Logical Methods in Computer Science
Volume 22, Issue 2, 2026, pp. 17:1–17:27
https://lmcs.episciences.org/

Submitted Oct. 15, 2024
Published May 22, 2026

A RESOLUTION-BASED INTERACTIVE PROOF SYSTEM FOR UNSAT

PHILIPP CZERNER, JAVIER ESPARZA, VALENTIN KRASOTIN, AND ADRIAN KRAUSS

Technical University of Munich
e-mail address: {czerner, esparza, krasotin}@cit.tum.de, adrian.krauss@tum.de

Abstract. Modern SAT or QBF solvers are expected to produce correctness certificates. However, UNSAT certificates have worst-case exponential size (unless NP = coNP), and at recent SAT competitions the largest certificates of unsatisfiability are reaching terabyte size. This puts limits to the development of SAT-solving services in which a client with limited computational power sends a formula to a solver running on a powerful server, which returns a certificate to be checked by the client.

Recently, Couillard, Czerner, Esparza, and Majumdar have suggested to replace certificates with interactive proof systems based on the IP = PSPACE theorem. They have presented an interactive protocol between a prover and a verifier for an extension of QBF. The overall running time of the protocol is linear in the time needed by a standard BDD-based algorithm, and the time invested by the verifier is polynomial in the size of the formula. (So, in particular, the verifier never has to read or process exponentially long certificates). We call such an interactive protocol *competitive* with the BDD algorithm for solving QBF.

While BDD algorithms are state-of-the-art for certain classes of QBF instances, no modern (UN)SAT solver is based on BDDs. For this reason, we initiate the study of interactive certification for more practical SAT algorithms. In particular, we address the question whether interactive protocols can be competitive with some variant of resolution. We present two contributions. First, we prove a theorem that reduces the problem of finding competitive interactive protocols to finding an *arithmetisation* of formulas satisfying certain commutativity properties. (Arithmetisation is the fundamental technique underlying the IP = PSPACE theorem.) Then, we apply the theorem to give the first interactive protocol for the Davis-Putnam resolution procedure. We also report on an implementation and give some experimental results.

Parts of this work appeared previously as [CEK24].

1. Introduction

Automated reasoning tools should provide evidence of their correct behaviour. A substantial amount of research has gone into proof-producing automated reasoning tools [Heu21, Nec97, Nam01, HJM+02, BRK+22]. These works define a notion of "correctness certificate" 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 [Heu21] for a survey). We call them SAT and UNSAT certificates, respectively.

There is a fundamental asymmetry between SAT and UNSAT certificates: while SAT certificates can be checked in polynomial (even linear) time in the size of the formula, this is very unlikely to be the case for UNSAT certificates, because it implies NP = coNP. In

DOI:10.46298/LMCS-22(2:17)2026

particular, all UNSAT certificates produced by current tools (like a DRAT certificate or a resolution proof) may have exponential size in the size of the formula, and UNSAT certificates produced by current tools may have hundreds of GiB or even, in extreme cases, hundreds of TiB [HKM16]. Despite much progress on reducing the size of proofs and improving the efficiency of checking proofs (see e.g. [HJKW17, Heu21]), this fundamental asymmetry puts limits to the development of SAT-solving services in which clients with limited computational power (say, a standard laptop) send formulas to a solver running on a very powerful server, and get back a SAT or UNSAT certificate. We are interested in such a SAT-solving scenario, where the costs of solving and certificate checking are assumed by different parties, and reducing the latter is important, even if it increases the former. Notice that this stands in contrast to the cost model of a SAT-solving tool that solves the formula, produces a certificate, and checks it, where one is only interested in minimising the total time.

The $\mathsf{IP} = \mathsf{PSPACE}$ theorem, proved in 1992 by Shamir [Sha92], presents a possible fundamental solution to this problem: *interactive proofs*[1]. A language is in IP if there exists a sound and complete *interactive proof protocol* between two agents, Prover and Verifier, that Verifier can execute in randomised polynomial time [GMR85, Bab85, LFKN92, AB09]. Completeness means that, for any input in the language, an *honest prover* that truthfully follows the protocol will convince Verifier to accept the input. Soundness means that, for any input not in the language, Verifier will reject it with high probability, no matter how Prover behaves. "Conventional" certification is the special case of interactive proof in which Prover sends Verifier only one message, the certificate, and Verifier is convinced with probability 1. The $\mathsf{IP} = \mathsf{PSPACE}$ theorem implies the existence of interactive proof protocols for UNSAT in which Verifier only invests polynomial time *in the size of the formula*. This is not the case for conventional certification protocols where Prover produces a certificate, like a resolution proof or a DRAT certificate. Indeed, while these certificates can be checked in linear time *in the size of the certificate*, they have worst-case exponential size in the size of the formula, and so Verifier needs exponential time in the worst case.

Despite its theoretical promise, the automated reasoning community has not yet developed tools for UNSAT or QBF with interactive proof protocols. In a recent paper, Couillard, Czerner, Esparza, and Majumdar venture a possible explanation [CCEM23]. They observe that the interactive certification protocols described in the literature have been designed to prove asymptotic complexity results, for which one works with a cost model that only counts the resources used by Verifier, that is, a model in which the steps of Prover have zero cost. Intuitively, in this approach the goal is to make Verifier faster, *regardless* of the cost for Prover. These leads to very inefficient Provers that typically construct the complete truth table of the formula, making them incompatible with automated reasoning tools. Indeed, after efficiently deciding that a given formula is UNSAT using a state-of-the-art algorithm, Prover would have to compute the truth table of the formula. The runtime of Prover would be *best-case* exponential in the number of variables, making the tool useless in practice.

Couillard *et al.* propose to search for interactive proof protocols where Prover uses algorithms that, while worst-case exponential, like any known algorithm for QBF, are more efficient in practice than brute force. In [CCEM23] they consider the standard BDD-based algorithm for QBF and design an interactive protocol based on it.

In this paper, we initiate the study of interactive certification for UNSAT for Provers based on more efficient SAT-solving algorithms than pure brute-force. For this, given an

[1] In our context it would be more adequate to speak of interactive certification, but we use the standard terminology.

algorithm *Alg* and an interactive protocol P, both for UNSAT, we say that P is *competitive* for *Alg* if the ratio between the runtime of Prover in P and the runtime of *Alg* on inputs of length n is bounded by a polynomial in n. So, loosely speaking, if P is competitive for *Alg*, then one can add interactive verification to *Alg* with only polynomial overhead. For example, Provers based on computing the truth table are not competitive with respect to any (reasonable variant of) resolution: Indeed, it is easy to construct a family of formulas with an arbitrary number of variables such that resolution takes polynomial time in the size of the formula, while constructing the truth table takes exponential time. This raises the question whether it is possible to design interactive protocols that are competitive for algorithms more efficient than brute-force methods.

