Making IP = PSPACE Practical: Efficient Interactive Protocols for BDD Algorithms

Eszter Couillard[1], Philipp Czerner[1], Javier Esparza[1], Rupak Majumdar[2]

{couillar, czerner, esparza}@in.tum.de, rupak@mpi-sws.org
[1] Department of Informatics, TU München, Germany
[2] Max Planck Institute for Software Systems, Germany

Abstract. We show that interactive protocols between a prover and a verifier, a well-known tool of complexity theory, can be used in practice to certify the correctness of automated reasoning tools.

Theoretically, interactive protocols exist for all PSPACE problems. The verifier of a protocol checks the prover's answer to a problem instance in probabilistic polynomial time, with polynomially many bits of communication, and with exponentially small probability of error. (The prover may need exponential time.) Existing interactive protocols are not used in practice because their provers use naive algorithms, inefficient even for small instances, that are incompatible with practical implementations of automated reasoning.

We bridge the gap between theory and practice by means of an interactive protocol whose prover uses BDDs. We consider the problem of counting the number of assignments to a QBF instance (#CP), which has a natural BDD-based algorithm. We give an interactive protocol for #CP whose prover is implemented on top of an extended BDD library. The prover has only a linear overhead in computation time over the natural algorithm.

We have implemented our protocol in blic, a certifying tool for #CP. Experiments on standard QBF benchmarks show that blic is competitive with state-of-the-art QBF-solvers. The run time of the verifier is negligible. While loss of absolute certainty can be concerning, the error probability in our experiments is at most 10^{-10} and reduces to 10^{-10k} by repeating the verification k times.

1. Introduction

Automated reasoning tools often underlie our assertions about the correctness of critical hardware and software components. In recent years, the scope and scalability of these techniques have grown significantly.

Automated reasoning tools are not immune to bugs. If we are to trust their verdict, it is important that they provide evidence of their correct behaviour. A substantial amount of research has gone into proof-producing automated reasoning tools [15, 22, 21, 13, 4]. These works define a notion of "correctness certificate" suitable for the reasoning problem at hand, and adapt the reasoning engine to produce independently checkable certificates. For example, SAT solvers produce either a satisfying assignment or a proof of unsatisfiability in some proof system, e.g. resolution (see [15] for a survey). Extending such certificates beyond boolean SAT is an active area of current research [4, 17, 23, 28, 3].

In the worst case, the size of certificates grows exponentially in the size of the input, even for boolean unsatisfiability (unless $\mathsf{NP} = \mathsf{coNP}$). If users have limited computational or communication resources, transferring and checking large certificates becomes a burden. Large certificates are not just a theoretical curiosity. In practice, resolution proofs for complex SAT problems may run to petabytes [14]. Ideally, we would prefer "small" certificates (polynomial in the size of the input) which can be checked independently in polynomial time.

The $\mathsf{IP} = \mathsf{PSPACE}$ theorem proves that certification with polynomial verification time is possible for any problem in PSPACE, provided one trades off absolute certainty for certainty with high probability [26]. The complexity class IP consists of those languages for which there is a polynomial-round, complete and sound *interactive protocol* [12, 2, 19, 1]—a sequence of interactions between a (computationally unbounded) prover and a (computationally bounded) verifier after which the verifier decides whether the prover correctly performed a computation. The protocol is complete if, whenever an input belongs to the language, there is an *honest prover* who can convince a polynomial-time randomised verifier in a polynomial number of rounds. The protocol is sound if, whenever an input does not belong to the language, the Verifier will reject the input with high probability — no matter what certificates are provided to the Verifier. That is, a "Prover" cannot fool the certification process.

Since every language in PSPACE has an interactive protocol, there are interactive protocols for UNSAT, QBF, *counting* QBF, safety verification of concurrent state machines, etc. Observe that the prover of a protocol may perform exponential time computations (which is unavoidable unless $\mathsf{P} = \mathsf{PSPACE}$), but the verifier only requires polynomial time in the original input.

If interactive protocols provide a foundation for small and efficiently verifiable certificates (at least for problems in PSPACE), why are they not in widespread practice? We believe the reason to be the following: for asymptotic complexity purposes, it suffices to use honest provers with best-case exponential complexity that naively enumerate all possibilities. Such provers are incompatible with automated reasoning tools, which use more sophisticated data structures and heuristics to scale to real-world examples. So we need to make *practical algorithms* for automated reasoning *efficiently certifying*. We call

an algorithm *efficiently certifying* if, in addition to computing the output, it can execute the steps of an honest prover in an interactive protocol with only polynomial overhead over its running time.

In this paper, we show that algorithms using reduced ordered binary decision diagrams (henceforth called BDDs) [9] can be made efficiently certifying. We consider #CP, the problem of computing the number of satisfying assignments of a *circuit with partial evaluation (CP)*. Besides boolean nodes, a CP contains *partial evaluation* nodes $\pi_{[x:=\mathsf{false}]}$ (resp., $\pi_{[x:=\mathsf{true}]}$) that take a boolean predicate as input, say φ, and output the result of setting x to false (resp., true) in φ. #CP generalises SAT, QBF, and *counting* SAT (#SAT), and has a natural algorithm using BDDs: Compute BDDs for each node of the circuit in topological order, and count the accepting paths of the final BDD.

The theoretical part of the paper proceeds in two steps. First, we present CPCertify, a complete and sound interactive protocol for #CP. CPCertify is similar to the SumCheck protocol [19]. It involves encoding boolean formulas as polynomials over a finite field. The prover is responsible for producing certain polynomials from the original circuit and evaluating them at points of the field chosen by the verifier. These polynomials are either multilinear (all exponents are at most 1) or quadratic (at most 2).

Second, we show that an honest prover in CPCertify can be implemented on top of a suitably extended BDD library. The run times of the certifying BDD algorithms are only a constant overhead over the computation time without certification—they depend linearly on the total number of nodes of the intermediate BDDs computed by the prover to solve the #CP instance. We use two key insights. The first is an encoding of multilinear polynomials as BDDs; we show that the intermediate BDDs represent all the multilinear polynomials a prover needs during the run of CPCertify. The second shows that the quadratic polynomials correspond to *intermediate steps* during the computation of the intermediate BDDs. We extend BDDs with additional "book-keeping" nodes that allow the prover to also compute the quadratic polynomials while solving the problem. So computing the polynomials required by CPCertify has *zero* additional cost; the only overhead is the cost of evaluating the polynomials at the field points chosen by the verifier.

We have implemented a certifying #CP solver based on our extended BDD library. Our experiments show that the solver is competitive with state-of-the-art non-certifying QBF solvers, and can outperform certifying QBF solvers based on BDDs. The number of bytes exchanged between the prover and the verifier are an order of magnitude smaller, and Verifier's run time several orders of magnitude smaller, than current encodings of QBF proofs, while bounding the error probability to below 10^{-10}. Thus, our results open the way for practically efficient, probabilistic certification of automated reasoning problems using interactive protocols.

Additional Related Work. Proof systems for SAT and QBF remain an active area of research—both in theoretical proof complexity and in practical tool development. Jussila, Sinz, and Biere [16, 27] showed how to extract extended resolution proofs from BDD operations. This is the basis for proof-producing SAT and QBF solvers based on BDDs [8, 7, 6]. As in our work, the proof uses intermediate nodes produced in the

construction of the BDD operations. We focus on interactive certification instead of extended resolution proofs, which can be exponentially larger than the input formula.

Recently, Luo et al. [20] consider the problem of providing *zero-knowledge* proofs of unsatisfiability, a motivation similar but not equal to ours. Their techniques require the verifier to work in time polynomial in the proof, which can be exponentially bigger than the input formula. In contrast, the verifier of CPCertify runs in polynomial time in the input. Since any language in PSPACE has a zero knowledge proof [5], our protocol can in principle be made zero knowledge. Whether that system scales in practice is left for future work.

2. Preliminaries

The Class IP. An *interactive protocol* between a *Prover* and a *Verifier* consists of a sequence of interactions in which a Verifier asks questions to a Prover, receives responses to the questions, and must ultimately decide if a common input x belongs to a language. The computational power of the Prover is unbounded but the Verifier is a randomised, polynomial-time algorithm.

Formally, let P, V denote (deterministic) Turing machines.

We say that $(r; m_1, ..., m_{2k})$ is a *k-round interaction*, with $r, m_1, ..., m_{2k} \in \{0,1\}^*$, if $m_{i+1} = V(r, m_1, ..., m_i)$ for even i and $m_{i+1} = P(m_1, ..., m_i)$ for odd i. We think of r as an additional sequence of bits given to Verifier V that is chosen randomly. The *output* $\text{out}(P, V)(x, r, k)$ is defined as m_{2k}, where $(r; m_1, ..., m_{2k})$ is the unique k-round interaction with $m_1 = x$.

A language L belongs to IP if there exist some V, P_H and polynomials p_1, p_2, p_3, s.t. $V(r, x, m_2, ..., m_i)$ runs in time $p_1(|x|)$ for all $r, x, m_2, ..., m_i$, and, for each x and an $r \in \{0,1\}^{p_2(|x|)}$ chosen uniformly at random:

1. (*Completeness*) $x \in L$ implies $\text{out}(P_H, V)(x, r, p_3(|x|)) = 1$ with probability 1, and
2. (*Soundness*) $x \notin L$ implies that for all P we have $\text{out}(P, V)(x, r, p_3(|x|)) = 1$ with probability at most $2^{-|x|}$.

Intuitively, in an interactive protocol, a computationally unbounded Prover interacts with a randomised polynomial-time Verifier for k rounds. In each round, Verifier sends probabilistic "challenges" to Prover, based on the input and the answers to prior challenges, and receives answers from Prover. At the end of k rounds, Verifier decides to accept or reject the input. The completeness property ensures that if the input belongs to the language L, then there is an "honest" Prover P_H who can always convince Verifier that indeed $x \in L$. If the input does not belong to the language, then the soundness property ensures that Verifier rejects the input with high probability no matter how a (dishonest) Prover tries to convince them.

It is known that IP = PSPACE [19, 26], that is, every language in PSPACE has a polynomial-round interactive protocol. The proof exhibits an interactive protocol for the language QBF of true quantified boolean formulae; in particular, the honest Prover is a

polynomial space, exponential time algorithm that uses a truth table representation of the formula to implement the protocol.

Polynomials. Interactive protocols make extensive use of polynomials over some prime finite field $\mathbb{F}$.

Let X be a finite set of variables. We use $x, y, z, ...$ for variables and $p, q, ...$ for polynomials. When we write a polynomial explicitly, we write it in brackets, e.g. $[3xy - z^2]$. We write $\mathbf{1}$ and $\mathbf{0}$ for the polynomials $[1]$ and $[0]$, respectively. We use the following operations on polynomials:

- *Sum, difference, and product.* Denoted $p + q$, $p - q$, $p \cdot q$, and defined as usual. For example, $[3xy - z^2] + [z^2 + yz] = [3xy + yz]$ and $[x + y] \cdot [x - y] = [x^2 - y^2]$.
- *Partial evaluation.* Denoted $\pi_{[x:=a]}\, p$, it returns the result of setting variable x to the field element a in the polynomial p, e.g. $\pi_{[x:=5]}[3xy - z^2] = [15y - z^2]$.
- *Degree reduction.* Denoted $\delta_x\, p$. It reduces the degree of x in all monomials of the polynomial to 1. For example, $\delta_x[x^3y + 3x^2 + 7z^2] = [xy + 3x + 7z^2]$.

A *(partial) assignment* is a (partial) mapping $\sigma : X \to \mathbb{F}$. We write $\Pi_\sigma\, p$ for $\pi_{[x_1:=\sigma(x_1)]} \ldots \pi_{[x_k:=\sigma(x_k)]}\, p$, where $x_1, ..., x_k$ are the variables for which σ is defined. Additionally, we call σ *binary* if $\sigma(x) \in \{0, 1\}$ for each $x \in X$.

Binary and multilinear polynomials. A polynomial is *multilinear in x* if the degree of x in p is 0 or 1. A polynomial is *multilinear* if it is multilinear in all its variables. For example, $[xy - y^2]$ is multilinear in x but not in y, and $[3xy - 2zy]$ is multilinear. A polynomial p is *binary* if $\Pi_\sigma\, p \in \{\mathbf{0}, \mathbf{1}\}$ for every binary assignment σ. Two polynomials p, q are *binary equivalent*, denoted $p \equiv_b q$, if $\Pi_\sigma\, p = \Pi_\sigma\, q$ for every binary assignment σ. (Note that non-binary polynomials can be binary equivalent.)

3. Circuits with Partial Evaluation

We introduce circuits with partial evaluation (CP), a compact representation of quantified boolean formulae, and formulate #CP, the problem of counting the number of satisfying assignments of a CP. #CP generalises QBF, the satisfiability problem for quantified boolean formulas. Figure 1 shows an example of a CP. Informally, it is a directed acyclic graph whose nodes are labelled with variables, boolean operators, or *partial evaluation operators* $\pi_{[x:=b]}$. Intuitively, $\pi_{[x:=b]}\varphi$ sets the variable x to the truth value b in the formula φ. In this way, each node of a circuit stands for a boolean function, and the complete circuit stands for the boolean function of the root. Figure 1 shows the formulae represented by each node.

Definition 1. *Let X denote a finite set of* variables *and $S \subseteq X$. A* circuit with partial evaluation and variables in S *(S-CP) has the form*

- true, false, *or x, where $x \in S$,*
- *$\neg\varphi$, $\varphi \wedge \psi$, or $\varphi \vee \psi$, where φ, ψ are S-CPs, or*
- *$\pi_{[y:=b]}\, \varphi$, where $y \in X \setminus S$, $b \in \{\mathsf{true}, \mathsf{false}\}$, and φ is an $(S \cup \{y\})$-CP.*

The set of free variables *of a S-CP φ is* $\text{free}(\varphi) := S$. *The* children *of a CP are inductively defined as follows:* true, false, *and x have no children; the children of $\varphi \wedge \psi$ and $\varphi \vee \psi$ are φ and ψ; and the only child of $\neg\varphi$ and $\pi_{[y:=b]}\,\varphi$ is φ. The set of* descendants *of φ is the smallest set M containing φ and all children of every element of M. The* size *of φ is* $|\varphi| := |M|$.

