Notes on Validated Model Counting
Version of April 8, 2022

Randal E. Bryant

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

1 Notation

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean variables. An *assignment* is a function α assigning Boolean values to the variables: $\alpha : X \rightarrow \{0, 1\}$. We can also view an assignment as a set of *literals* $\{l_1, l_2, \ldots, l_n\}$, where each literal l_i is either x_i or $\overline{x}_i$, corresponding to the assignments $\alpha(x_i)$ = 1 or 0, respectively.

1.1 Boolean Functions

A *Boolean function* $f : 2^X \rightarrow \{0, 1\}$ can be characterized by the set of assignments for which the function evaluates to 1: $\mathcal{M}(f) = \{\alpha | f(\alpha) = 1\}$. Let **1** denote the Boolean function that assigns value 1 to every assignment, and **0** denote the assignment that assigns value 0 to every assignment. These are characterized by the universal and empty assignment sets, respectively.

From this we can define the *complement* of function f as the function $\neg f$ such that $\mathcal{M}(\neg f) = \{\alpha | f(\alpha) = 0\}$. We can also define the conjunction and disjunction operations over functions f_1 and f_2 as characterized by the sets $\mathcal{M}(f_1 \wedge f_2) = \mathcal{M}(f_1) \cap \mathcal{M}(f_2)$ and $\mathcal{M}(f_1 \vee f_2) = \mathcal{M}(f_1) \cup \mathcal{M}(f_2)$.

For assignment α and a Boolean formula E over X, we use the notation $\alpha[E/x_i]$ to denote the assignment α', such that $\alpha'(x_j) = \alpha(x_j)$ for all $j \neq i$ and $\alpha'(x_i) = \alpha(E)$, where $\alpha(E)$ is the value obtained by evaluating formula E with each variable assigned the value given by α. In particular, the notation $\alpha[\overline{x_i}/x_i]$ indicates the assignment in which the value assigned to x_i is complemented, while others remain unchanged.

A Boolean function f is said to be *independent* of variable x_i if every $\alpha \in \mathcal{M}(f)$ has $\alpha[\overline{x_i}/x_i] \in \mathcal{M}(f)$. The *dependency set* of f, denoted $D(f)$ consists of all variables x_i for which f is *not* independent.

1.2 Separable Cost Functions

Let $\mathcal{Z}$ denote the elements of a commutative ring. A *separable cost function* $\sigma : X \rightarrow \mathcal{Z}$ assigns a value from the ring to each variable. We extend this function by defining the cost of literal $\overline{x}_i$ as $\sigma(\overline{x}_i) = 1 - \sigma(x_i)$, the cost of an assignment as $\sigma(\alpha) = \prod_{l_i \in \alpha} \sigma(l_i)$, and the cost of a function f as $\sigma(f) = \sum_{\alpha \in \mathcal{M}(f)} \sigma(\alpha)$.

Example 1: Let $\mathcal{Z}$ be the set of rational numbers and $\sigma(x_i) = 1/2$ for all variables x_i. The cost of every assignment will be $1/2^n$, and the cost of a function will be its

density, denoted $\rho(f)$. That is $\rho(f) \in [0,1]$ will be the fraction of assignments for which the function evaluates to 1. Given an ability to compute the density of function f, we can scale this by 2^n to compute $|\mathcal{M}(f)|$, the core task of model counting. Using density as our metric has the advantage that it does not vary when the function is embedded in a larger domain $X' \supseteq X$. As a variant, we can support some forms of weighted model counting by assigning weights of the form $\sigma(x_i) = w_i$, where $0 \leq w_i \leq 1$ for every variable x_i.

Example 2: Let $\mathcal{Z}$ be a field with $|\mathcal{Z}| > 2n$, and let $\mathcal{H}$ be the set of functions mapping elements of X to elements of $\mathcal{Z}$. For two distinct functions f_1 and f_2 and a randomly chosen $h \in \mathcal{H}$, the probability that $h(f_1) = h(f_2)$ will be at most $2^n/|\mathcal{Z}| < 1/2$. Therefore, these functions can be used as part of a randomized algorithm for equivalence testing [1].

1.3 Computing Cost Functions

Three key properties of separable cost functions makes it possible, in some cases, to compute the cost of a Boolean formula without enumerating all of its satisfying solutions.

Lemma 1 (Complementation). *For separable cost function σ and Boolean function f: $\sigma(\neg f) = 1 - \sigma(f)$.*

Lemma 2 (Variable-Partitioned Conjunction). *For separable cost function σ and Boolean functions f_1 and f_2 such that $D(f_1) \cap D(f_2) = \emptyset$: $\sigma(f_1 \wedge f_2) = \sigma(f_1) \cdot \sigma(f_2)$.*

We use the notation $f_1 \wedge_{\mathrm{pv}} f_2$ to denote the conjunction of f_1 and f_2 under the condition that f_1 and f_2 are defined over disjoint sets of variables.

Lemma 3 (Assignment-Partitioned Disjunction). *For separable cost function σ and Boolean functions f_1 and f_2 such that $\mathcal{M}(f_1) \cap \mathcal{M}(f_2) = \emptyset$: $\sigma(f_1 \vee f_2) = \sigma(f_1) + \sigma(f_2)$.*

We use the notation $f_1 \vee_{\mathrm{pa}} f_2$ to denote the disjunction of f_1 and f_2 under the condition that f_1 and f_2 hold for mutually exclusive assignments.

2 Proof Framework for Cost Functions

We introduce the CRAT clausal proof framework for generating a checkable proof of the valuation of a Boolean formula, given in conjunctive normal form, according to a specified cost function. A CRAT proof provides two capabilities: 1) a schema for computing a cost function using the ring operations, and 2) a proof that this schema accurately characterizes the cost function for the input formula. A proof can be either *function independent*, meaning that it holds for any cost function, or *function dependent*, meaning that it holds for only for a specific cost function.

The CRAT format draws its inspiration from the QRAT format for quantified Boolean formulas (QBF). The following are its key properties:

– Clause addition is allowed, as long as either 1) the new clause is blocked, or 2) it satisfies the RAT property with respect to a sequence of earlier antecedent clauses.
– Extension variables can be introduced only according to the operations $\wedge_{\mathrm{pv}}$, $\vee_{\mathrm{pa}}$, and *abstraction*
  - The checker tracks the dependency set for every input and extension variable. When an extension variable is introduced based on the $\wedge_{\mathrm{pv}}$ operation, the dependency sets of its arguments must be disjoint.
  - When an extension variable is introduced based on the $\vee_{\mathrm{pa}}$ operation, the proof step must cite earlier steps providing a RUP proof that the two arguments are mutually exclusive.
  - The *abstraction* operation allows an extension variable to serve as an alias for one of more other variables. Use of this rule renders the proof function dependent.
  - Boolean complement is provided implicitly by allowing the arguments of the extension operations to be literals and not just variables.
– Unlike DRAT, the deletion of a clause requires showing that this clause is implied by one or more other clauses.
– Unlike QRAT, it need not support universal quantification.

A CRAT proof follows the same general form as a QRAT satisfaction proof. That is, starting with the set of input clauses it produces a sequence of steps that both add and delete clauses. At the final step, there must be only a single unit clause consisting of some variable or its complement. Except for trivial cases, the final variable will be an extension variable. The sequence of extension operations define the schema for computing the cost function. That is, a value for the cost function is computed for every literal according to the rules of Lemmas 1–3. For every abstraction step, all variables having the indicated extension variable as their alias must evaluate to the same value.

References

1. 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)