We present the first interactive protocol competitive for (a simplified version of) the well-known Davis-Putnam procedure—see e.g. Section 2.9 of [Har09]. Our version fixes a total order on variables, resolves exhaustively with respect to the next variable, say x, and then "locks" all clauses containing x or $\neg x$, ensuring that they are never resolved again w.r.t. any variable. For this, we first present the gist of our idea in general terms. We introduce a technique that, given an algorithm *Alg* for UNSAT satisfying certain conditions, constructs an interactive protocol that is competitive with respect to *Alg*. Let us be more precise. We consider algorithms for UNSAT that, given a formula φ_0, construct a sequence $\varphi_0, \varphi_1, ..., \varphi_k$ of formulas such that φ_i is equisatisfiable to φ_{i+1}, and there is a polynomial algorithm that decides if φ_k is unsatisfiable. Our interactive protocols are based on the well-known idea of encoding the formulas in this sequence as polynomials over a finite field in such a way that the truth value of the formula for a given assignment is determined by the value of the polynomial on that assignment. The encoding procedure is called *arithmetisation*, and has been extensively used for designing both SAT-solving algorithms (see e.g. [BN21] on algebraic proof systems) and interactive proof protocols [GMR85, Bab85, LFKN92, AB09]. We introduce the notion of an arithmetisation *compatible* with a given algorithm. Loosely speaking, compatibility means that for each step $\varphi_i \mapsto \varphi_{i+1}$, there is an operation on polynomials mimicking the operation on formulas that transforms φ_i into φ_{i+1}. We prove that the problem of finding a competitive interactive protocol for a given algorithm *Alg* for UNSAT reduces to finding an arithmetisation compatible with *Alg*. We then show that, while standard arithmetisations are not compatible with the Davis-Putnam algorithm, a non-standard arithmetisation is. In our opinion, this is the main insight of our paper: in order to find interactive protocols for algorithms for UNSAT, one can profit from non-standard arithmetisations.

The Davis-Putnam procedure was proposed in the 1960s, and constitutes a first link in a chain of increasingly efficient SAT-solving procedures that includes different flavours of resolution, DPLL and, ultimately, conflict-driven clause learning (CDCL). Modern techniques for solving UNSAT are several orders of magnitude faster than Davis-Putnam, and so our results do not yet have any practical implication for SAT solving: in order to achieve polynomial time for Verifier, we would need to use an algorithm for Prover that, while much better than brute force, is still too inefficient compared to the current state of the art. Whether practical interactive protocols can be obtained for other links of the chain remains open. On the one hand, our technique does not extend to them, and so, besides non-standard arithmetisations, new ideas are required. On the other hand, complexity theory offers several candidates for such ideas: besides the `IP` model used in our paper, one may also consider the multi-prover model `MIP`, as well as probabilistically checkable proofs.

In the last section of the paper, we report on an implementation of our interactive protocol. We conduct some experiments comparing the time and memory consumption of Prover and Verifier for Davis-Putnam in both the conventional and interactive certification settings. In the conventional setting, Prover computes a Davis-Putnam resolution proof and sends it to Verifier, who checks its correctness. In the interactive setting, Prover still computes the resolution proof but, instead of sending it to Verifier, engages with Verifier in our competitive interactive protocol. Roughly speaking, our results indicate that Prover is slower in the interactive setting by a nearly constant factor, while Verifier is faster in the interactive setting by several orders of magnitude.

The paper is structured as follows. Section 2 contains preliminaries. Section 3 presents interactive proof systems and defines interactive proof systems competitive with a given algorithm. Section 4 defines our version of the Davis-Putnam procedure. Section 5 introduces arithmetisations, and defines arithmetisations compatible with a given algorithm. Section 6 presents an interactive proof system for Davis-Putnam. Section 7 reports on our implementation and experiments. Section 8 contains conclusions.

2. Preliminaries

Multisets. A multiset over a set S is a mapping $m\colon S \to \mathbb{N}$. We also write multisets using set notation, for example we write $\{x, x, y\}$ or $\{2 \cdot x, y\}$. Given two multisets m_1 and m_2, we define $m_1 \oplus m_2$ as the multiset given by $(m_1 \oplus m_2)(s) = m_1(s) + m_2(s)$ for every $s \in S$, and $m_1 \ominus m_2$ as the multiset given by $(m_1 \ominus m_2)(s) = \max\{0, m_1(s) - m_2(s)\}$ for every $s \in S$.

Formulas, CNF, and resolution. A Boolean *variable* has the form x_i where $i = 1, 2, 3, ...$ Boolean *formulas* are defined inductively: *true*, *false* and variables are formulas; if φ and ψ are formulas, then so are $\neg\varphi$, $\varphi \vee \psi$, and $\varphi \wedge \psi$. A *literal* is a variable or the negation of a variable. A formula φ is in *conjunctive normal form (CNF)* if it is a conjunction of disjunctions of literals. We represent a formula in CNF as a *multiset* of *clauses* where a clause is a *multiset* of literals. For example, the formula $(x \vee x \vee x \vee \neg y) \wedge z \wedge z$ is represented by the multiset $\{\{3x, \neg y\}, 2\{z\}\}$.

Remark 2.1. Usually CNF formulas are represented as *sets* of clauses, which are defined as *sets* of literals. Algorithms that manipulate CNF formulas using the set representation are assumed to silently remove duplicate formulas or duplicate literals. In this paper, due to the requirements of interactive protocols, we need to make these steps explicit. In particular, we use multiset notation for clauses. For example, $C(x)$ denotes the number of occurrences of x in C.

We assume in the paper that formulas are in CNF. Abusing language, we use φ to denote both a (CNF) formula and its multiset representation.

Resolution. Resolution is a proof system for CNF formulas. Given a variable x, a clause C containing exactly one occurrence of x and a clause C' containing exactly one occurrence of $\neg x$, the *resolvent* of C and C' with respect to x is the clause $Res_x(C, C') := (C \ominus \{x\}) \oplus (C' \ominus \{\neg x\})$.

For example, $Res_x(\{x, \neg y, z\}, \{\neg x, \neg w\}) = \{\neg y, z, \neg w\}$. It is easy to see that $C \wedge C'$ and $Res_x(C, C')$ are equisatisfiable. A *resolution refutation* for a formula φ is a sequence of clauses ending in the empty clause whose elements are either clauses of φ or resolvents of two previous clauses in the sequence. It is well known that φ is unsatisfiable iff there exists

a resolution refutation for it. There exist families of formulas, like the pigeonhole formulas, for which the length of the shortest resolution refutation grows exponentially in the size of the formula, see e.g. [Hak85, BT88].

Polynomials. Interactive protocols make extensive use of polynomials over a finite field $\mathbb{F}$. Let X be a finite set of variables. We use $x, y, z, ...$ for variables and $p, p_1, p_2, ...$ for polynomials. We use the following operations on polynomials:

- *Sum, difference, and product*, denoted $p_1 + p_2$, $p_1 - p_2$, $p_1 \cdot p_2$, 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 the variable x to the field element a in the polynomial p, e.g. $\pi_{[x:=5]}(3xy - z^2) = 15y - z^2$.

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

The following lemma is at the heart of all interactive proof protocols. Intuitively, it states that if two polynomials are different, then they are different for almost every input. Therefore, by picking an input at random, one can check polynomial equality with high probabillity.