We represent a CP φ as a directed acyclic graph. The nodes of the graph are the descendants of φ. A CP φ encodes a boolean predicate P_φ, which maps assignments $\sigma\colon \text{free}(\varphi) \to \{\mathsf{false}, \mathsf{true}\}$ to a truth value $P_\varphi(\sigma) \in \{\mathsf{false}, \mathsf{true}\}$. It does so in the obvious manner, e.g., $P_x(\sigma) := \sigma(x)$, $P_{\varphi\wedge\psi}(\sigma) := P_\varphi(\sigma) \wedge P_\psi(\sigma)$, etc. We use $\pi_{[x:=b]}$ as partial evaluation operator, so $P_{\pi_{[x:=b]}\varphi}(\sigma) := P_\varphi(\sigma \cup \{x \mapsto b\})$. Intuitively, $\pi_{[x:=b]}\,\varphi$ replaces each occurrence of x in φ by b. An assignment σ *satisfies* φ if $P_\varphi(\sigma) = \mathsf{true}$. We define the macros

$$\forall_x \varphi := \pi_{[x:=0]}\,\varphi \wedge \pi_{[x:=1]}\,\varphi$$
$$\exists_x \varphi := \pi_{[x:=0]}\,\varphi \vee \pi_{[x:=1]}\,\varphi$$

Figure 1 shows a CP for the quantified boolean formula $\forall_y(\neg x \vee (x \wedge y))$.

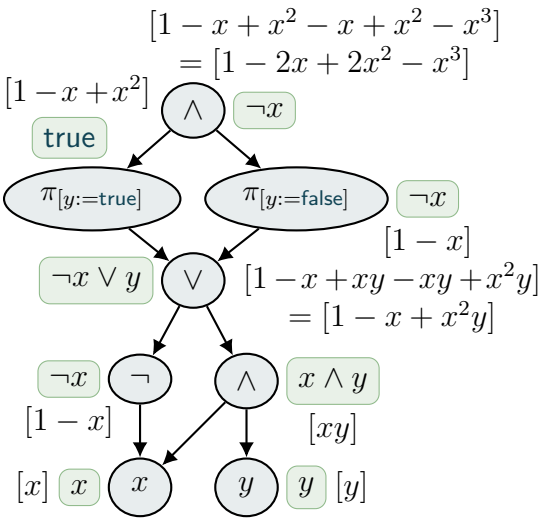

Figure 1: A CP (Section 3), the boolean functions represented by each node (in boxes), and the arithmetisation of the formulae (Section 4.1).

We consider the following problem:

#CP		
	Input	CP φ.
	Output	The number of satisfying assignments of φ.

Given a quantified boolean formula, we can use the macros for quantifiers to construct in linear time an equivalent CP, i.e., a CP with the same satisfying assignments. Similarly, #SAT instances can also be reduced to #CP.

Structure of the rest of the paper. In Section 4, we give an interactive protocol for #CP called CPCertify. In Section 5, we implement an honest Prover for CPCertify on top of an extended BDD-based algorithm for #CP. The prover runs in time polynomial in the size of the largest BDD for any of the subcircuits of the initial circuit. Together, these results yield our main result, Theorem 1, showing that any BDD-based algorithm can be modified to run an interactive protocol with small polynomial overhead. Finally, Section 6 presents empirical results.

4. An Interactive Protocol for #CP

In this section we describe an interactive protocol for #CP, following the SumCheck protocol of [19]. Section 4.1 introduces arithmetisation, a technique to transform #CP

into an equivalent problem about polynomials. Section 4.2 shows how to transform #CP into an equivalent problem about evaluating polynomials *of low degree*. Finally, Section 4.3 presents an interactive protocol for this problem.

4.1. Arithmetisation

We define a mapping $[\![\cdot]\!]$ that assigns to each CP φ a polynomial $[\![\varphi]\!]$ over the variables free(φ), called the *arithmetisation* of φ:

- $[\![\mathsf{true}]\!] := \mathbf{1}$; $[\![\mathsf{false}]\!] := \mathbf{0}$; $[\![x]\!] := [x]$ for every $x \in X$; and $[\![\neg\varphi]\!] := \mathbf{1} - [\![\varphi]\!]$;
- $[\![\varphi \wedge \psi]\!] := [\![\varphi]\!] \cdot [\![\psi]\!]$; and $[\![\varphi \vee \psi]\!] := [\![\varphi]\!] + [\![\psi]\!] - [\![\varphi]\!] \cdot [\![\psi]\!]$;
- $[\![\pi_{[x:=b]}\,\varphi]\!] := \pi_{[x:=[\![b]\!]]}[\![\varphi]\!]$, with $x \in \text{free}(\varphi)$, $b \in \{\mathsf{true}, \mathsf{false}\}$.

Figure 1 also shows the polynomials corresponding to the nodes of the CP.

Let $\mathbb{F}$ be a fixed prime finite field. Given an arbitrary truth assignment $\sigma\colon X \to \{\mathsf{true}, \mathsf{false}\}$, let $\overline{\sigma}\colon X \to \mathbb{F}$ be the binary assignment given by $\overline{\sigma}(x) = 1$ if $\sigma(x) = \mathsf{true}$ and $\overline{\sigma}(x) = 0$ if $\sigma(x) = \mathsf{false}$, where 0 and 1 denote the additive and multiplicative identities in $\mathbb{F}$. The mapping $[\![\cdot]\!]$ is defined to satisfy the following property, whose proof is immediate:

Proposition 1. *Let φ be an S-CP encoding some boolean predicate P_φ. Then $P_\varphi(\sigma) = \Pi_{\overline{\sigma}}[\![\varphi]\!]$ for every truth assignment σ to S.*

So, intuitively, the polynomial $[\![\varphi]\!]$ is a conservative extension of the predicate P_φ: It returns the same values for all binary assignments. Accordingly, in the rest of the paper we abuse language and write σ instead of $\overline{\sigma}$ for the binary assignment corresponding to the truth assignment σ.

Observe that #CP can be reformulated as follows: given a CP φ, compute the number of binary assignments σ s.t. $\Pi_\sigma[\![\varphi]\!] = \mathbf{1}$.

4.2. Degree Reduction

Given a CP φ, its associated polynomial can have degree exponential in the height of φ. Since we are ultimately interested in evaluating polynomials over binary assignments, and since $x^2 = x$ for $x \in \{0, 1\}$, we can convert polynomials to low degree without changing their behaviour on binary assignments.

For this, we use a *degree-reduction* operator δ_x for every variable x. The operator $\delta_x p$ reduces the exponent of all powers of x in p to 1. For example, $\delta_x[x^2y + 3xy^2 - 2x^3y^2 + 4] = [xy + 3xy^2 - 2xy^2 + 4]$. Observe that $\delta_x p \equiv_b p$. Instead of working on the input CP directly, we first convert it into a *circuit with partial evaluation and degree reduction* by inserting degree-reduction operators after binary operations. This ensures all intermediate polynomials obtained by arithmetisation have low degree.

Definition 2. *A* circuit with partial evaluation and degree reduction over the set S of variables (S-CPD) *is defined in the same manner as an S-CP, extended as follows:*

- *if φ is an S-CPD and $x \in S$, then $\delta_x\varphi$ is an S-CPD,*
- $[\![\delta_x\varphi]\!] := \delta_x[\![\varphi]\!]$*, and*
- *φ is the only child of $\delta_x\varphi$.*

For an S-CPD φ we define $\mathrm{free}(\varphi)$*, $|\varphi|$, children, descendants, and the graphical representation as for S-CPs.*

We convert a CP φ into a CPD $\mathrm{conv}(\varphi)$ by adding a degree-reduction operator for each free variable before any binary operation.

Definition 3. *Given a CP φ with* $\mathrm{free}(\varphi) = \{x_1, ..., x_k\}$*, its associated CPD* $\mathrm{conv}(\varphi)$ *is inductively defined as follows:*

- $\mathrm{conv}(\mathsf{false}) = \mathsf{false}$, $\mathrm{conv}(\mathsf{true}) := \mathsf{true}$,
- $\mathrm{conv}(\neg\psi) := \neg\,\mathrm{conv}(\psi)$,
- $\mathrm{conv}(\pi_{[x:=b]}\,\psi) := \pi_{[x:=b]}\,\mathrm{conv}(\psi)$*, and*
- $\mathrm{conv}(\psi_1 \circledast \psi_2) := \delta_{x_1}...\delta_{x_k}(\mathrm{conv}(\psi_1) \circledast \mathrm{conv}(\psi_2))$*, for* $\circledast \in \{\vee, \wedge\}$.

Figure 2 shows the CPD $\mathrm{conv}(\varphi)$ for the CP φ of Figure 1, together with the polynomials corresponding to each node.

We collect some basic properties of CPDs:

Lemma 1. *Let φ be a CP.*

(a) $[\![\mathrm{conv}(\varphi)]\!]$ *is a binary multilinear polynomial and* $[\![\mathrm{conv}(\varphi)]\!] \equiv_b [\![\varphi]\!]$.

(b) For every descendant ψ of $\mathrm{conv}(\varphi)$*,* $[\![\psi]\!]$ *has maximum degree* 2.

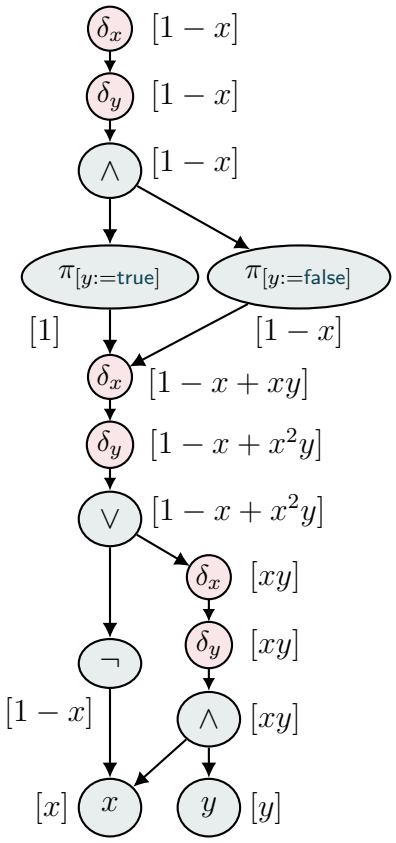

Figure 2: CPD and polynomials for the CP of Figure 1.

CPDs have another useful property. Recall that given a CP φ we are interested in its number of satisfying assignments. The next lemma shows that this number can be computed by evaluating the polynomial $[\![\mathrm{conv}(\varphi)]\!]$ on *a single input.*

Lemma 2. *A CP φ with n free variables has $m < |\mathbb{F}|$ satisfying assignments iff* $\Pi_\sigma[\![\mathrm{conv}(\varphi)]\!] = m \cdot 2^{-n}$*, where σ is the assignment satisfying $\sigma(x) := 2^{-1}$ in the field $\mathbb{F}$ for every variable x.*[1]

[1] Any prime field $\mathbb{F}$ with $|\mathbb{F}| > 2$ has an element c such that $2c = 1$.

4.3. CPCertify: An Interactive Protocol for #CP

We describe an interactive protocol, called CPCertify, for a CP φ with n free variables. Let X denote the variables used in φ. Prover and Verifier fix a finite field with at least $m+1$ elements, where m is an upper bound on the number of assignments (e.g. $m = 2^n$). Prover tries to convince the Verifier that $\Pi_\sigma[\![\mathrm{conv}(\varphi)]\!] = K$ for some $K \in \mathbb{F}$.

In the protocol, Verifier challenges Prover to compute polynomials of the form $\Pi_\sigma([\![\psi]\!])$, where ψ is a node of the CPD $\mathrm{conv}(\varphi)$ and $\sigma\colon \mathrm{free}(\psi) \to \mathbb{F}$ is a (non-binary!) assignment; we call the expression $\Pi_\sigma[\![\mathrm{conv}(\psi)]\!]$ a *challenge*. Observe that all assignments are chosen by Verifier. Prover answers with some $k \in \mathbb{F}$. We call the expression $\Pi_\sigma[\![\mathrm{conv}(\psi)]\!] = k$ a *claim*, or the *answer* to the challenge $\Pi_\sigma[\![\mathrm{conv}(\psi)]\!]$.

CPCertify consists of an initialisation and a number of rounds, one for each descendant of $\mathrm{conv}(\varphi)$. Rounds are executed in topological order, starting at the root, i.e. at $\mathrm{conv}(\varphi)$ itself. The structure of a round for a node ψ of $\mathrm{conv}(\varphi)$ depends on whether ψ is an internal node (including the root), or a leaf.

At each point, Verifier keeps track of a set $\mathcal{C}$ of claims that must be checked.

Initialisation. Verifier sends Prover the challenge $\Pi_\sigma[\![\mathrm{conv}(\varphi)]\!]$, where $\sigma(x) := 2^{-1}$ for every $x \in \mathrm{free}(\varphi)$. Prover returns the claim $\Pi_\sigma[\![\mathrm{conv}(\varphi)]\!] = K$ for some $K \in \mathbb{F}$. (By Lemma 2, this amounts to claiming that φ has $K \cdot 2^n$ satisfying assignments.) Verifier initialises $\mathcal{C} := \{\Pi_\sigma[\![\mathrm{conv}(\varphi)]\!] = K\}$.

