5.1 Binary Decision Diagrams

Binary Decision Diagrams [1] also represent Boolean formulas as directed graphs consisting of ITE operators. They can be seen as equivalent to a class of graphs we will call *input-controlled* ITE graphs. These are ones for which the **I** input to every operator node is a variable.

If we combine the properties of variable partitioned and input control, we arrive at the representation known as *Free* BDDs [5]. If we further restrict the variables in a variable-controlled, input-partitioned ITE graph to obey the same ordering along all paths formed in the graph, we arrive at OBDDs. An OBDD can be reduced by a simple set of rules to give an ROBDD representation, and this form is canonical [3].

5.2 Sentental Decision Diagrams

Darwiche uses the Sentential Decision Diagram (SDD) representation [4] as his framework for several model counting programs. An SDD can be seen as a combination of 1) a generalization of VPITE graphs consisting of k-input multiplexors, and 2) an ordering restriction on the variables. That is, each operator node in an SDD is defined to have k data inputs and k control inputs, where the Boolean functions corresponding to the control inputs $p_1, p_2, \ldots, p_k$ satisfy $p_i \wedge p_j = \perp$ for $1 \leq i < j \leq k$ and $\bigvee_{i=1,k} p_i = \top$. The k data inputs represent Boolean functions $s_1, s_2, \ldots, s_k$. In addition, the dependency sets for the control and data inputs must be disjoint: $D(p_i) \cap D(s_j) = \emptyset$ for $1 \leq i, j \leq k$. Furthermore, the ordering of the variables in an SDD must obey a hierarchical decomposition defined by a tree structure known as a *vtree*. Indeed, an SDD can be seen as equivalent to a generalization of BDDs devised by McMillan, which he termed *tree* BDDs [6].

The graph part of an SDD can be readily translated into a VPITE graph by expanding each k-input node into $k-1$ ITE nodes, giving an output value defined as:

$$\mathit{ITE}(p_1, s_1, \mathit{ITE}(p_2, s_2, \mathit{ITE}(\cdots \mathit{ITE}(p_{k-1}, s_{k-1}, s_k)\cdots))$$

If we express the size of an SDD as the number of branches, this translation from SDD to ITE graph will expand the size by a constant factor.

6 Probabilistic Equivalence Checking

One further feature of VPITE graphs is that there is a randomized polynomial time (RP) algorithm for determining whether two graphs represent equivalent Boolean functions. This algorithm generalizes the RP equivalence-testing algorithm for free BDDs described by Blum, Chandra, and Wegman [2].

References

1. Akers, S.B.: Binary decision diagrams. IEEE Transactions on Computers **27**(6), 509–516 (June 1978)
2. Blum, M., Chandra, A.K., Wegman, M.N.: Equivalence of free Boolean graphs can be decided probabilistically in polynomial time. Information Processing Letters **10**(2), 80–82 (18 March 1980)
3. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. Computers **35**(8), 677–691 (1986)
4. Darwiche, A.: SDD: A new canonical representation of propositional knowledge bases. In: International Joint Conference on Artificial Intelligence. pp. 819–826 (2011)
5. Gergov, J., Meinel, C.: Efficient Boolean manipulation with OBDD's can be extended to FBDD's. IEEE Transactions on Computers **43**(10), 1197–1209 (October 1994)
6. McMillan, K.L.: Hierarchical representations of discrete functions, with applications to model checking. In: Computer-Aided Verification (CAV). LNCS, vol. 818, pp. 41–54 (1994)