Lemma 2.2 (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 $d/|\mathbb{F}|$. □

3. Interactive Proof Systems

An *interactive protocol* is a sequence of interactions between two parties: *Prover* and *Verifier*. Prover has unbounded computational power, whereas Verifier is a randomised, polynomial-time algorithm. Initially, the parties share an input x that Prover claims belongs to a given language L (e.g. UNSAT). The parties alternate in sending messages to each other according to a protocol. Intuitively, Verifier repeatedly asks Prover to send informations. At the end of the protocol, Verifier accepts or rejects the input.

Formally, let V, P denote (randomised) online algorithms, i.e. given a sequence of inputs $m_1, m_2, ... \in \{0, 1\}^*$ they compute a sequence of outputs, e.g. $V(m_1), V(m_1, m_2), ...$ We say that $(m_1, ..., m_{2k})$ is a *k-round interaction*, with $m_1, ..., m_{2k} \in \{0, 1\}^*$, if $m_{i+1} = V(m_1, ..., m_i)$ for odd i and $m_{i+1} = P(m_1, ..., m_i)$ for even i.

The *output* $\mathsf{out}_{V,P,k}(x)$ is m_{2k}, where $(m_1, ..., m_{2k})$ is a k-round interaction with $m_1 = x$. We also define the *Verifier-time* $\mathsf{vtime}_{V,P,k}(x)$ as the expected time it takes V to compute $m_2, m_4, ..., m_{2k}$ for any k-round interaction $(m_1, ..., m_{2k})$ with $m_1 = x$. We define the *Prover-time* $\mathsf{ptime}_{V,P,k}(x)$ analogously.

Let L be a language and $p : \mathbb{N} \to \mathbb{N}$ a polynomial. A tuple (V, P_H, p) is an *interactive protocol for L* if for each $x \in \{0, 1\}^*$ of length n we have $\mathsf{vtime}_{V,P_H,p(n)}(x) \in \mathcal{O}(\mathrm{poly}\, n)$ and:
(1) (*Completeness*) $x \in L$ implies $\mathsf{out}_{V,P_H,p(n)}(x) = 1$ with probability 1, and

(2) (*Soundness*) $x \notin L$ implies that for all P we have $\mathsf{out}_{V,P,p(n)}(x) = 1$ with probability at most 2^{-n}.

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 it.

IP is the class of languages for which there exists an interactive protocol. It is known that $\mathsf{IP} = \mathsf{PSPACE}$ [LFKN92, Sha92], 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 formulas; in particular, the honest Prover is a polynomial-space, exponential-time algorithm.

3.1. **Competitive Interactive Protocols.** In an interactive protocol there are no restrictions on the running time of Prover. The existence of an interactive protocol for some coNP-complete problem in which Prover runs in polynomial time would imply e.g. $\mathsf{NP} \subseteq \mathsf{BPP}$. Since this is widely believed to be false, Provers are allowed to run in exponential time, as in the proofs of [LFKN92, Sha92]. However, while all known approaches for UNSAT use exponential time in the worst case, some perform much better in practice than others. For example, the Provers of [LFKN92, Sha92] run in exponential time *in the best case*. This motivates our next definition: instead of stating that Prover must always be efficient, we say that it must have a bounded overhead *compared* to some given algorithm *Alg*.

Formally, let $L \subseteq \{0,1\}^*$ be a language, let *Alg* be an algorithm for L, and let (V, P_H, p) be an interactive protocol for L. We say that (V, P_H, p) is *competitive with Alg* if for every input $x \in \{0,1\}^*$ of length n we have $\mathsf{ptime}_{V,P_H,p(n)}(x) \in \mathcal{O}(\mathrm{poly}(n)T(x))$, where $T(x)$ is the time it takes *Alg* to run on input x.

Recently, Couillard, Czerner, Esparza and Majumdar [CCEM23] have constructed an interactive protocol for QBF that is competitive with BDDSolver, the straightforward BDD-based algorithm that constructs a BDD for the satisfying assignments of each subformula, starting at the leaves of the syntax tree and progressively moving up. In this paper, we will investigate UNSAT and give an interactive protocol that is competitive with DavisPutnam, a decision procedure for UNSAT based on a restricted version of resolution.

4. The Davis-Putnam Resolution Procedure

We introduce the variant of the Davis-Putnam resolution procedure [DP60, Har09] for which we later construct a competitive interactive protocol[2]. Recall that in our setting, clauses are multisets, and given a clause C and a literal l, $C(l)$ denotes the number of occurrences of l in C.

Definition 4.1. Let x be a variable. *Full x-resolution* is the procedure that takes as input a formula φ satisfying $C(x) + C(\neg x) \leq 1$ for every clause C, and returns the formula $R_x(\varphi)$ computed as follows:

[2]In Harrison's book [Har09], the Davis-Putnam procedure consists of three rules. The version in Definition 4.1 uses only Rule III, which is sometimes called the Davis-Putnam resolution procedure. Unfortunately, at the time of writing this paper, the Wikipedia article for the Davis-Putnam algorithm uses a different terminology (even though it cites [Har09]): it calls the three-rule procedure the Davis-Putnam *algorithm*, and the algorithm consisting only of Rule III the Davis-Putnam *procedure*.

Algorithm 1 DavisPutnam(φ)

for $i = 1, ..., n$ **do**
  $\varphi := R_{x_i}(\varphi)$
  for $j = i + 1, ..., n$ **do**
    $\varphi := C_{x_j}(\varphi)$
if $\square \in \varphi$ **then**
  return "unsatisfiable"
else
  return "satisfiable"

Table 1: Run of DavisPutnam on an input φ, and arithmetisation of the intermediate formulas.

Step	Formula	Arithmetisation
Inp.	$\varphi = \{\{x,y\},\{x,\neg y,\neg z\},\{\neg x,\neg z\},$ $\{\neg x,\neg y,\neg z\},\{y,z\},\{\neg y,z\}\}$	$\mathcal{B}(\varphi) = (1-x)(1-y)+(1-x)y^3z^3+x^3z^3$ $+x^3y^3z^3+(1-y)(1-z)+y^3(1-z)$
R_x	$\varphi_1 = \{\{y,\neg z\},\{y,\neg y,\neg z\},\{\neg y,\neg z,\neg z\}$ $\{\neg y,\neg z,\neg y,\neg z\},\{y,z\},\{\neg y,z\}\}$	$\mathcal{B}(\varphi_1) = (1-y)z^3+(1-y)y^3z^3+y^3z^6$ $+y^6z^6+(1-y)(1-z)+y^3(1-z)$
C_y	$\varphi_2 = \{\{y,\neg z\},2\cdot\{\neg y,\neg z,\neg z\},$ $\{y,z\},\{\neg y,z\}\}$	$\mathcal{B}(\varphi_2) = (1-y)z^3+2y^3z^6$ $+(1-y)(1-z)+y^3(1-z)$
C_z	$\varphi_3 = \{\{y,\neg z\},2\cdot\{\neg y,\neg z\}$ $\{y,z\},\{\neg y,z\}\}$	$\mathcal{B}(\varphi_3) = (1-y)z^3+2y^3z^3$ $+(1-y)(1-z)+y^3(1-z)$
R_y	$\varphi_4 = \{2\cdot\{\neg z,\neg z\},3\cdot\{\neg z,z\},\{z,z\}\}$	$\mathcal{B}(\varphi_4) = 2z^6+3z^3(1-z)+(1-z)^2$
C_z	$\varphi_5 = \{2\cdot\{\neg z\},\{z\}\}$	$\mathcal{B}(\varphi_5) = 2z^3+(1-z)$
R_z	$\varphi_6 = \{2\cdot\square\}$	$\mathcal{B}(\varphi_6) = 2$

(1) For every pair C_1, C_2 of clauses of φ such that $x \in C_1$ and $\neg x \in C_2$, add to φ the resolvent w.r.t. x of C_1 and C_2 (i.e. set $\varphi := \varphi \oplus Res_x(C_1, C_2)$).
(2) Remove all clauses containing x or $\neg x$.