Round for an internal node. A round for an internal node ψ runs as follows:

(a) Verifier collects all claims $\{\Pi_{\sigma_i}[\![\psi]\!] = k_i\}_{i=1}^m$ in $\mathcal{C}$ relating to ψ, with assignments $\sigma_1, ..., \sigma_m\colon \mathrm{free}(\psi) \to \mathbb{F}$ and $k_1, ..., k_m \in \mathbb{F}$. (Initially $\psi = \mathrm{conv}(\varphi)$ and the only claim is $\Pi_\sigma[\![\mathrm{conv}(\varphi)]\!] = K$.)

(b) If $m > 1$, Verifier interacts with Prover to compute a unique claim $\Pi_\sigma[\![\psi]\!] = k$ such that very likely[2] the claim is true only if all claims $\{\Pi_{\sigma_i}[\![\psi]\!] = k_i\}_{i=1}^m$ are true. For this, Verifier sends a number of challenges, and checks that the answers are *consistent* with the prior claims. Based on these answers, Verifier then derives new claims. (See "Description of step (b)" below.)

(c) Verifier interacts with Prover to compute a claim $\Pi_{\sigma'}[\![\psi']\!] = k'$ for each child ψ' of ψ. This is similar to (b): if $\Pi_\sigma[\![\psi]\!] \neq k$, i.e. the unique claim from (b) does not hold, then very likely one of the resulting claims will be wrong. Depending on the type of ψ, the claims are computed based on the answers of Prover to challenges sent by Verifier. (See "Description of step (c)" below.)

(d) In total, Verifier removed the claims $\{\Pi_{\sigma_i}[\![\psi]\!] = k_i\}_{i=1}^m$ from $\mathcal{C}$, and replaced them by one claim $\Pi_{\sigma'}[\![\psi']\!] = k'$ for each child ψ' of ψ.

Observe that, since a node ψ can be a child of several nodes, Verifier may collect multiple claims for ψ, one for each parent node.

Round for a leaf. If ψ is a leaf, then $\psi = x$ for a variable x, or $\psi \in \{\mathsf{true}, \mathsf{false}\}$. Verifier removes all claims $\{\Pi_{\sigma_i}[\![\psi]\!] = k_i\}_{i=1}^m$ from $\mathcal{C}$, computes the values $c_i := \Pi_{\sigma_i}[\![\psi]\!]$, and rejects if $k_i \neq c_i$ for any i.

[2]The precise bound on the failure probability will be given in Proposition 2.

Observe that if all claims made by Prover about leaves are true, then very likely Prover's initial claim is also true.

Description of step (b). Let $\{\Pi_{\sigma_i}[\![\psi]\!] = k_i\}_{i=1}^m$ be the claims in $\mathcal{C}$ relating to node ψ. Verifier and Prover conduct step (b) as follows:

(b.1) While there exists $x \in X$ s.t. $\sigma_1(x), ..., \sigma_m(x)$ are not pairwise equal:

  (b.1.1) For every $i \in \{1, ..., m\}$, let σ_i' denote the partial assignment which is undefined on x and otherwise matches σ_i. Verifier sends the challenges $\{\Pi_{\sigma_i'}[\![\psi]\!]\}_{i=1}^m$ to Prover. Prover answers with claims $\{\Pi_{\sigma_i'}[\![\psi]\!] = p_i\}_{i=1}^m$. Note that $p_1, ..., p_m$ are univariate polynomials with free variable x.

  (b.1.2) Verifier checks whether $k_i = \pi_{[x:=\sigma_i(x)]}\, p_i$ holds for each i. If not, Verifier rejects. Otherwise, Verifier picks $r \in \mathbb{F}$ uniformly at random and updates $\sigma_i(x) := r$ and $k_i := \pi_{[x:=r]} p_i$ for every $i \in \{1, ..., m\}$.

(b.2) If after exiting the loop the values $k_1, ..., k_m$ are not pairwise equal, Verifier rejects. Otherwise (that is, if $k_1 = k_2 = \cdots = k_m$), the set $\mathcal{C}$ now contains a unique claim $\Pi_\sigma[\![\psi]\!] = k$ relating to ψ.

Example 1. *Consider the case in which $X = \{x\}$, and Prover has made two claims, $\Pi_{\sigma_1}[\![\psi]\!] = k_1$ and $\Pi_{\sigma_2}[\![\psi]\!] = k_2$ with $\sigma_1(x) = 1$ and $\sigma_2(x) = 2$. In step (b.1.1) we have $\sigma_1' = \sigma_2'$ (both are the empty assignment), and so Verifier sends the challenge $[\![\psi]\!]$ to Prover twice, who answers with claims $[\![\psi]\!] = p_1$ and $[\![\psi]\!] = p_2$. In step (b.1.2) Verifier checks that $p_1(1) = k_1$ and $p_2(2) = k_2$ hold, picks a random number r, and updates $\sigma_1(x) := \sigma_2(x) := r$ and $k_1 := p_1(r), k_2 := p_2(r)$. Now the condition of the while loop fails, so Verifier moves to (b.2) and checks $k_1 = k_2$.*

Description of step (c). Let $\Pi_\sigma[\![\psi]\!] = k$ be the claim computed by Verifier in step (b). Verifier removes this claim from $\mathcal{C}$ and replaces it by claims about the children of ψ, depending on the structure of ψ:

(c.1) If $\psi = \psi_1 \circledast \psi_2$, for a $\circledast \in \{\vee, \wedge\}$, then Verifier sends Prover challenges $\Pi_\sigma[\![\psi_i]\!]$ for $i \in \{1, 2\}$, and Prover sends claims $\Pi_\sigma[\![\psi_i]\!] = k_i$ back. Verifier checks the consistency condition $k = \pi_{[x:=k_1]}\pi_{[y:=k_2]}[\![x \circledast y]\!]$, rejecting if it does not hold. If the condition holds, the claim $\Pi_\sigma[\![\psi_i]\!] = k_i$ is added to $\mathcal{C}$, to be checked in the round for ψ_i.

(c.2) If $\psi = \neg\psi'$, then Verifier adds the claim $\Pi_\sigma[\![\psi']\!] = 1 - k$ to ψ'.

(c.3) If $\psi = \pi_{[x:=b]}\, \psi'$, Verifier sets $\sigma' := \sigma \cup \{x \mapsto b\}$ and adds the claim $\Pi_{\sigma'}[\![\psi']\!] = k$ to $\mathcal{C}$.

(c.4) If $\psi = \delta_x \psi'$, then Verifier sends Prover the challenge $\Pi_{\sigma'}[\![\psi']\!]$, where σ' denotes the partial assignment which is undefined on x and otherwise matches σ. Prover returns the claim $p := \Pi_{\sigma'}[\![\psi']\!]$. Observe that p is a univariate polynomial over x. Verifier checks the consistency condition $\pi_{[x:=\sigma(x)]}\delta_x\, p = k$, rejecting if it does not hold. If it holds, Verifier picks an $r \in \mathbb{F}$ uniformly at random, conducts the updates $\sigma(x) := r$ and $k := \pi_{[x:=r]}\, p$, and adds $\Pi_\sigma[\![\psi']\!] = k$ to the set of claims about ψ'.

This concludes the description of the interactive protocol. We now show CPCertify is complete and sound.

Proposition 2 (CPCertify is complete and sound)**.** *Let φ be a CP with n free variables. Let $\Pi_\sigma[\![\mathrm{conv}(\varphi)]\!] = K$ be the claim initially sent by Prover to Verifier. If the claim is true, then Prover has a strategy to make Verifier accept. If not, for every Prover, Verifier accepts with probability at most $4n|\varphi|/|\mathbb{F}|$.*

If the original claim is correct, Prover can answer every challenge truthfully and all claims pass all of Verifier's checks. So Verifier accepts. If the claim is not correct, we proceed round by round. We bound the probability that the Verifier is tricked in a single step to at most $2/|\mathbb{F}|$ using the Schwartz-Zippel Lemma. We then bound the number of such steps to $2n|\varphi|$ and use a union bound.

5. A BDD-based Prover

We assume familiarity with *reduced ordered binary decision diagrams* (BDDs) [9]. We use BDDs over $X = \{x_1, ..., x_n\}$. We fix the variable order $x_1 < x_2 < ... < x_n$, i.e. the root node would decide based on the value of x_n.

Definition 4. *BDDs are defined inductively as follows:*

- $\langle\mathsf{true}\rangle$ *and* $\langle\mathsf{false}\rangle$ *are BDDs of level* 0*;*
- *if $u \neq v$ are BDDs of level ℓ_u, ℓ_v and $i > \ell_u, \ell_v$, then $\langle x_i, u, v\rangle$ is a BDD of level i;*
- *we identify $\langle x_i, u, u\rangle$ and u, for a BDD u of level ℓ_i and $i > \ell_u$.*

The level of a BDD w is denoted $\ell(w)$. The set of descendants *of w is the smallest set S with $w \in S$ and $u, v \in S$ for all $\langle x, u, v\rangle \in S$. The* size *$|w|$ of w is the number of its descendants.*

The arithmetisation *of a BDD w is the polynomial $[\![w]\!]$ defined as follows: $[\![\langle\mathsf{true}\rangle]\!] := \mathbf{1}$, $[\![\langle\mathsf{false}\rangle]\!] := \mathbf{0}$ and $[\![\langle x, u, v\rangle]\!] := [1 - x] \cdot [\![u]\!] + [x] \cdot [\![v]\!]$.*

Figure 3 shows a BDD for the boolean function $\varphi(x, y, z) = (x \wedge y \wedge \neg z) \vee (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge z)$ and the arithmetisation of each node.

BDDSolver: A BDD-based Algorithm for #CP. An instance φ of #CP can be solved using BDDs. Starting at the leaves of φ, we iteratively compute a BDD for each node ψ of the circuit encoding the boolean predicate P_ψ. At the end of this procedure we obtain a BDD for P_φ. The number of satisfying assignments of ψ is the number of accepting paths of the BDD, which can be computed in linear time in the size of the BDD.

For a node $\psi = \psi_1 \circledast \psi_2$, given BDDs representing the predicates P_{φ_1} and P_{φ_2}, we compute a BDD for the predicate $P_\varphi := P_{\varphi_1} \circledast P_{\varphi_2}$, using the $\mathrm{Apply}_\circledast$ operator on BDDs. We name this algorithm for solving #CP "BDDSolver."

From BDDSolver **to** CPCertify**.** Our goal is to modify BDDSolver to play the role of an honest Prover in CPCertify with minimal overhead. In CPCertify, Prover repeatedly performs the same task: evaluate polynomials of the form $\Pi_\sigma[\![\psi]\!]$, where ψ is a descendant of the CPD $\mathrm{conv}(\varphi)$, and σ assigns values to all free variables of ψ except possibly one. Therefore, the polynomials have at most one free variable and, as we have seen, degree at most 2.

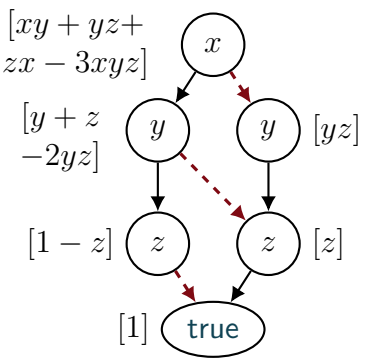

Figure 3: A BDD and its arithmetisation. For $\langle x, u, v\rangle$, we denote the link from x to v with a solid edge and x to u with a dotted edge. We omit links to $\langle\mathsf{false}\rangle$.

Before defining the concepts precisely, we give a brief overview of this section.

- First (Proposition 3), we show that BDDs correspond to binary multilinear polynomials. In particular, BDDs allow for efficient evaluation of the polynomial. As argued in Lemma 1(a), for every descendant ψ of φ, the CPD $\mathrm{conv}(\psi)$ (which is a descendant of $\mathrm{conv}(\varphi)$) evaluates to a multilinear polynomial. In particular, Prover can use standard BDD algorithms to calculate the corresponding polynomials $\Pi_\sigma[\![\psi]\!]$ for all descendants ψ of $\mathrm{conv}(\varphi)$ that are neither binary operators nor degree reductions.
- Second (the rest of the section), we prove a surprising connection: the intermediate results obtained while executing the BDD algorithms (with slight adaptations) correspond precisely to the remaining descendants of $\mathrm{conv}(\varphi)$.

The following proposition proves that BDDs represent exactly the binary multilinear polynomials.

Proposition 3. *(a) For a BDD w, $[\![w]\!]$ is a binary multilinear polynomial.*

(b) For a binary multilinear polynomial p there is a unique BDD w s.t. $p = [\![w]\!]$.

5.1. Extended BDDs

During the execution of CPCertify for a given CPD $\mathrm{conv}(\varphi)$, Prover sends to Verifier claims of the form $\Pi_\sigma[\![\psi]\!]$, where ψ is a descendant of $\mathrm{conv}(\varphi)$, and $\sigma\colon X \to \mathbb{F}$ is a partial assignment. While all polynomials computed by CPCertify are binary, not all are multilinear: some polynomials have degree 2. For these polynomials, we introduce *extended BDDs* (eBDDs) and give eBDD-based algorithms for the following two tasks:

1. Compute an eBDD representing $[\![\psi]\!]$ for every node ψ of $\mathrm{conv}(\varphi)$.
2. Given an eBDD for $[\![\psi]\!]$ and a partial assignment σ, compute $\Pi_\sigma[\![\psi]\!]$.

Computing eBDDs for CPDs: Informal introduction. Consider a CP φ and its associated CPD $\mathrm{conv}(\varphi)$. Each node of φ induces a chain of nodes in $\mathrm{conv}(\varphi)$, consisting of degree-reduction nodes $\delta_{x_1}, ..., \delta_{x_n}$, followed by the node itself (see Figure 4). Given

BDDs u and v for the children of the node in the CP, we can compute a BDD for the node itself using a well-known BDD algorithm $\mathrm{Apply}_{\circledast}(u, v)$ parametric in the boolean operation $\circledast$ labelling the node [9]. Our goal is to transform $\mathrm{Apply}_{\circledast}$ into an algorithm that computes eBDDs *for all nodes in the chain*, i.e. eBDDs for all the polynomials $p_0, p_1, ..., p_n$ of Figure 4.

Roughly speaking, $\mathrm{Apply}_{\circledast}(u, v)$ recursively computes BDDs $w_0 = \mathrm{Apply}_{\circledast}(u_0, v_0)$ and $w_1 = \mathrm{Apply}_{\circledast}(u_1, v_1)$, where u_b and v_b are the b-children of u and v, and then returns the BDD with w_0 and w_1 as 0- and 1-child, respectively.[3]

Most importantly, we modify $\mathrm{Apply}_{\circledast}$ to run in breadth-first order. Figure 5 shows a graphical representation of a run of $\mathrm{Apply}_{\vee}(u, v)$, where u and v are the two BDD nodes labelled by x. Square nodes represent pending calls to $\mathrm{Apply}_{\circledast}$. Initially there is only one square call $\mathrm{Apply}_{\vee}(u, v)$ (Figure 5, top left). $\mathrm{Apply}_{\vee}$ calls itself recursively for u_0, v_0 and u_1, v_1 (Figure 5, top right). Each of the two calls splits again into two; however, the first three are identical (Figure 5, bottom left), and so reduce to two. These two calls can now be resolved directly; they return nodes true and false, respectively. At this point, the children of $\mathrm{Apply}_{\circledast}(u, v)$ become $\langle y, \mathsf{true}, \mathsf{true}\rangle = \mathsf{true}$, and $\langle y, \mathsf{true}, \mathsf{false}\rangle$, which exists already as well (Figure 5, bottom right).

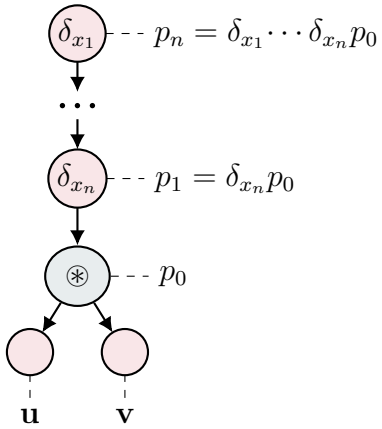

Figure 4: A node of a CP ($\circledast$) gets a chain of degree reduction nodes in the associated CPD.

We look at the diagrams of Figure 5 not as a visualisation aid, but as graphs with two kinds of nodes: standard BDD nodes, represented as circles, and *product* nodes, represented as squares. We call them *extended BDDs*. Each node of an extended BDD is assigned a polynomial in the expected way: the polynomial $[\![u]\!]$ of a standard BDD node u with variable x is $x \cdot [\![u_1]\!] + (1 - x) \cdot [\![u_0]\!]$, the polynomial $[\![v]\!]$ of a square $\wedge$-node v is $[\![v_0]\!] \cdot [\![v_1]\!]$, etc. In this way we assign to each eBDD a polynomial. In particular, we obtain the intermediate polynomials p_0, p_1, p_2, p_3 of the figure, one for each level in the recursion. In the rest of the section we show that these are *precisely* the polynomials $p_0, p_1, ..., p_n$ of Figure 4.

Thus, in order to compute eBDDs for all nodes of a CPD $\mathrm{conv}(\varphi)$, it suffices to compute BDDs for all nodes of the CP φ. Since we need to do this anyway to solve #CP, the polynomial certification does not incur any overhead.

Extended BDDs. As for BDDs, we define eBDDs over $X = \{x_1, ..., x_n\}$ with the variable order $x_1 < x_2 < ... < x_n$.

Definition 5. *Let $\circledast$ be a binary boolean operator. The set of eBDDs (for $\circledast$) is inductively defined as follows:*

[3] In fact, this is only true when u and v are nodes at the same level and $\mathrm{Apply}_{\circledast}(u_0, v_0) \neq \mathrm{Apply}_{\circledast}(u_1, v_1)$, but at this point we only want to convey some intuition.

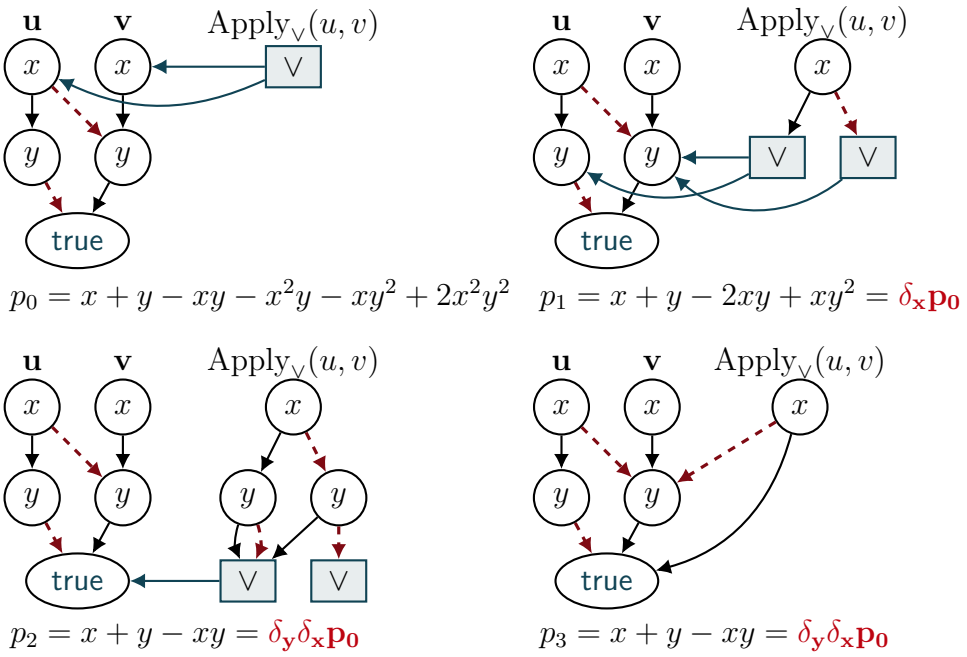

Figure 5: Run of Apply$_\vee(u, v)$, but with recursive calls evaluated in breadth-first order. All missing edges go to node false.

- *every BDD is also an eBDD of the same level;*
- *if u, v are BDDs (not eBDDs!), then $\langle u \circledast v\rangle$ is an eBDD of level l where $l :=$* $\max\{\ell(u), \ell(v)\}$*; we call eBDDs of this form* product nodes*;*
- *if $u \neq v$ are eBDDs and $i > \ell(u), \ell(v)$, then $\langle x_i, u, v\rangle$ is an eBDD of level i;*
- *we identify $\langle x_i, u, u\rangle$ and u for an eBDD u and $i > \ell(u)$.*

The set of descendants *of an eBDD w is the smallest set S with $w \in S$ and $u, v \in S$ for all $\langle u \circledast v\rangle, \langle x, u, v\rangle \in S$ The* size *of w is its number of descendants. For $u, v \in \{\langle\mathsf{true}\rangle, \langle\mathsf{false}\rangle\}$ we identify $\langle u \circledast v\rangle$ with $\langle\mathsf{true}\rangle$ or $\langle\mathsf{false}\rangle$ according to the result of $\circledast$, e.g. $\langle\langle\mathsf{true}\rangle \vee \langle\mathsf{false}\rangle\rangle = \langle\mathsf{true}\rangle$, as $\mathsf{true} \vee \mathsf{false} = \mathsf{true}$. The* arithmetisation *of an eBDD for a boolean operator $\circledast \in \{\wedge, \vee\}$ is defined as for BDDs, with the extensions $[\![\langle u \wedge v\rangle]\!] = [\![u]\!] \cdot [\![v]\!]$ and $[\![\langle u \vee v\rangle]\!] = [\![u]\!] + [\![v]\!] - [\![u]\!] \cdot [\![v]\!]$.*

Example 2. *The diagrams in Figure 5 are eBDDs for $\circledast := \vee$. Nodes of the form $\langle x, u, v\rangle$ and $\langle u \vee v\rangle$ are represented as circles and squares, respectively. Consider the top-left diagram. Abbreviating $x \oplus y := (x \wedge \neg y) \vee (\neg x \wedge y)$ we get $[\![\text{Apply}_\vee(u, v)]\!] = [\![(x \oplus y) \wedge (x \wedge y)]\!] = [\![x \oplus y]\!] \cdot [\![x \wedge y]\!] = (x(1-y) + (1-x) \cdot y - xy(1-x)(1-y)) \cdot xy$, which is the polynomial p_0 shown in the figure.*

5.1.1. Computing eBDDs for CPDs.

Given a node of a CP corresponding to a binary operator $\circledast$, Prover has to compute polynomials p_0, $\delta_{x_1}p_0$, ..., $\delta_{x_n}...\delta_{x_1}p_0$ corresponding to the nodes of the CPD shown on

ComputeEBDD(w)

Input: eBDD w
Output: sequence $w_0, ..., w_n$ of eBDDs

$w_0 := w$; output w_0
for $i = 0, \cdots, \ell(w) - 1$ **do**
  $w_{i+1} := w_i$
  for every node $\langle u \circledast v \rangle$ of w_i
    at level $n - i$ **do**
    for $b \in \{0, 1\}$ **do**
      $u_b := \pi_{[x_{n-i}:=b]}\, u$
      $v_b := \pi_{[x_{n-i}:=b]}\, v$
      $t_b := \langle u_b \circledast v_b \rangle$
    $w_{i+1} := w_{i+1}\,[\, \langle u \circledast v \rangle \,/\, \langle x_{n-i}, t_0, t_1 \rangle\,]$
  output w_{i+1}

EvaluateEBDD(w, σ) =: $E_\sigma(w)$

Input: eBDD w; assignment $\sigma\colon X \to \mathbb{F}$
Output: $\Pi_\sigma[\![w]\!]$

if $P(w)$ is defined return $P(w)$
if $w \in \{\langle \mathsf{true} \rangle, \langle \mathsf{false} \rangle\}$ return $[\![w]\!]$
if $w = \langle u \wedge v \rangle$
  $P(w) := E_\sigma(u) \cdot E_\sigma(v)$
if $w = \langle u \vee v \rangle$
  $P(w) := E_\sigma(u) + E_\sigma(v) - E_\sigma(u)E_\sigma(v)$
if $w = \langle x, u, v \rangle$ and $\sigma(x)$ undefined
  $P(w) := [1 - x] \cdot E_\sigma(u) + [x] \cdot E_\sigma(v)$
if $w = \langle x, u, v \rangle$ and $\sigma(x) = s \in \mathbb{F}$
  $P(w) := [1 - s] \cdot E_\sigma(u) + [s] \cdot E_\sigma(v)$
return $P(w)$

Table 1: On the left: Algorithm computing eBDDs for the sequence $[\![w]\!]$, $\delta_{x_n}[\![w]\!]$, $\delta_{x_{n-1}}\delta_{x_n}[\![w]\!]$, ..., $\delta_{x_1} \cdots \delta_{x_n}[\![w]\!]$ of polynomials. On the right: Recursive algorithm to evaluate the polynomial represented by an eBDD at a given partial assignment. $P(w)$ is a mapping used to memoize the polynomials returned by recursive calls.

the right. We show that Prover can compute these polynomials by representing them as eBDDs. Table 1 describes an algorithm that gets as input an eBDD w of level n, and outputs a sequence $w_0, w_1, ..., w_{n+1}$ of eBDDs such that $w_0 = w$; $[\![w_{i+1}]\!] = \delta_{x_{n-i}}[\![w_i]\!]$ for every $0 \le i \le \ell(w) - 1$; and w_{n+1} is a BDD. Interpreted as sequence of eBDDs, Figure 5 shows a run of this algorithm.

Notation. Given an eBDD w and eBDDs u, v such that $\ell(u) \ge \ell(v)$, we let $w[u/v]$ denote the result of replacing u by v in w. For an eBDD $w = \langle x_i, w_0, w_1 \rangle$ and $b \in \{0, 1\}$ we define $\pi_{[x_i:=b]}w := w_b$, and for $j > i$ we set $\pi_{[x_j:=b]}w := w$. (Note that $[\![\pi_{[x_j:=b]}w]\!] = \pi_{[x_j:=b]}[\![w]\!]$ holds for any j where it is defined.)

Proposition 4. *Let ψ_1, ψ_2 denote CPs and u_1, u_2 BDDs with $[\![u_i]\!] = [\![\psi_i]\!]$, $i \in \{1, 2\}$. Let $w := \langle u_1 \circledast u_2 \rangle$ denote an eBDD. Then* ComputeEBDD(w) *satisfies $[\![w_0]\!] = [\![\psi_1 \circledast \psi_2]\!]$ and $[\![w_{i+1}]\!] = \delta_{x_{n-i}}[\![w_i]\!]$ for every $0 \le i \le n - 1$; moreover, w_n is a BDD with $w_n =$ $\mathrm{Apply}_\circledast(u_1, u_2)$. Finally, the algorithm runs in time $\mathcal{O}(T)$, where $T \in \mathcal{O}(|u_1| \cdot |u_2|)$ is the time taken by $\mathrm{Apply}_\circledast(u_1, u_2)$.*

Evaluating polynomials represented as eBDDs. Recall that Prover must evaluate expressions of the form $\Pi_\sigma[\![\psi]\!]$ for some CPD ψ, where σ assigns values to all variables of ψ except for possibly one. We give an algorithm to evaluate arbitrary expressions $\Pi_\sigma[\![w]\!]$, where w is an eBDD, and show that if there is at most one free variable then the algorithm takes linear time in the size of ψ. The algorithm is shown on the right of Table 1. It has the standard structure of BDD procedures: It recurs on the structure of the eBDD, memoizing the result of recursive calls so that the algorithm is called at most once with a given input.