Full x-cleanup is the procedure that takes as input a formula φ satisfying $C(x) + C(\neg x) \leq 2$ for every clause C, and returns the formula $C_x(\varphi)$ computed as follows:

(1) Remove from φ all clauses containing both x and $\neg x$.
(2) Remove from each remaining clause all duplicates of x or $\neg x$.

The *Davis-Putnam resolution procedure* is the algorithm for UNSAT that, given a total order $x_1 \prec x_2 \prec \cdots \prec x_n$ on the variables of an input formula φ, executes Algorithm 1. The algorithm assumes that φ is a set of sets of literals, that is, clauses contain no duplicate literals, and φ contains no duplicate clauses. We let $\square$ denote the empty clause.

Observe that while the initial formula contains no duplicate clauses, the algorithm may create them, and they are not removed.

Example 4.2. Table 1 shows on the left a run of DavisPutnam on a formula φ with three variables and six clauses. The right column is explained in Section 6.1.

It is well-known that the Davis-Putnam resolution procedure is complete, but we give a proof suitable for our purposes. Let $\varphi[x := \mathit{true}]$ denote the result of replacing all occurrences

of x in φ by *true* and all occurrences of $\neg x$ by *false*. Define $\varphi[x := \mathit{false}]$ reversely. Further, let $\exists x\varphi$ be an abbreviation of $\varphi[x := \mathit{true}] \vee \varphi[x := \mathit{false}]$. We have:

Lemma 4.3. *Let x be a variable and φ a formula in CNF such that $C(x) + C(\neg x) \leq 1$ for every clause C. Then $R_x(\varphi) \equiv \exists x\varphi$.*

Proof. Let $C_1, ..., C_k$ be the clauses of φ. We have

$$\begin{aligned}
\exists x\varphi &\equiv \varphi[x := \mathit{true}] \vee \varphi[x := \mathit{false}] \\
&\equiv \Big(\bigwedge_{i\in[k]} C_i[x := \mathit{true}] \Big) \vee \Big(\bigwedge_{j\in[k]} C_j[x := \mathit{false}] \Big) \\
&\equiv \bigwedge_{i,j\in[k]} \big(C_i[x := \mathit{true}] \vee C_j[x := \mathit{false}] \big) \\
&\equiv \bigwedge_{i\in[k],\, x,\neg x\notin C_i} C_i \wedge \bigwedge_{i,j\in[k],\neg x\in C_i, x\in C_j} \big(C_i[x := \mathit{true}] \vee C_j[x := \mathit{false}] \big) \\
&\equiv R_x(\varphi).
\end{aligned}$$

For the second-to-last equivalence, consider a clause C_i containing neither x nor $\neg x$. Then $C_i \vee C_i$ is a clause of $\bigwedge_{i,j\in[k]} \big(C_i[x := \mathit{true}] \vee C_j[x := \mathit{false}] \big)$, and it subsumes any other clause of the form $C_i \vee C_j$. The first conjunct of the penultimate line contains these clauses. Furthermore, if C_i contains x or if C_j contains $\neg x$, then the disjunction $C_i[x := \mathit{true}] \vee C_j[x := \mathit{false}]$ is a tautology and can thus be ignored. It remains to consider the pairs (C_i, C_j) of clauses such that $\neg x \in C_i$ and $x \in C_j$. This is the second conjunct. □

Lemma 4.4. *Let x be a variable and φ a formula in CNF such that $C(x) + C(\neg x) \leq 2$ for every clause C. Then $C_x(\varphi) \equiv \varphi$.*

Proof. Since $x \vee \neg x \equiv \mathit{true}$, a clause containing both x and $\neg x$ is valid and thus can be removed. Furthermore, duplicates of x in a clause can be removed because $x \vee x \equiv x$. □

Theorem 4.5. DavisPutnam *is sound and complete.*

Proof. Let φ be a formula over the variables $x_1, ..., x_n$. By Lemmas 4.3 and 4.4, after termination the algorithm arrives at a formula without variables equivalent to $\exists x_n \cdots \exists x_1 \varphi$. This final formula is equivalent to the truth value of whether φ is satisfiable; that is, φ is unsatisfiable iff the final formula contains the empty clause. □

5. Constructing Competitive Interactive Protocols for UNSAT

We consider algorithms for UNSAT that, given a formula, execute a sequence of *macrosteps*. Throughout this section, we use DavisPutnam as running example.

Definition 5.1. A *macrostep* is a partial mapping M that transforms a formula φ into a formula $M(\varphi)$ equisatisfiable to φ.

The first macrostep is applied to the input formula. The algorithm accepts if the formula returned by the last macrostep is equivalent to *false*. Clearly, all these algorithms are sound.

Example 5.2. The macrosteps of DavisPutnam are R_x and C_x for each variable x. On a formula with n variables, DavisPutnam executes exactly $\frac{n(n+1)}{2}$ macrosteps.

We present an abstract design framework to obtain competitive interactive protocols for these macrostep-based algorithms. As in [LFKN92, Sha92, CCEM23], the framework is based on *arithmetisation* of formulas. Arithmetisations are mappings that assign to a formula a polynomial with integer coefficients. In protocols, Verifier asks Prover to return the result of evaluating polynomials obtained by arithmetising formulas not over the integers, but over a prime field $\mathbb{F}_q$, where q is a sufficiently large prime. An arithmetisation is useful for the design of protocols if the value of the polynomial on a *binary input*, that is, an assignment that assigns 0 or 1 to every variable, determines the truth value of the formula under the assignment. We are interested in the following class of arithmetisations, just called *arithmetisations* for brevity:

Definition 5.3. Let $\mathcal{F}$ and $\mathcal{P}$ denote the sets of formulas and polynomials over a set of variables. An *arithmetisation* is a mapping $\mathcal{A}\colon \mathcal{F} \to \mathcal{P}$ such that for every formula φ and every assignment σ to its variables:

(a) σ satisfies φ iff $\Pi_\sigma \mathcal{A}(\varphi) = 0$,[3] and
(b) $\Pi_\sigma \mathcal{A}(\varphi) \pmod q$ can be computed in time $\mathcal{O}(|\varphi| \operatorname{polylog} q)$ for any prime q.

In particular, two formulas φ, ψ over the same set of variables are equivalent if and only if for every binary assignment σ, $\Pi_\sigma \mathcal{A}(\varphi)$ and $\Pi_\sigma \mathcal{A}(\psi)$ are either both zero or both nonzero.

Example 5.4. Let $\mathcal{A}$ be the mapping inductively defined as follows:

$$\begin{array}{lll} \mathcal{A}(\mathit{true}) := 0 & \mathcal{A}(\neg x) := x & \mathcal{A}(\varphi_1 \wedge \varphi_2) := \mathcal{A}(\varphi_1) + \mathcal{A}(\varphi_2) \\ \mathcal{A}(\mathit{false}) := 1 & \mathcal{A}(x) := 1 - x & \mathcal{A}(\varphi_1 \vee \varphi_2) := \mathcal{A}(\varphi_1) \cdot \mathcal{A}(\varphi_2). \end{array}$$

For example, $\mathcal{A}((x \vee \mathit{false}) \wedge \neg x) = ((1 - x) \cdot 1) + x = 1$. It is easy to see that $\mathcal{A}$ is an arithmetisation in the sense of Definition 5.3. Notice that $\mathcal{A}$ can map equivalent formulas to different polynomials. For example, $\mathcal{A}(\neg x) = x$ and $\mathcal{A}(\neg x \wedge \neg x) = 2x$.

We define when an arithmetisation $\mathcal{A}$ is compatible with a macrostep M.

Definition 5.5. Let $\mathcal{A}\colon \mathcal{F} \to \mathcal{P}$ be an arithmetisation and let $M\colon \mathcal{F} \to \mathcal{F}$ be a macrostep. $\mathcal{A}$ is *compatible with* M if there exists a partial mapping $P_M\colon \mathcal{P} \to \mathcal{P}$ and a *pivot variable* $x \in X$ satisfying the following conditions:

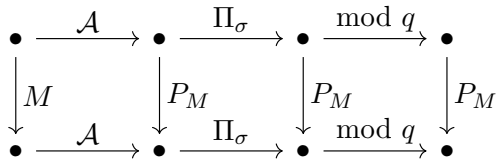
(a) P_M *simulates* M: For every formula φ where $M(\varphi)$ is defined, we have $\mathcal{A}(M(\varphi)) = P_M(\mathcal{A}(\varphi))$.
(b) P_M *commutes with partial evaluations*: For every polynomial p and every assignment $\sigma\colon X \setminus \{x\} \to \mathbb{Z}$: $\Pi_\sigma(P_M(p)) = P_M(\Pi_\sigma(p))$.
(c) $P_M(p \pmod q) = P_M(p) \pmod q$ for any prime q. [4]
(d) P_M can be computed in polynomial time.

An arithmetisation $\mathcal{A}$ is *compatible* with *Alg* if it is compatible with every macrostep executed by *Alg*.

[3] In most papers one requires that σ satisfies φ iff $\Pi_\sigma \mathcal{A}(\varphi) = 1$. Because of our later choice of arithmetisations, we prefer $\Pi_\sigma \mathcal{A}(\varphi) = 0$.

[4] We implicitly extend P_M to polynomials over $\mathbb{F}_q$ in the obvious way: we consider the input p as a polynomial over $\mathbb{Z}$ by selecting the smallest representative in $\mathbb{N}$ for each coefficient, apply P_M, and then take the coefficients of the output polynomial modulo q.

Graphically, an arithmetisation $\mathcal{A}$ is compatible with M if there exists a mapping P_M such that the following diagram commutes:



We can now state and prove the main theorem of this section.

Theorem 5.6. *Let Alg be an algorithm for UNSAT and let $\mathcal{A}$ be an arithmetisation compatible with Alg such that for every input φ*

(a) *Alg executes a sequence of $k \in \mathcal{O}(\text{poly}|\varphi|)$ macrosteps, which compute a sequence $\varphi_0, \varphi_1, ..., \varphi_k$ of formulas with $\varphi_0 = \varphi$,*
(b) *$\mathcal{A}(\varphi_i)$ has maximum degree at most $d \in \mathcal{O}(\text{poly}|\varphi|)$, for any i, and*
(c) *$\mathcal{A}(\varphi_k)$ is a constant polynomial such that $|\mathcal{A}(\varphi_k)| \leq 2^{2^{\mathcal{O}(|\varphi|)}}$.*

Then there is an interactive protocol for UNSAT that is competitive with Alg.

To prove Theorem 5.6, we first define a generic interactive protocol for UNSAT depending only on *Alg* and $\mathcal{A}$, and then prove that it satisfies the properties of an interactive proof system: if φ is unsatisfiable and Prover is honest, Verifier always accepts; and if φ is satisfiable, then Verifier accepts with probability at most $2^{-|\varphi|}$, regardless of Prover.

5.1. **Interactive Protocol.** The interactive protocol for a given algorithm *Alg* operates on polynomials over a prime finite field, instead of the integers. Given a prime q, we write $\mathcal{A}_q(p) := \mathcal{A}(p) \pmod q$ for the polynomial over $\mathbb{F}_q$ (the finite field with q elements) that one obtains by taking the coefficients of $\mathcal{A}(p)$ modulo q.

At the start of the protocol, Prover sends Verifier a prime q, and then exchanges messages with Verifier about the values of polynomials over $\mathbb{F}_q$, with the goal of convincing Verifier that $\mathcal{A}(\varphi_k) \neq 0$ by showing $\mathcal{A}_q(\varphi_k) \neq 0$ instead. The following lemma demonstrates that this is both sound and complete; (a) shows that a dishonest Prover cannot cheat in this way, and (b) shows that an honest Prover can always convince Verifier.

Lemma 5.7. *Let φ_k be the last formula computed by Alg.*

(a) *For every prime q, we have that $\mathcal{A}_q(\varphi_k) \neq 0$ implies that φ is unsatisfiable.*
(b) *If φ is unsatisfiable, then there exists a prime q s.t. $\mathcal{A}_q(\varphi_k) \neq 0$.*

Proof. For every prime q, if $\mathcal{A}_q(\varphi_k) \neq 0$ then $\mathcal{A}(\varphi_k) \neq 0$. For the converse, pick any prime q larger than $\mathcal{A}(\varphi_k)$. □

Note that we show later that actually Prover can always find a suitable q much smaller than $\mathcal{A}(\varphi_k)$ (see Lemma 5.13).

We let $\varphi = \varphi_0, \varphi_1, ..., \varphi_k$ denote the sequence of formulas computed by *Alg*, and d the bound on the polynomials $\mathcal{A}(\varphi_i)$ of Theorem 5.6. Observe that the formulas in the sequence can be exponentially larger than φ, and so Verifier cannot even read them. For this reason, during the protocol Verifier repeatedly sends Prover partial assignments σ chosen at random, and Prover sends back to Verifier *claims* about the formulas of the sequence of the form $\Pi_\sigma \mathcal{A}_q(\varphi_i) = w$. The first claim is about φ_k, the second about φ_{k-1}, and so on. Verifier stores the current claim by maintaining variables i, w, and σ. The protocol guarantees that

Table 2: An interactive protocol for an algorithm for UNSAT describing the behaviour of Verifier and the honest Prover.