Proposition 5. *Let w denote an eBDD, $\sigma : X \to \mathbb{F}$ a partial assignment, and k the number of variables assigned by σ. Then* EvaluateEBDD *evaluates the polynomial $\Pi_\sigma[\![w]\!]$ in time $\mathcal{O}\big(\mathrm{poly}(2^{n-k}) \cdot |w|\big)$.*

5.2. Efficient Certification

In the CPCertify algorithm, Prover must (a) compute polynomials for all nodes of the CPD, and (b) evaluate them on assignments chosen by Verifier. In the last section we have seen that ComputeEBDD (for binary operations of the CP), combined with standard BDD algorithms (for all other operations), yields eBDDs representing all these polynomials—at no additional overhead, compared to a BDD-based implementation. This covers part (a). Regarding (b), recall that all polynomials computed in (a) have at most one variable. Therefore, using EvaluateEBDD we can evaluate a polynomial in linear time in the size of the eBDD representing it.

The Verifier CPCertify is implemented in a straightforward manner. As the algorithm runs in polynomial size w.r.t. the CP (and not the computed BDDs, which may be exponentially larger), incurring overhead is less of a concern.

Theorem 1 (Main Result)**.** *If* BDDSolver *solves an instance φ of* #CP *with n variables in time T, with $T > n|\varphi|$, then*

(a) Prover computes eBDDs for all nodes of $\mathrm{conv}(\varphi)$ *in time $\mathcal{O}(T)$,*

(b) Prover responds to Verifier's challenges in time $\mathcal{O}(nT)$, and

(c) Verifier executes CPCertify *in time $\mathcal{O}(n^2|\varphi|)$, with failure probability at most $4n|\varphi|/|\mathbb{F}|$.*

As presented above, EvaluateEBDD incurs a factor-of-n overhead, as every node of the CPD must be evaluated. In our implementation, we use a caching strategy to reduce the complexity of Theorem 1(b) to $\mathcal{O}(T)$.

Note that the bounds above assume a uniform cost model. In particular, operations on BDD nodes and finite field arithmetic are assumed to be $\mathcal{O}(1)$. This is a reasonable assumption, as for a constant failure probability $\log|\mathbb{F}| \approx \log n$. Hence the finite field remains small. (It is possible to verify the number of assignments even if it exceeds $|\mathbb{F}|$, see below.)

5.3. Implementation concerns

We list a number of points that are not described in detail in this paper, but need to be considered for an efficient implementation.

Finite field arithmetic. It is not necessary to use large finite fields. In particular, one can avoid the overhead of arbitrarily sized integers. For our implementation we fix the finite field $\mathbb{F} := \mathbb{Z}_p$, with $p = 2^{61} - 1$ (the largest Mersenne prime to fit in 64 bits).

Incremental eBDD representation. Algorithm ComputeEBDD computes a sequence of eBDDs. These must not be stored explicitly, otherwise one incurs a space-overhead. Instead, we only store the last eBDD as well as the differences between each subsequent element of the sequence. To evaluate the eBDDs, we then revert to a previous state by applying the differences appropriately.

Evaluation order. It simplifies the implementation if Prover only needs to evaluate nodes of the CPD in some (fixed) topological order. CPCertify can easily be adapted to guarantee this, by picking the next node appropriately in each iteration, and by evaluating only one child of a binary operator $\psi_1 \circledast \psi_2$. The value of the other child can then be derived by solving a linear equation.

Efficient evaluation. As stated in Theorem 1, using EvaluateEBDD Prover needs $\Omega(nT)$ time to respond to Verifier's challenges. In our implementation we instead use a caching strategy that reduces this time to $\mathcal{O}(T)$. Essentially, we exploit the special structure of $\mathrm{conv}(\varphi)$: Verifier sends a sequence of challenges

$$\Pi_{\sigma_0}\delta_{x_1}...\delta_{x_n}w, \quad \Pi_{\sigma_1}\delta_{x_2}...\delta_{x_n}w, \quad ..., \quad \Pi_{\sigma_n}w$$

where assignments σ_i and σ_{i+1} differ only in variables x_i and x_{i+1}. The corresponding eBDDs likewise change only at levels i and $i+1$. We cache the linear coefficients of eBDD nodes that contribute to the arithmetisation of the root top-down, and the arithmetised values of nodes bottom up. As a result, only levels $i, i+1$ need to be updated.

Large numbers of assignments. If the number of satisfying assignments of a CP exceeds $|\mathbb{F}|$, Verifier would not be able to verify the count accurately. Instead of choosing $|\mathbb{F}| \geq 2^n$, which incurs a significant overhead, Verifier can query the precise number of assignments, and then choose $|\mathbb{F}|$ randomly. This introduces another possibility of failure, but (roughly speaking) it suffices to double $\log|\mathbb{F}|$ for the additional failure probability to match the existing one. Our implementation does not currently support this technique.

6. Evaluation

We have implemented an eBDD library, blic (BDD Library with Interactive Certification) [4], that is a stand-in replacement for BDDs but additionally performs the role of Prover in the CPCertify protocol. We have also implemented a client that executes the protocol as Verifier. The eBDD library is about 900 lines of C++ code and the CPCertify protocol is about 400 lines. We have built a prototype certifying QBF solver in blic, totalling about 2600 lines of code. We aim to answer the following questions in our evaluation:

RQ1. Is a QBF solver with CPCertify-based certification competitive? If so, how high is the overhead of implementing CPCertify on top of the BDD operations?

RQ2. What is the amount of communication for Prover and Verifier in executing the CPCertify protocol, what is the time requirement for Verifier, and how do these

[4] https://gitlab.lrz.de/i7/blic

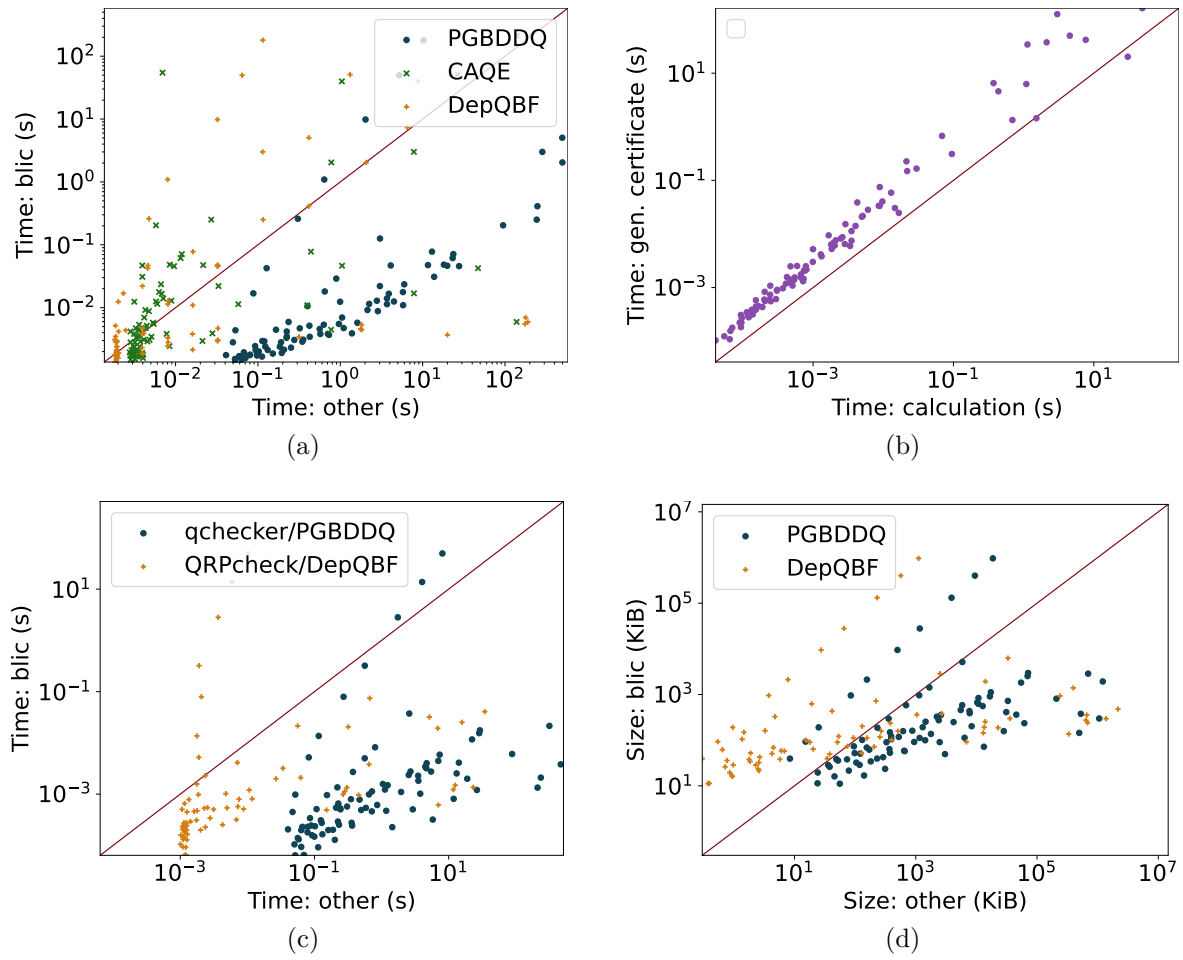

Figure 6: (a) Time taken on instances (dashed lines are $y = 100x$ and $y = 0.01x$), (b) Cost of generating a certificate over computing the solution, (c) Time to verify the certificate, (d) Size of certificates

numbers compare to proof sizes and proof checking times for certificates based on resolution and other proof systems?

RQ1: Performance of blic**.** We compare blic with CAQE, DepQBF, and PGBDDQ, three state-of-the-art QBF solvers. CAQE [28, 10] does not provide any certificates in its most recent version. DepQBF [18, 11] is a certifying QBF solver. PGBDDQ [7, 24] is an independent implementation of a BDD-based QBF solver. Both DepQBF and PGBDDQ provide specialised checkers for their certificates, though PGBDDQ can also proofs in standard QRAT format. Note that PGBDDQ is written in Python and generates proofs in an ASCII-based format, incurring overhead compared to the other tools.

We take 172 QBF instances (all unsatisfiable) from the *Crafted Instances* track of the QBF Evaluation 2022.[5] The *Prenex CNF* track of the QBF competition is not evaluated

[5] CAQE and DepQBF were the winner and runner-up in this category. The configuration we used differs from the competition, as described in Appendix H.

Instance		Solve time (s)		Certificate (MiB)		Verifier time (s)	
n	result	blic	PGBDDQ	blic	PGBDDQ	blic	qchecker
10	sat	0.03	3.67	1.20	8.48	0.01	3.80
10	unsat	0.03	3.66	1.20	8.45	0.01	3.83
15	sat	0.13	18.07	4.12	44.25	0.02	18.45
15	unsat	0.13	18.14	4.11	44.20	0.02	18.55
20	sat	0.54	82.92	11.59	198.54	0.07	80.28
20	unsat	0.53	83.02	11.64	198.76	0.06	79.05
25	sat	1.56	261.16	23.94	566.95	0.14	238.99
25	unsat	1.55	261.25	23.86	565.36	0.15	237.94
40	sat	25.22	4863.71	132.43	7464.96	0.95	5141.08
40	unsat	25.25	4827.06	132.67	7467.84	0.99	5463.54

Table 2: Comparison of certificate generation, bytes exchanged between prover and verifier, and time taken to verify the certificate on a set of QBF benchmarks from [7]. "Solve time" is time taken to solve the instance and to generate a certificate (seconds), "Certificate" is the size of proof encoding for PGBDDQ, and bytes exchanged by CPCertify for blic, and "Verifier time" is time to verify the certificate (Verifier's run time for blic and time taken by qchecker).

here. It features instances with a large number of variables. BDD-based solvers perform poorly under these circumstances without additional optimisations. Our overall goal is not to propose a new approach for solving QBF, but rather to certify a BDD-based approach, so we wanted to focus on cases where the existing BDD-based approaches are practical.

We ran each benchmark with a 10 minute timeout; all tools other than CAQE were run with certificate production. All times were obtained on a machine with an Intel Xeon E7-8857 CPU and 1.58 TiB RAM[6] running Linux. See Appendix H for a detailed description. blic solved 96 out of 172 benchmarks, CAQE solved 98, DepQBF solved 87, and PGBDDQ solved 91. Figure 6(a) shows the run times of blic compared to the other tools. The plot indicates that blic is competitive on these instances, with a few cases, mostly from the Lonsing family of benchmarks, where blic is slower than DepQBF by an order of magnitude. Figure 6(b) shows the overhead of certification: for each benchmark (that finishes within a 10min timeout), we plot the ratio of the time to compute the answer to the time it takes to run Prover in CPCertify. The dotted regression line shows CPCertify has a 2.8× overhead over computing BDDs. For this set of examples, the error probability never exceeds $10^{-8.9}$ ($10^{-11.6}$ when Lonsing examples are excluded); running the verifier k times reduces it to $10^{-8.9k}$.

RQ2: Communication Cost of Certification and Verifier Time. We explore **RQ2** by comparing the number of bytes exchanged between Prover and Verifier and the time needed for Verifier to execute CPCertify with the number of bytes in an QBF proof

[6] blic uses at most 60 GiB on the shown benchmarks, 5 GiB when excluding timeouts.

and the time required to verify the proof produced by DepQBF and PGBDDQ, for which we use QRPcheck [23, 25] and qchecker [7, 24], respectively. Note that the latter is written in Python.

We show that the overhead of certification is low. Figure 6(c) shows the run time of Verifier—this is generally negligible for blic, except for the Lonsing and KBKF families, which have a large number of variables, but very small BDDs. Figure 6(d) shows the total number of bytes exchanged between Prover and Verifier in blic against the size of the proofs generated by PGBDDQ and DepQBF. For large instances, the number of bytes exchanged in blic is significantly smaller than the size of the proofs. The exception are again the Lonsing and KBKNF families of instances. For both plots, the dotted line results from a log-linear regression.