(1) Prover picks an appropriate prime q; i.e. a prime s.t. $\mathcal{A}_q(\varphi_k) \neq 0$, where φ_k is the last formula computed by *Alg*. (The algorithm to compute q is given later.)
(2) Prover sends both q and $\mathcal{A}_q(\varphi_k)$ to Verifier. If Prover sends $\mathcal{A}_q(\varphi_k) = 0$, Verifier rejects.
(3) Verifier sets $i := k$, $w := \mathcal{A}_q(\varphi_k)$ (sent by Prover in the previous step), and σ to an arbitrary assignment. (Since initially $\mathcal{A}_q(\varphi_k)$ is a constant, σ is irrelevant.)
(4) For each $i = k, ..., 1$, the claim about φ_i is reduced to a claim about φ_{i-1}:
4.1 Let x denote the pivot variable of M_i and set σ' to the partial assignment that is undefined on x and otherwise matches σ. Prover sends the polynomial $p := \Pi_{\sigma'}\mathcal{A}_q(\varphi_{i-1})$, which is a univariate polynomial in x.
4.2 If the degree of p exceeds d or $\pi_{[x:=\sigma(x)]}P_{M_i}(p) \neq w$, Verifier rejects. Otherwise, Verifier chooses an $r \in \mathbb{F}_q$ uniformly at random and updates $w := \pi_{[x:=r]}p$ and $\sigma(x) := r$.
(5) Finally, Verifier checks the claim $\Pi_\sigma\mathcal{A}_q(\varphi_0) = w$ by itself and rejects if it does not hold. Otherwise, Verifier accepts.

the claim about φ_i *reduces* to the claim about φ_{i-1}, in the following sense: if a dishonest Prover makes a false claim about φ_i but a true claim about φ_{i+1}, Verifier detects with high probability that the claim about φ_i is false and rejects. Therefore, in order to make Verifier accept a satisfiable formula φ, a dishonest Prover must keep making false claims, and in particular make a false last claim about $\varphi_0 = \varphi$. The protocol also guarantees that a false claim about φ_0 is always detected by Verifier.

The protocol is described in Table 2. It presents the steps of Verifier and an *honest* Prover.

Example 5.8. In the next section we use the generic protocol of Table 2 to give an interactive protocol for $Alg :=$ DavisPutnam, using an arithmetisation called $\mathcal{B}$. Table 3 shows a possible run of this protocol on the formula φ of Table 1. We can already explain the shape of the run, even if $\mathcal{B}$ is not defined yet.

Recall that on input φ, DavisPutnam executes six steps, shown on the left column of Table 1, that compute the formulas $\varphi_1, ..., \varphi_6$. Each row of Table 3 corresponds to a round of the protocol. In round i, Prover sends Verifier the polynomial p_i corresponding to the claim $\Pi_\sigma\mathcal{A}_q(\varphi_i)$ (column Honest Prover). Verifier performs a check on the claim (line with $\stackrel{?}{=}$). If the check passes, Verifier updates σ and sends it to Prover as the assignment to be used for the next claim.

5.2. **The interactive protocol is correct and competitive with *Alg*.** We need to show that the interactive protocol of Table 2 is correct and competitive with *Alg*. We do so by means of a sequence of lemmas. Lemmas 5.10-5.12 bound the error probability of Verifier and the running time of both Prover and Verifier as a function of the prime q. Lemma 5.13

shows that Prover can efficiently compute a suitable prime. The last part of the section combines the lemmas to prove Theorem 5.6.

Completeness. We start by establishing that an honest Prover can always convince Verifier.

Lemma 5.9. *If φ is unsatisfiable and Prover is honest (i.e. acts as described in Table 2), then Verifier accepts with probability 1.*

Proof. We show that Verifier accepts. First we show that Verifier does not reject in step 2, i.e. that $\mathcal{A}_q(\varphi_k) \neq 0$. Since φ is unsatisfiable by assumption, by Definition 5.1 we have that φ_k is unsatisfiable. Then, Definition 5.3(a) implies $\mathcal{A}_q(\varphi_k) \neq 0$ (note that $\mathcal{A}_q(\varphi_k)$ is constant, by Theorem 5.6(c)).

Let us now argue that the claim w Verifier tracks (i.e., the claim given by the current values of the variables) is always true. In step 3, it is initialised with $w := \mathcal{A}_q(\varphi_k)$, so the claim is true at that point.

Next, let w now be the claim about φ_i with $1 \leq i \leq k$, i.e. $w = \Pi_\sigma \mathcal{A}_q(\varphi_i)$ holds by induction hypothesis, and Verifier wants to reduce it to a claim about φ_{i-1} (performs step 4). In step 4.2, Verifier checks $\pi_{[x:=\sigma(x)]} P_{M_i}(p) \stackrel{?}{=} w$. As Prover is honest, it sent $p := \Pi_{\sigma'} \mathcal{A}_q(\varphi_{i-1})$ in step 4.1; so the check is equivalent to

$$\begin{aligned} w &\stackrel{?}{=} \pi_{[x:=\sigma(x)]} P_{M_i}(\Pi_{\sigma'} \mathcal{A}_q(\varphi_{i-1})) && \text{(Definition 5.5(b))} \\ &= \Pi_\sigma P_{M_i}(\mathcal{A}_q(\varphi_{i-1})) && \text{(Definition 5.5(a,c))} \\ &= \Pi_\sigma \mathcal{A}_q(M_i(\varphi_{i-1})) = \Pi_\sigma \mathcal{A}_q(\varphi_i) \end{aligned}$$

By induction hypothesis $w = \Pi_\sigma \mathcal{A}_q(\varphi_i)$ holds, and thus Verifier does not reject.

In step 4.2, Verifier also updates the claim about φ_i to a claim about φ_{i-1}. It selects a random number r and sets $w := \pi_{[x:=r]} p$ and $\sigma(x) := r$. Due to $p = \Pi_{\sigma'} \mathcal{A}_q(\varphi_{i-1})$, the new claim, $w = \Pi_\sigma \mathcal{A}_q(\varphi_{i-1})$, will hold for all possible values of r.

In step 5, we have a claim about φ_0, i.e. $w = \Pi_\sigma \mathcal{A}_q(\varphi_0)$, and the invariant that the claim is true, so Verifier will accept. □

Probability of error. Establishing soundness is more involved. The key idea of the proof (which is the same idea as for other interactive protocols) is that for Verifier to accept erroneously, the claim it tracks must at some point be true. However, initially the claim is false. It thus suffices to show that each step of the algorithm is unlikely to turn a false claim into a true one.

Lemma 5.10. *Let $\mathcal{A}, d, k$ as in Theorem 5.6. If φ is satisfiable, then for any Prover, honest or not, Verifier accepts with probability at most $dk/q \in \mathcal{O}(\text{poly}(|\varphi|)/q)$.*

Proof. Let $i \in \{k, ..., 1\}$, let σ, w denote the values of these variables at the beginning of step 4.1 in iteration i, and let σ', w' denote the corresponding (updated) values at the end of step 4.2.

We say that Prover *tricks* Verifier at iteration i if the claim tracked by Verifier was false at the beginning of step 4 and is true at the end, i.e. $\Pi_\sigma \mathcal{A}_q(\varphi_i) \neq w$ and $\Pi_{\sigma'} \mathcal{A}_q(\varphi_{i-1}) = w'$.

The remainder of the proof is split into three parts.

(a) If Verifier accepts, it was tricked.
(b) For any i, Verifier is tricked at iteration i with probability at most d/q.

(c) Verifier is tricked with probability at most $dk/q \in \mathcal{O}(\text{poly}(|\varphi|)/q)$.

Part (a). If φ is satisfiable, then so is φ_k (Definition 5.1), and thus $\Pi_\sigma \mathcal{A}_q(\varphi_k) = 0$ (Definition 5.3(a); also note that $\Pi_\sigma \mathcal{A}_q(\varphi_k)$ is constant). Therefore, in step 2 Prover either claims $\Pi_\sigma \mathcal{A}_q(\varphi_k) = 0$ and Verifier rejects, or the initial claim in step 3 is false.

If Verifier is never tricked, the claim remains false until step 5 is executed, at which point Verifier will reject. So to accept, Verifier must be tricked.

Part (b). Let $i \in \{k, ..., 1\}$ and assume that the claim is false at the beginning of iteration i of step 4. Now there are two cases. If Prover sends the polynomial $p = \Pi_{\sigma'} \mathcal{A}_q(\varphi_{i-1})$, then, as argued in the proof of Lemma 5.9, Verifier's check is equivalent to $w \stackrel{?}{=} \Pi_\sigma \mathcal{A}_q(\varphi_i)$, which is the current claim. However, we have assumed that the claim is false, so Verifier would reject. Hence, Prover must send a polynomial $p \neq \Pi_{\sigma'} \mathcal{A}_q(\varphi_{i-1})$ (of degree at most d) to trick Verifier.

By Lemma 2.2, the probability that Verifier selects an r with $\pi_{[x:=r]} p = \pi_{[x:=r]} \Pi_{\sigma'} \mathcal{A}_q(\varphi_{i-1})$ is at most d/q. Conversely, with probability at least $1 - d/q$, the new claim is false as well and Verifier is not tricked in this iteration.

Part (c). We have shown that the probability that Verifier is tricked in one iteration is at most d/q. By union bound, Verifier is thus tricked with probability at most dk/q, as there are k iterations. By conditions (a) and (b) of Theorem 5.6, we get $dk/q \in \mathcal{O}(\text{poly}(|\varphi|)/q)$. □

Running time of Verifier. The next lemma estimates Verifier's running time in terms of $|\varphi|$ and q.

Lemma 5.11. *Verifier runs in time* $\mathcal{O}(\text{poly}(|\varphi| \log q))$.

Proof. Verifier performs operations on polynomials of degree at most d with coefficients in $\mathbb{F}_q$. So a polynomial can be represented using $d \log q$ bits, and arithmetic operations are polynomial in that representation. Additionally, Verifier needs to execute P_{M_i} for each i, which can also be done in polynomial time (Definition 5.5(c)). There are $k \in \mathcal{O}(\text{poly}|\varphi|)$ iterations.

Finally, Verifier checks the claim $\Pi_\sigma \mathcal{A}_q(\varphi) = w$ for some assignment σ and $w \in \mathbb{F}_q$. Definition 5.3 ensures that this takes $\mathcal{O}(|\varphi| \,\text{polylog}\, q)$ time. The overall running time is therefore in $\mathcal{O}(\text{poly}(|\varphi| d \log q))$. The final result follows from condition (b) of Theorem 5.6. □

Running time of Prover. We give a bound on the running time of Prover, excluding the time needed to compute the prime q.

Lemma 5.12. *Assume that $\mathcal{A}$ is an arithmetisation satisfying the conditions of Theorem 5.6. Let T denote the time taken by Alg on φ. The running time of Prover, excluding the time needed to compute the prime q, is $\mathcal{O}(T \,\text{poly}|\varphi| \log q))$.*

Proof. After picking the prime q, Prover has to compute $\Pi_\sigma \mathcal{A}_q(\varphi_i)$ for different $i \in [k]$ and assignments σ. The conditions of Theorem 5.6 guarantee that this can be done in time $\mathcal{O}(|\varphi_i| \,\text{polylog}\, q) \subseteq \mathcal{O}(|\varphi_i| \,\text{poly}(|\varphi| \log q))$. We have $\sum_i |\varphi_i| \leq T$, as *Alg* needs to write each φ_i during its execution. The total running time follows by summing over i. □

Computing the prime q. The previous lemmas show the dependence of Verifier's probabitity of error and the running times of Prover and Verifier as a function of $|\varphi|$ and q. Our final lemma gives a procedure for Prover to compute a suitable prime q. Together with the previous lemmas, this will easily yield Theorem 5.6.

Lemma 5.13. *For every $c > 0$ there exists a randomised procedure for Prover to find a prime $q \in 2^{\mathcal{O}(|\varphi|)}$ such that $q \geq 2^{c|\varphi|}$ and $\mathcal{A}_q(\varphi_k) \neq 0$ in expected time $\mathcal{O}(T|\varphi|)$, where T is the running time of Alg.*

Proof. Assume wlog. that $c > 1$. Prover first runs *Alg* to compute φ_k and then chooses a prime q with $2^{c|\varphi|} \leq q < 2^{c|\varphi|+1}$ uniformly at random; thus $q \in 2^{\mathcal{O}(|\varphi|)}$ is guaranteed. If Prover arrives at $\mathcal{A}_q(\varphi_k) = 0$, Prover chooses another prime q in the same way, until one is chosen s.t. $\mathcal{A}_q(\varphi_k) \neq 0$.

Since $|\mathcal{A}(\varphi_k)| \leq 2^{2^{|\varphi|}}$, $\mathcal{A}(\varphi_k)$ is divisible by at most $2^{|\varphi|}$ different primes. Using the prime number theorem, there are $\Omega(2^{c|\varphi|}/c|\varphi|)$ primes $2^{c|\varphi|} \leq q < 2^{c|\varphi|+1}$, so the probability that the picked q divides $\mathcal{A}(\varphi_k)$ is $\mathcal{O}(c|\varphi|/2^{(c-1)|\varphi|})$.

Therefore, for any $c > 1$ this probability is at most, say, $1/2$ for sufficiently large $|\varphi|$. In expectation, Prover thus needs to test 2 primes q, and each test takes time $\mathcal{O}(|\varphi_k| \operatorname{polylog} q)$ (see Definition 5.3(b)), which is in $\mathcal{O}(T|\varphi|)$. □

Proof of Theorem 5.6. We can now conclude the proof of the theorem.

Completeness was already proved in Lemma 5.9.