In addition to the Crafted Instances, we compare against PGBDDQ on a challenging family of benchmarks used in the PGBDDQ paper (matching the parameters of [7, Table 3]); these are QBF encodings of a linear domino placing game.[7] Our results are summarised in Table 2. The upper bound on Verifier error is $10^{-9.22}$. We show that blic outperforms PGBDDQ both in overall cost of computing the answer and the certificates as well as in the number of bytes communicated and the time used by Verifier.

Our results indicate that giving up absolute certainty through interactive protocols can lead to an order of magnitude smaller communication cost and several orders of magnitude smaller checking costs for the verifier.

7. Conclusion

We have presented a solver that combines BDDs with an interactive protocol. blic can be seen as a self-certifying BDD library able to certify the correctness of arbitrary sequences of BDD operations. In order to trust the result, a user must only trust the verifier (a straightforward program that poses challenges to the prover). We have shown that blic (including certification time) is competitive with other solvers, and Verifier's time and error probabilities are negligible.

Our results show that IP = PSPACE can become an important result not only in theory but also in the practice of automatic verification. From this perspective, our paper is a first step towards practical certification based on interactive protocols. While we have focused on BDDs, we can ask the more general question: which practical automated reasoning algorithms can be made efficiently certifying? For example, whether there is an interactive protocol and an efficient certifying version of modern SAT solving algorithms is an interesting open challenge.

References

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2006.

[7] DepQBF only solved 1 of 10 instances within 120min, and is thus not compared.

[2] László Babai. Trading group theory for randomness. In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 421–429. ACM, 1985.

[3] Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2014.

[4] Haniel Barbosa, Andrew Reynolds, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Arjun Viswanathan, Scott Viteri, Yoni Zohar, Cesare Tinelli, and Clark W. Barrett. Flexible proof production in an industrial-strength SMT solver. In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings*, volume 13385 of *Lecture Notes in Computer Science*, pages 15–35. Springer, 2022.

[5] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 37–56. Springer, 1988.

[6] Randal E. Bryant, Armin Biere, and Marijn J. H. Heule. Clausal proofs for pseudo-boolean reasoning. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2022.

[7] Randal E. Bryant and Marijn J. H. Heule. Dual proof generation for quantified boolean formulas with a bdd-based solver. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 433–449. Springer, 2021.

[8] Randal E. Bryant and Marijn J. H. Heule. Generating extended resolution proofs with a bdd-based SAT solver. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg,*

March 27 - April 1, 2021, Proceedings, Part I, volume 12651 of *Lecture Notes in Computer Science*, pages 76–93. Springer, 2021.

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

[10] CAQE. https://github.com/ltentrup/caqe, 2023. [Online; accessed 03-February-2023].

[11] depqbf. https://github.com/lonsing/depqbf, 2017. [Online; accessed 03-February-2023].

[12] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985.

[13] T.A. Henzinger, R. Jhala, R. Majumdar, G.C. Necula, G. Sutre, and W. Weimer. Temporal-safety proofs for systems code. In *CAV 02: Computer-Aided Verification*, Lecture Notes in Computer Science 2404, pages 526–538. Springer-Verlag, 2002.

[14] Marijn Heule. Everything's bigger in texas: "the largest math proof ever". In Christoph Benzmüller, Christine L. Lisetti, and Martin Theobald, editors, *GCAI 2017, 3rd Global Conference on Artificial Intelligence, Miami, FL, USA, 18-22 October 2017*, volume 50 of *EPiC Series in Computing*, pages 1–5. EasyChair, 2017.

[15] Marijn J. H. Heule. Proofs of unsatisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 635–668. IOS Press, 2021.

[16] Toni Jussila, Carsten Sinz, and Armin Biere. Extended resolution proofs for symbolic SAT solving with quantification. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 54–60. Springer, 2006.

[17] Guy Katz, Clark W. Barrett, Cesare Tinelli, Andrew Reynolds, and Liana Hadarean. Lazy proofs for dpll(t)-based SMT solvers. In Ruzica Piskac and Muralidhar Talupur, editors, *2016 Formal Methods in Computer-Aided Design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016*, pages 93–100. IEEE, 2016.

[18] Florian Lonsing and Uwe Egly. Depqbf 6.0: A search-based QBF solver beyond traditional QCDCL. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 371–384. Springer, 2017.

[19] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[20] Ning Luo, Timos Antonopoulos, William R. Harris, Ruzica Piskac, Eran Tromer, and Xiao Wang. Proving UNSAT in zero knowledge. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2203–2217. ACM, 2022.

[21] K. Namjoshi. Certifying model checkers. In *CAV 01: Computer Aided Verification*, Lecture Notes in Computer Science 2102, pages 2–13. Springer-Verlag, 2001.

[22] G.C. Necula. Proof-carrying code. In *Principles of Programming Languages*, pages 106–119. ACM Press, 1997.

[23] Aina Niemetz, Mathias Preiner, Florian Lonsing, Martina Seidl, and Armin Biere. Resolution-based certificate extraction for QBF - (tool presentation). In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 430–435. Springer, 2012.

[24] PGBDDQ. https://github.com/rebryant/pgbdd, 2023. [Online; accessed 03-February-2023].

[25] QRPcheck. http://fmv.jku.at/qrpcheck/, 2023. [Online; accessed 03-February-2023].

[26] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[27] Carsten Sinz and Armin Biere. Extended resolution proofs for conjoining bdds. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *Computer Science - Theory and Applications, First International Symposium on Computer Science in Russia, CSR 2006, St. Petersburg, Russia, June 8-12, 2006, Proceedings*, volume 3967 of *Lecture Notes in Computer Science*, pages 600–611. Springer, 2006.

[28] Leander Tentrup and Markus N. Rabe. Clausal abstraction for DQBF. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 388–405. Springer, 2019.

A. Proof of Lemma 1

Before we start with the proof, we show a technical property, namely that binary equivalence is preserved by addition and multiplication.

Lemma 3. *Let p_1, p_2, q_1, q_2 denote polynomials with $p_i \equiv_b q_i$ for $i \in \{1,2\}$. Then $p_1 + p_2 \equiv_b q_1 + q_2$, $p_1 \cdot p_2 \equiv_b q_1 \cdot q_2$, and $\pi_{[x:=r]}p_1 \equiv_b \pi_{[x:=r]}q_1$ for $r \in \{0,1\}$.*

Proof. Let σ denote an arbitrary binary assignment. Then

$$\begin{array}{llll} \Pi_\sigma(p_1 + p_2) & = \Pi_\sigma p_1 + \Pi_\sigma p_2 & = \Pi_\sigma q_1 + \Pi_\sigma q_2 & = \Pi_\sigma(q_1 + q_2) \\ \Pi_\sigma(p_1 \cdot p_2) & = \Pi_\sigma p_1 \cdot \Pi_\sigma p_2 & = \Pi_\sigma q_1 \cdot \Pi_\sigma q_2 & = \Pi_\sigma(q_1 \cdot q_2) \\ \Pi_\sigma \pi_{[x:=r]} p_1 & = \pi_{[x:=r]} \Pi_\sigma p_1 & = \pi_{[x:=r]} \Pi_\sigma q_1 & = \Pi_\sigma \pi_{[x:=r]} q_1 \end{array}$$

□

Lemma 1. *Let φ be a CP.*

(a) $[\![\mathrm{conv}(\varphi)]\!]$ is a binary multilinear polynomial and $[\![\mathrm{conv}(\varphi)]\!] \equiv_b [\![\varphi]\!]$.

(b) For every descendant ψ of $\mathrm{conv}(\varphi)$, $[\![\psi]\!]$ has maximum degree 2.

Proof. **Part (a).** We proceed by structural induction on φ, to show $[\![\mathrm{conv}(\varphi)]\!] \equiv_b [\![\varphi]\!]$.

- The base case $\varphi \in \{\mathsf{true}, \mathsf{false}\}$ is trivial.
- Let $\varphi = \neg\psi$. By hypothesis, $[\![\mathrm{conv}(\psi)]\!] \equiv_b [\![\psi]\!]$. Now we have

  $$[\![\mathrm{conv}(\neg\psi)]\!] = [\![\neg\,\mathrm{conv}(\psi)]\!] = 1 - [\![\mathrm{conv}(\psi)]\!] \overset{(*)}{\equiv}_b 1 - [\![\psi]\!] = [\![\varphi]\!]$$

  At $(*)$ we use both the induction hypothesis and Lemma 3. We use $(*)$ for the other cases as well, with the same meaning.
- Let $\varphi = \pi_{[x:=a]}\psi$.

  $$[\![\mathrm{conv}(\pi_{[x:=a]}\psi)]\!] = \pi_{[x:=[\![a]\!]}][\![\mathrm{conv}(\psi)]\!] \overset{(*)}{\equiv}_b \pi_{[x:=a]}[\![\psi]\!] = [\![\varphi]\!]$$

- Let $\varphi = \psi_1 \wedge \psi_2$. We use $\delta_x p \equiv_b p$ for any polynomial p and variable x.

  $$\begin{aligned} &[\![\mathrm{conv}(\psi_1 \wedge \psi_2)]\!] = [\![\delta_{x_1}...\delta_{x_k}(\mathrm{conv}(\psi_1) \wedge \mathrm{conv}(\psi_2))]\!] \\ &\quad \equiv_b [\![\mathrm{conv}(\psi_1) \wedge \mathrm{conv}(\psi_2)]\!] = [\![\mathrm{conv}(\psi_1)]\!] \cdot [\![\mathrm{conv}(\psi_2)]\!] \overset{(*)}{\equiv}_b [\![\psi_1]\!] \cdot [\![\psi_2]\!] \\ &\quad = [\![\psi_1 \wedge \psi_2]\!] = [\![\varphi]\!] \end{aligned}$$

- For $\varphi = \psi_1 \vee \psi_2$ the argument is analogous.

It remains to show that $[\![\mathrm{conv}(\varphi)]\!]$ is multilinear. Again, we do a structural induction. The base case $\varphi \in \{\mathsf{true}, \mathsf{false}\}$ is again trivial. The other cases all follow from the observation that, given multilinear polynomials p, q over variables $x_1, ..., x_k$, variable x and $a \in \{\mathsf{true}, \mathsf{false}\}$, all of $1 - p$, $\pi_{[x:=a]}p$, $\delta_{x_1}...\delta_{x_k}(p \cdot q)$, and $\delta_{x_1}...\delta_{x_k}(p + q - p \cdot q)$ are multilinear.

Part (b). If there is a descendant ψ' of φ, s.t. $\mathrm{conv}(\psi') = \psi$, then the statement follows from part (a), as $[\![\psi]\!]$ is multilinear. This leaves two cases. First, if $\psi = \psi_1 \circledast \psi_2$, then $\psi_i = \mathrm{conv}(\psi_i')$ for $i \in \{1,2\}$ by construction. Therefore, $[\![\psi_i]\!]$ is multilinear. So if $\circledast = \wedge$ we get $[\![\psi]\!] = [\![\psi_1]\!] \cdot [\![\psi_2]\!]$, which has maximum degree 2. Analogously for $\circledast = \vee$.

Second, we have the case $\psi = \delta_x\psi_1$. By induction, we find that ψ_1 has maximum degree 2, which cannot be increased by δ_x. □

B. Proof of Lemma 2

Lemma 2. *A CP φ with n free variables has $m < |\mathbb{F}|$ satisfying assignments iff $\Pi_\sigma[\![\mathrm{conv}(\varphi)]\!] = m \cdot 2^{-n}$, where σ is the assignment satisfying $\sigma(x) := 2^{-1}$ in the field $\mathbb{F}$ for every variable x.*[8]

Proof. Let $X = \{x_1, ..., x_n\} := \mathrm{free}(\varphi)$, and let $S := \{\sigma \mid \sigma : X \to \{0,1\}\}$ be the set of binary assignments on X. We also set $p := [\![\mathrm{conv}(\varphi)]\!]$. Then

$$m \overset{(1)}{=} \sum_{\sigma \in S} \Pi_\sigma [\![\varphi]\!] \overset{(2)}{=} \sum_{\sigma \in S} \Pi_\sigma p$$

where (1) uses Proposition 1 and $m < |\mathbb{F}|$, and (2) uses $[\![\varphi]\!] \equiv_b p = [\![\mathrm{conv}(\varphi)]\!]$ (Lemma 1).

We now introduce the notation $\Sigma_x p := \pi_{[x:=0]} p + \pi_{[x:=1]} p$ for $x \in X$. Using this notation, we rewrite above equation.

$$m = \Sigma_{x_1} \Sigma_{x_2} \cdots \Sigma_{x_n} p \tag{$*$}$$

Crucially, we can now use the fact that p is multilinear (again Lemma 1), to derive

$$p = \delta_x p = [1 - x] \cdot \pi_{[x:=0]} p + [x] \cdot \pi_{[x:=1]} p$$

for any $x \in X$. Setting x to $1/2$ yields

$$\begin{aligned} \pi_{[x:=1/2]} p &= \pi_{[x:=1/2]} \Big([1 - x] \cdot \pi_{[x:=0]} p + [x] \cdot \pi_{[x:=1]} p \Big) \\ &= (\pi_{[x:=0]} p + \pi_{[x:=1]} p)/2 = \Sigma_x p / 2 \end{aligned}$$

In other words, $\Sigma_x p = 2 \cdot \pi_{[x:=1/2]} p$. By plugging this into $(*)$ we get $m = 2^n \Pi_\sigma p$, where $\sigma(x) := 1/2$ for $x \in X$, as desired. □

C. Proof of Proposition 2

The core of the argument in Proposition 2 uses the Schwartz-Zippel Lemma, of which we only need a very simple version.

Lemma 4 (Schwartz-Zippel Lemma)**.** *Let p_1, p_2 be distinct univariate polynomials over $\mathbb{F}$ of degree at most $d \geq 0$. Let r be selected uniformly at random from $\mathbb{F}$. The probability that $p_1(r) = p_2(r)$ holds is at most $d/|\mathbb{F}|$.*

Proof. Since $p_1 \neq p_2$ the polynomial $p := p_1 - p_2$ is not the zero polynomial and has degree at most d. Therefore p has at most d zeros, and so the probability of $p(r) = 0$ is at most $d/|\mathbb{F}|$. □

Now we move on to the proof.

[8] Any prime field $\mathbb{F}$ with $|\mathbb{F}| > 2$ has an element c such that $2c = 1$.

Proposition 2 (CPCertify is complete and sound)**.** *Let φ be a CP with n free variables. Let $\Pi_\sigma[\![\mathrm{conv}(\varphi)]\!] = K$ be the claim initially sent by Prover to Verifier. If the claim is true, then Prover has a strategy to make Verifier accept. If not, for every Prover, Verifier accepts with probability at most $4n|\varphi|/|\mathbb{F}|$.*

Proof. If the claim is true, Prover can always answer every challenge posed by Verifier truthfully. True claims pass all the checks conducted by Verifier, and so Verifier accepts.

Assume now the claim is false. We show that Verifier accepts with probability at most $4n|\varphi|/|\mathbb{F}|$.

First, we consider the contents of $\mathcal{C}$, the set of claims yet to be checked, after each step of the protocol. This gives rise to a sequence $\mathcal{C}_0, \mathcal{C}_1, ...\mathcal{C}_l$, where $\mathcal{C}_0$ contains only the initial claim. In particular, we consider each iteration of step (b.1) separately, so executing the loop s times adds s elements to the sequence.

As (by assumption) the initial claim is false, $\mathcal{C}_0$ contains a false claim. For the moment, assume that Verifier accepts. They therefore must complete all rounds without rejecting. As the nodes are processed in topological order (every node is processed before its descendants), eventually $\mathcal{C}$ must become empty: an inner node ψ replaces all claims about itself with claims about its children, and each leaf node ψ removes all claims about itself.

So $\mathcal{C}_l = \emptyset$ contains only true claims and thus there must be an i, s.t. $\mathcal{C}_i$ contains one false claim, but $\mathcal{C}_{i+1}$ contains only true claims. For any such i, we say that Prover *tricks* Verifier in step i. In other words: if Verifier accepts, it was tricked at some point.

We will now show that at each step, Verifier is tricked either with probability 0 or probability at most $2/|\mathbb{F}|$, and the latter case occurs at most $2n|\varphi|$ times. By union bound, this implies the stated bound.

First, we show that Verifier can only be tricked in an iteration of step (b.1), or in step (c.4). Steps (a) and (b.2) do not modify $\mathcal{C}$. The arguments for steps (c.1-c.3) are analogous, so we only present (c.1) with $\circledast = \wedge$ here. Let $\Pi_\sigma[\![\psi]\!] = k$ be the claim to be checked. We have

$$\Pi_\sigma[\![\psi]\!] = \Pi_\sigma[\![\psi_1 \wedge \psi_2]\!] = \Pi_\sigma([\![\psi_1]\!] \cdot [\![\psi_2]\!]) = \Pi_\sigma[\![\psi_1]\!] \cdot \Pi_\sigma[\![\psi_2]\!]$$

So $\Pi_\sigma[\![\psi]\!] = k$ is equivalent to $\Pi_\sigma[\![\psi_1]\!] = k_1 \wedge \Pi_\sigma[\![\psi_2]\!] = k_2 \wedge k_1 k_2 = k$. In other words, if $\Pi_\sigma[\![\psi]\!] \neq k$, then either $k_1 k_2 \neq k$ and Verifier rejects immediately (note $\pi_{[x:=k_1]}\pi_{[y:=k_2]}[\![x \wedge y]\!] = k_1 k_2$), or one of the two claims added to $\mathcal{C}$ is false.

For step (c.4) we get

$$\Pi_\sigma[\![\psi]\!] = \pi_{[x:=\sigma(x)]}\Pi_{\sigma'}[\![\delta_x \psi']\!] = \pi_{[x:=\sigma(x)]}\Pi_{\sigma'}\delta_x[\![\psi']\!] = \pi_{[x:=\sigma(x)]}\delta_x\Pi_{\sigma'}[\![\psi']\!]$$

so $\Pi_\sigma[\![\psi]\!] = k$ is equivalent to $\Pi_{\sigma'}[\![\psi']\!] = p \wedge \pi_{[x:=\sigma(x)]}\delta_x p = k$. Conversely, if the claim does not hold we get either $\pi_{[x:=\sigma(x)]}\delta_x p \neq k$ and Verifier rejects immediately, or $\Pi_{\sigma'}[\![\psi']\!] \neq p$. Note that p is a univariate polynomial with degree at most 2 (Lemma 1b), not a constant — Verifier cannot simply add the claim $\Pi_{\sigma'}[\![\psi']\!] = p$ to $\mathcal{C}$. However, by Lemma 4, $\Pi_{\sigma'}[\![\psi']\!] \neq p$ implies that $\pi_{[x:=r]}\Pi_{\sigma'}[\![\psi']\!] = \pi_{[x:=r]}p$ holds with probability at most $2/|\mathbb{F}|$. Otherwise, the claim added to $\mathcal{C}$ is false and Verifier is not tricked.

Step (b.1) remains. We remark that, though unintuitive, the probability that Verifier is tricked does not increase with m, the number of claims to be merged. If all claims $\{\Pi_{\sigma_i}[\![\psi]\!] = k_i\}_{i=1}^m$ are true, then clearly Verifier cannot be tricked in this step. So fix an i s.t. $\Pi_{\sigma_i}[\![\psi]\!] \neq k_i$. Similar to step (c.4) above we get that either $\pi_{[x:=\sigma_i(x)]}\, p_i \neq k_i$ and Verifier rejects, or $\Pi_{\sigma_i'}[\![\psi]\!] \neq p_i$. In the latter case, $\pi_{[x:=r]}\Pi_{\sigma_i'}[\![\psi]\!] = \pi_{[x:=r]}p_i$ again holds with probability at most $2/|\mathbb{F}|$.

To conclude the proof, we argue that steps (b.1) and (c.4) occur at most $2n|\varphi|$ times in total. Step (c.4) occurs at most $|\mathrm{conv}(\varphi)| \leq n|\varphi|$ times. For (b.1) we note that it is executed at most n times for each node with more than one parent. The conversion of φ to a CPD does not increase the number of such nodes, so it also occurs at most $n|\varphi|$ times. □

D. Proof of Proposition 3

Proposition 3. *(a) For a BDD w, $[\![w]\!]$ is a binary multilinear polynomial.*

(b) For a binary multilinear polynomial p there is a unique BDD w s.t. $p = [\![w]\!]$.

Proof. **Part (a).** We proceed by induction, and show that a BDD w with $i := \ell(w)$ is multilinear in $x_1, ..., x_i$ and does not depend on $x_{i+1}, ..., x_n$. The base case $i = 0$ is trivial. For the induction step, let $w = \langle x_i, u, v\rangle$. We have $[\![w]\!] = [1 - x_i] \cdot [\![u]\!] + [x_i] \cdot [\![v]\!]$. By induction hypothesis, $[\![u]\!], [\![v]\!]$ are multilinear in $x_1, ..., x_{i-1}$ and do not depend on $x_i, ..., x_n$. The claim follow immediately.

It remains to argue that $[\![w]\!]$ is binary, for a BDD w. We again proceed by induction on $\ell(w)$. For $w \in \{\langle \mathsf{false}\rangle, \langle \mathsf{true}\rangle\}$, this is clear, so let $w = \langle x, u, v\rangle$ and let σ denote a binary assignment. We get

$$\begin{aligned}\Pi_\sigma[\![w]\!] &= \Pi_\sigma\Big([1 - x_i] \cdot [\![u]\!] + [x_i] \cdot [\![v]\!]\Big)\\ &= [1 - \sigma(x_i)] \cdot \Pi_\sigma[\![u]\!] + [\sigma(x_i)] \cdot \Pi_\sigma[\![v]\!] \in \{\Pi_\sigma[\![u]\!], \Pi_\sigma[\![v]\!]\}\end{aligned}$$

The last step uses $\sigma(x_i) \in \{0, 1\}$. By induction hypothesis, both $\Pi_\sigma[\![u]\!]$ and $\Pi_\sigma[\![v]\!]$ are $\mathbf{0}$ or $\mathbf{1}$, and the claim follows.

Part (b). Before we prove this part, we remark that the statement follows from two well-known facts: multilinear polynomials are uniquely determined by the values they take on binary assignments, and BDDs uniquely represent arbitrary boolean functions. Here, however, we give an elementary proof.

We show that such a w exists if p is a polynomial over variables $x_1, ..., x_i$, for all $0 \leq i \leq n$. We proceed by induction on i. For $i = 0$ we have $p \in \{\mathbf{0}, \mathbf{1}\}$ and choose the appropriate $w \in \{\langle \mathsf{false}\rangle, \langle \mathsf{true}\rangle\}$. For $i > 0$, let $p_b := \pi_{[x_i:=b]}p$, for $b \in \{0, 1\}$, and let w_b denote a BDD for p_b.

We have $p = [1 - x_i] \cdot p_0 + [x_i] \cdot p_1$. If $p_0 = p_1$, then $p = p_0$ does not depend on x_i, so $p = [\![w_0]\!]$. Otherwise, with $w := \langle x_i, w_0, w_1\rangle$ we have $[\![w]\!] = [1 - x_i] \cdot [\![w_0]\!] + [x_i] \cdot [\![w_1]\!] = p$.

It remains to show that w is unique. We will do this by proving that $[\![u]\!] \neq [\![v]\!]$ for all BDDs $u \neq v$. Assume the contrary, and choose a counterexample $u \neq v$ s.t. $[\![u]\!] = [\![v]\!]$ and $\ell(u) + \ell(v)$ is minimal. Wlog. we assume $\ell(u) \leq \ell(v) =: i$.

First, we consider the case that $\ell(u) < i$. Then $v = \langle x_i, v_0, v_1 \rangle$ and we get $[\![u]\!] = \pi_{[x_i:=b]}[\![u]\!] = \pi_{[x_i:=b]}[\![v]\!] = [\![v_b]\!]$ for $b \in \{0,1\}$. Due to $v_0 \neq v_1$ (else v would not be a BDD) we find a b with $u \neq v_b$, but $[\![u]\!] = [\![v_b]\!]$. This contradicts minimality of the counterexample.

Second, we have the case $\ell(u) = \ell(v) = i$. Clearly, $i > 0$, so $v = \langle x_i, v_0, v_1 \rangle$ and $u = \langle x_i, u_0, u_1 \rangle$. Due to $v \neq u$ there is a $b \in \{0,1\}$ with $v_b \neq u_b$. But we get $[\![v_b]\!] = \pi_{[x_i:=b]}[\![v]\!] = \pi_{[x_i:=b]}[\![u]\!] = [\![u_b]\!]$, so again a smaller counterexample exists. □

E. Proof of Proposition 4

Proposition 4. *Let ψ_1, ψ_2 denote CPs and u_1, u_2 BDDs with $[\![u_i]\!] = [\![\psi_i]\!]$, $i \in \{1,2\}$. Let $w := \langle u_1 \circledast u_2 \rangle$ denote an eBDD. Then* ComputeEBDD(w) *satisfies $[\![w_0]\!] = [\![\psi_1 \circledast \psi_2]\!]$ and $[\![w_{i+1}]\!] = \delta_{x_{n-i}}[\![w_i]\!]$ for every $0 \leq i \leq n-1$; moreover, w_n is a BDD with $w_n =$ $\mathrm{Apply}_{\circledast}(u_1, u_2)$. Finally, the algorithm runs in time $\mathcal{O}(T)$, where $T \in \mathcal{O}(|u_1| \cdot |u_2|)$ is the time taken by $\mathrm{Apply}_{\circledast}(u_1, u_2)$.*

The proof will take up the remainder of this section.

For the time bound, observe that EvaluateEBDD performs the same operations as $\mathrm{Apply}_{\circledast}$, but with a breadth-first traversal of the BDD, instead of a depth-first one. Clearly, this does not increase the time complexity.

The bound $T \in \mathcal{O}(|u_1| \cdot |u_2|)$ is a well-known bound for $\mathrm{Apply}_{\circledast}$, it relies on there being at most $|u_1| \cdot |u_2|$ recursive calls, as each call corresponds to a pair of BDD nodes (and identical calls are memoised). Naturally, the same bound holds for EvaluateEBDD, where the number of created product nodes is bounded by $|u_1| \cdot |u_2|$. Each of these product nodes is operated on once, by replacing it with a BDD nodes.

Now we move to showing correctness, which will follow from Lemmata 9 and 10. We start with a basic property of the degree reduction operator.

Lemma 5. *Let p denote a polynomial and x a variable. Then we have $\delta_x p = [1-x] \cdot \pi_{[x:=0]}p + [x] \cdot \pi_{[x:=1]}p$.*

Proof. We write p as $p = p_0 + [x] \cdot p_1 + ... + [x^k] \cdot p_k$ for polynomials $p_0, ..., p_k$ which do not depend on x. We get

$$\begin{aligned}&[1-x] \cdot \pi_{[x:=0]}p + [x] \cdot \pi_{[x:=1]}p = [1-x] \cdot p_0 + [x] \cdot (p_0 + p_1 + ... + p_k)\\ &= p_0 + [x] \cdot (p_1 + ... + p_k) = \delta_x p\end{aligned}$$

□

We first show that the innermost loop of ComputeEBDD computes a degree reduction.

Lemma 6. *Let $\langle u \circledast v \rangle$ be a product eBDD, and let $s := \langle x_{n-i}, t_0, t_1 \rangle$ be the eBDD computed by the innermost loop of* ComputeEBDD *(Table 1, left). Then $[\![s]\!] = \delta_{x_{n-i}}[\![\langle u \circledast v \rangle]\!]$.*

Proof. For $\circledast = \wedge$ and $b \in \{0,1\}$ we get

$$\begin{aligned}[\![t_b]\!] &= [\![\langle u_b \wedge v_b \rangle]\!] = [\![u_b]\!] \cdot [\![v_b]\!] = [\![\pi_{[x_{n-i}:=b]} u]\!] \cdot [\![\pi_{[x_{n-i}:=b]} v]\!] \\ &= \pi_{[x_{n-i}:=b]}[\![u]\!] \cdot \pi_{[x_{n-i}:=b]}[\![v]\!] = \pi_{[x_{n-i}:=b]}\Big([\![u]\!] \cdot [\![v]\!]\Big) = \pi_{[x_{n-i}:=b]}[\![\langle u \wedge v \rangle]\!]\end{aligned}$$

Analogously, one can derive $[\![t_b]\!] = \pi_{[x_{n-i}:=b]}[\![\langle u \circledast v \rangle]\!]$ for $\circledast = \vee$ as well. Finally:

$$\begin{aligned}[\![s]\!] &= [1 - x_{n-i}] \cdot [\![t_0]\!] + [x_{n-i}] \cdot [\![t_1]\!] \\ &= [1 - x_{n-i}] \cdot \pi_{[x_{n-i}:=0]}[\![\langle u \circledast v \rangle]\!] + [x_{n-i}] \cdot \pi_{[x_{n-i}:=1]}[\![\langle u \circledast v \rangle]\!] \\ &= \delta_{x_{n-i}}[\![\langle u \circledast v \rangle]\!]\end{aligned}$$

The last step uses Lemma 5. □

Now we show some simple invariants of the algorithms.

Lemma 7. *w_i only has product nodes at levels $1, ..., n-i$.*

Proof. For w_0 the statement holds vacuously. Assume the statement holds for w_i. The algorithm computes w_{i+1} by replacing each product node $\langle u \circledast v \rangle$ of w_i at level $n-i$ with a non-product node $\langle x_{n-i}, t_0, t_1 \rangle$. So w_{i+1} has no product nodes at level $n-i$. □

Lemma 8. *$[\![w_i]\!]$ is multilinear in all of $x_{n-i+1}, ..., x_n$.*

Proof. We prove that for every level $j \in \{1, ..., n\}$ and every node v of w_i at level j the polynomial $[\![v]\!]$ is a multilinear polynomial over the variables $x_{n-i+1}, ..., x_n$. We proceed by induction on j. The case $j = 0$ is trivial. For the inductive step, consider two cases. If v is a product node, then $j \leq n - i$ by Lemma 7. By Definition 5, $[\![v]\!]$ is a polynomial over $x_1, ..., x_j$. So $[\![v]\!]$ does not depend on $x_{n-i+1}, ..., x_n$, and in particular $[\![v]\!]$ is multilinear in them. If v is not a product node, then $v = \langle x_j, v_0, v_1 \rangle$ for nodes v_0, v_1 and $[\![v]\!] = [1 - x_j] \cdot [\![v_0]\!] + [x_j] \cdot [\![v_1]\!]$ (Definition 4). Again by Definition 4 $[\![v_0]\!], [\![v_1]\!]$ do not depend on x_j, and by induction hypothesis, they are multilinear in $x_{n-i+1}, ..., x_n$. So $[\![v]\!]$ is multilinear in $x_{n-i+1}, ..., x_n$ as well. □

Lemma 9. $[\![w_{i+1}]\!] = \delta_{x_{n-i}}[\![w_i]\!]$.

Proof. For any node v of w_i, let v^* denote the corresponding node in w_{i+1}. More precisely, let $u_1, v_1, ..., u_l, v_l$ denote the sequence of replaced nodes, i.e. $w_{i+1} = w_i[u_1/v_1] \cdots [u_l/v_l]$. Then $v^* := v[u_1/v_1] \cdots [u_l/v_l]$. So if v is a product node at level $n-i$, v^* denotes the BDD node that replaces it, and for the other nodes, v^* and v are the same node, except that some descendant of v have been replaced by new nodes. Note that $w_i^* = w_{i+1}$, and that $\langle x, v_0, v_1 \rangle^* = \langle x, v_0^*, v_1^* \rangle$.

We prove the stronger claim that $[\![v^*]\!] = \delta_{x_{n-i}}[\![v]\!]$ for every descendant v of w_{i+1}. We proceed by induction. For the two leaves true and false the statement clearly holds. For the induction step, we consider two cases.

- v is a product node. If $\ell(v) < n - i$, then $v^* = v$. By Lemma 8, $[\![v]\!]$ is multilinear in x_{n-i}, and so $[\![v^*]\!] = \delta_{x_{n-i}}[\![v]\!] = [\![v]\!]$. If $\ell(v) = n - i$, then $[\![v^*]\!] = \delta_{x_{n-i}}[\![v]\!]$ follows from Lemma 6.

- $v = \langle x, v_0, v_1 \rangle$. As noted above, we have $v^* = \langle x, v_0^*, v_1^* \rangle$, so

$$\begin{aligned}\delta_{x_{n-i}}[\![v]\!] &\overset{(1)}{=} \delta_{x_{n-i}}\Big([1-x]\cdot[\![v_0]\!] + [x]\cdot[\![v_1]\!]\Big)\\ &\overset{(2)}{=} [1-x]\cdot\delta_{x_{n-i}}[\![v_0]\!] + [x]\cdot\delta_{x_{n-i}}[\![v_1]\!]\\ &\overset{(3)}{=} [1-x]\cdot[\![v_0^*]\!] + [x]\cdot\delta_{x_{n-i}}[\![v_1^*]\!] \overset{(4)}{=} [\![v^*]\!]\end{aligned}$$

  where (1) expands the arithmetisation, (2) uses that either $x \neq x_{n-i}$, or $[\![v_b]\!]$ does not depend on x_{n-i} and $\delta_{x_{n-i}}[\![v_0]\!] = [\![v_0]\!]$, for $b \in \{0,1\}$, (3) uses the induction hypothesis, and (4) folds the arithmetisation back.

□

Lemma 10. *w_n is a BDD and $w_n = \mathrm{Apply}_{\circledast}(u_1, u_2)$.*

Proof. By Lemma 7, w_n has no product nodes. By Definition 5, an eBDD without product nodes is a BDD. Moreover,

$$[\![\mathrm{Apply}_{\circledast}(u_1,u_2)]\!] \equiv_b [\![\psi_1 \circledast \psi_2]\!] = [\![w_0]\!] \equiv_b \delta_{x_n}[\![w_0]\!] = [\![w_1]\!] \equiv_b \ldots \equiv_b [\![w_n]\!]$$

By Proposition 3a, both $[\![\mathrm{Apply}_{\circledast}(u_1,u_2)]\!]$ and $[\![w_n]\!]$ are multilinear polynomials. It is well-known that two multilinear polynomials that coincide on all binary inputs must be equal, so we get $[\![\mathrm{Apply}_{\circledast}(u_1,u_2)]\!] = [\![w_n]\!]$, and by Proposition 3b, $\mathrm{Apply}_{\circledast}(u_1,u_2) = w_n$ □

F. Proof of Proposition 5

Proposition 5. *Let w denote an eBDD, $\sigma : X \to \mathbb{F}$ a partial assignment, and k the number of variables assigned by σ. Then* EvaluateEBDD *evaluates the polynomial $\Pi_\sigma[\![w]\!]$ in time $\mathcal{O}\big(\mathrm{poly}(2^{n-k}) \cdot |w|\big)$.*

Proof. The algorithm memoises the computed polynomials for each node, so the total number of calls is in $\mathcal{O}(|w|)$. In each call, a constant number of operations on polynomials are performed. For any eBDD w, the polynomial $[\![w]\!]$ has maximum degree at most 2. (This follows immediately from Definition 5: a product node can only have two BDDs as children, not eBDDs.) A polynomial with $n-k$ free variables and maximum degree at most 2 can be represented using 3^{n-k} coefficients (one for each monomial), and operations can be performed efficiently on this representation. □

G. Proof of Theorem 1

Theorem 1 (Main Result)**.** *If* BDDSolver *solves an instance φ of* #CP *with n variables in time T, with $T > n|\varphi|$, then*

(a) Prover computes eBDDs for all nodes of $\mathrm{conv}(\varphi)$ *in time $\mathcal{O}(T)$,*

(b) Prover responds to Verifier's challenges in time $\mathcal{O}(nT)$, and

(c) Verifier executes CPCertify *in time $\mathcal{O}(n^2|\varphi|)$, with failure probability at most $4n|\varphi|/|\mathbb{F}|$.*

Proof. **Part (a).** This follows immediately from Proposition 4.

Part (b). Let S denote the set of descendants of $\mathrm{conv}(\varphi)$, and set

$$S_{\mathrm{bdd}} := \{\mathrm{conv}(\psi) : \psi \text{ descendant of } \varphi\} \subseteq S$$

to the descendants that correspond to BDDs (and not eBDDs). Note $|S_{\mathrm{bdd}}| = |\varphi|$ and $|S| \leq n|\varphi|$. Additionally, let B_ψ denote the eBDD representing $\psi \in S$, computed by ComputeEBDD. Note that B_ψ is a BDD if $\psi \in S_{\mathrm{bdd}}$, and (as BDDs are unique, see Proposition 3) those necessarily match the BDDs computed by BDDSolver. We thus observe $\sum_{\psi \in S_{\mathrm{bdd}}} |B_\psi| \leq T$.

For the eBDD $\psi \in S \setminus S_{\mathrm{bdd}}$, each node also appears in the computation of ComputeEBDD. In the sum $\sum_{\psi \in S} |B_\psi|$, however, a node is counted up to n times, so we get $\sum_{\psi \in S} |B_\psi| \leq nT$.

As shown in Proposition 5, responding to one challenge takes time linear in $|w|$, where w is the evaluated eBDD. Step (b.1.1) of CPCertify sends at most n challenges for each node in S_{bdd}, which are evaluated in time linear in $\sum_{\psi \in S_{\mathrm{bdd}}} nf|B_\psi| \leq nT$. Step (c) sends a challenge for each node in S, which take at most $\sum_{\psi \in S} |B_\psi| \leq nT$ time.

Part (c). As argued for part (b), Verifier sends at most $n|\varphi|$ challenges. The challenge consist of one partial assignment, which has size at most n. The failure probability follows from Proposition 2. □

H. Evaluation – Detailed Description

H.1. Instances

We used the instances from the crafted instances track of the QBF Evaluation 2022 (http://www.qbflib.org/QBFEVAL_20_DATASET.zip). These are re-used from the QBF Evaluation 2020. It should be noted that these instances are all unsatisfiable. (Instances in QDIMACS format are specified so that the outermost quantifier is existential.)

We also use the linear domino placement game used for evaluating PGBDDQ [7]. They can be obtained at https://github.com/rebryant/pgbddq-artifact. We reproduce the parameters of [7, Table 3]. On these instances we run only PGBDDQ and our tool, which are both BDD-based, and we allow both to use the provided variable ordering, which improves performance significantly.

H.2. Tools

The following four tools were compared:

tool	language	version	link
CAQE	Rust	`8b646df`	https://github.com/ltentrup/caqe
DepQBF	C	`2ad3995`	https://github.com/lonsing/depqbf
PGBDDQ	Python	`d5cbc96`	https://github.com/rebryant/pgbdd
qchecker	Python	`d5cbc96`	https://github.com/rebryant/pgbdd
QRPcheck	C	1.0.3	http://fmv.jku.at/qrpcheck/
blic (ours)	C++	`bd3d298`	https://gitlab.lrz.de/i7/blic

All solvers were run without a preprocessor, limiting their performance. In the QBF Evaluation 2022, CAQE combined with the preprocessor Bloqqer achieved first place in the crafted instances track. For comparison, we ran our tool and CAQE on the preprocessed instances: Bloqqer solves 97 of 172 by itself, of the remaining 74 instances our tool solves 34, while CAQE solves 50 (timeout of 10min).

We ran DepQBF with certificate generation enabled. More precisely, we used flags

```
-trace=bqrp -dep-man=simple -no-lazy-qpup -no-dynamic-nenofex
-no-qbce-dynamic -no-trivial-falsity -no-trivial-truth
```

These flags disable features that are not supported in conjunction with certificate generation. This reduces the performance of DepQBF: in its default configuration it can solve 104 of 172 instances (compared to 87 when certification was enabled).

To verify certificates generated by PGBDDQ we use the tool qchecker, which is part of PGBDDQ. It is specialised for the certificates PGBDDQ generates.

H.3. Time Measurement

Times are measured in the calling process. These times exceed self-reported times by about 1-2ms. Running our tool repeatedly on one (arbitrarily chosen) instance yields an average run-to-run deviation of 11ms, maximum deviation of 34ms (compared to a total running time of 3.08s).

To measure the time taken by our tool for Verifier and Prover parts of CPCertify, it is necessary to measure the contributions of each round of the interactive protocol. As the protocol is executed within a single process, these data are collected internally in our tool.

H.3.1. Comparison with results in [7].

As we use the same instances and configuration of PGBDDQ, we can compare the times we obtained with the times in [7, Table 3] to verify that we can reproduce their numbers.

Our results match their results relatively closely. Our times are between 15% and 29% slower for $10 \leq N \leq 25$, and $45 - 49\%$ slower for $N = 45$, with one outlier at 58% (a subsequent run was 54% slower, making it unlikely that the issue is intermittent). This can likely be accounted for by the faster (in terms of single-thread performance) Intel Core i7-7700K processor used in [7], and differences in main memory.