Soundness. We need to ensure that the error probability is at most $2^{-|\varphi|}$. By Lemma 5.10, the probability p of error satisfies $p \leq dk/q$, where $dk \in \mathcal{O}(\text{poly}(|\varphi|))$. So there is a $\xi > 0$ with $dk \leq 2^{\xi|\varphi|}$. Using $c := 1 + \xi$ as constant for Lemma 5.13, we are done.

Verifier's running time. By Lemma 5.11, Verifier runs in time $\mathcal{O}(\text{poly}(|\varphi| \log q))$. Using the prime $q \in 2^{\mathcal{O}(|\varphi|)}$ of Lemma 5.13, the running time is $\mathcal{O}(\text{poly}(|\varphi|)$.

Competitivity. By Lemma 5.12, Prover runs in time $\mathcal{O}(T \operatorname{poly}(|\varphi| \log q))$ plus the time need to compute the prime, which, by Lemma 5.13, is in $\mathcal{O}(T \operatorname{poly}(|\varphi|))$. Again using $q \in \mathcal{O}(2^{|\varphi|})$, we find that the protocol is competitive with *Alg*.

6. An Interactive Proof System Competitive with the Davis-Putnam Resolution Procedure

In order to give an interactive proof system for the Davis-Putnam resolution procedure, it suffices to find an arithmetisation which is compatible with the full x-resolution step R_x and the full x-cleanup step C_x such that all properties of Theorem 5.6 are satisfied. In this section, we present such an arithmetisation.

6.1. **An arithmetisation compatible with R_x and C_x.** We find an arithmetisation compatible with both R_x and C_x. Let us first see that the arithmetisation of Example 5.4 does not work.

Example 6.1. The arithmetisation $\mathcal{A}$ of Example 5.3 is not compatible with R_x. To see this, let $\varphi = (\neg x \vee \neg y) \wedge (x \vee \neg z) \wedge \neg w$. We have $R_x(\varphi) = (\neg y \vee \neg z) \wedge \neg w$, $\mathcal{A}(R_x(\varphi)) = yz + w$, and $\mathcal{A}(\varphi) = xy + (1-x)z + w = x(y-z) + z + w$. If $\mathcal{A}$ were compatible with R_x, then there would exist an operation P_{R_x} on polynomials such that $P_{R_x}(x(y-z)+z+w) = yz + w$ by Definition 5.5(a), and from Definition 5.5(b), we get $P_{R_x}(\Pi_\sigma(x(y-z)+z+w)) = \Pi_\sigma(yz+w)$ for all partial assignments $\sigma : \{y, z, w\} \to \mathbb{Z}$. For $\sigma := \{y \mapsto 1, z \mapsto 0, w \mapsto 1\}$, it follows that $P_{R_x}(x+1) = 1$, but for $\sigma := \{y \mapsto 2, z \mapsto 1, w \mapsto 0\}$, it follows that $P_{R_x}(x+1) = 2$, a contradiction.

We thus present a non-standard arithmetisation.

Definition 6.2. The arithmetisation $\mathcal{B}$ of a CNF formula φ is the recursively defined polynomial

$$\begin{array}{lll} \mathcal{B}(\mathit{true}) := 0 & \mathcal{B}(x) := 1 - x & \mathcal{B}(\varphi_1 \wedge \varphi_2) := \mathcal{B}(\varphi_1) + \mathcal{B}(\varphi_2) \\ \mathcal{B}(\mathit{false}) := 1 & \mathcal{B}(\neg x) := x^3 & \mathcal{B}(\varphi_1 \vee \varphi_2) := \mathcal{B}(\varphi_1) \cdot \mathcal{B}(\varphi_2). \end{array}$$

Example 6.3. The right column of Table 1 shows the polynomials obtained by applying $\mathcal{B}$ to the formulas on the left. For example, we have $\mathcal{B}(\varphi_5) = \mathcal{B}(\neg z \wedge \neg z \wedge z) = 2\mathcal{B}(\neg z) + \mathcal{B}(z) = 2z^3 + 1 - z$.

We first prove that $\mathcal{B}$ is indeed an arithmetisation.

Proposition 6.4. *For every formula φ and every assignment $\sigma : X \to \{0,1\}$ to the variables X of φ, we have that σ satisfies φ iff $\Pi_\sigma \mathcal{B}(\varphi) = 0$.*

Proof. We prove the statement by induction on the structure of φ. The statement is trivially true for $\varphi \in \{\mathit{true}, \mathit{false}, x, \neg x\}$. For $\varphi = \varphi_1 \vee \varphi_2$, we have

$$\begin{aligned} &\sigma \text{ satisfies } \varphi \Leftrightarrow \sigma \text{ satisfies } \varphi_1 \vee \varphi_2 \Leftrightarrow \sigma \text{ satisfies } \varphi_1 \text{ or } \sigma \text{ satisfies } \varphi_2 \\ &\overset{IH}{\Leftrightarrow} \Pi_\sigma \mathcal{B}(\varphi_1) = 0 \vee \Pi_\sigma \mathcal{B}(\varphi_2) = 0 \Leftrightarrow \Pi_\sigma \mathcal{B}(\varphi_1) \cdot \Pi_\sigma \mathcal{B}(\varphi_2) = 0 \\ &\Leftrightarrow \Pi_\sigma \mathcal{B}(\varphi_1 \vee \varphi_2) = 0 \Leftrightarrow \Pi_\sigma \mathcal{B}(\varphi) = 0, \end{aligned}$$

and for $\varphi = \varphi_1 \wedge \varphi_2$, we have

$$\begin{aligned} &\sigma \text{ satisfies } \varphi \Leftrightarrow \sigma \text{ satisfies } \varphi_1 \wedge \varphi_2 \Leftrightarrow \sigma \text{ satisfies } \varphi_1 \text{ and } \sigma \text{ satisfies } \varphi_2 \\ &\overset{IH}{\Leftrightarrow} \Pi_\sigma \mathcal{B}(\varphi_1) = 0 \wedge \Pi_\sigma \mathcal{B}(\varphi_2) = 0 \Leftrightarrow \Pi_\sigma \mathcal{B}(\varphi_1) + \Pi_\sigma \mathcal{B}(\varphi_2) = 0 \\ &\Leftrightarrow \Pi_\sigma \mathcal{B}(\varphi_1 \wedge \varphi_2) = 0 \Leftrightarrow \Pi_\sigma \mathcal{B}(\varphi) = 0. \end{aligned}$$

The equivalence $\Pi_\sigma \mathcal{B}(\varphi_1) = 0 \wedge \Pi_\sigma \mathcal{B}(\varphi_2) = 0 \Leftrightarrow \Pi_\sigma \mathcal{B}(\varphi_1) + \Pi_\sigma \mathcal{B}(\varphi_2) = 0$ holds because $\Pi_\sigma \mathcal{B}(\varphi)$ cannot be negative for binary assignments σ. □

6.1.1. *$\mathcal{B}$ is compatible with R_x.* We exhibit a mapping $\gamma_x \colon \mathcal{P} \to \mathcal{P}$ satisfying the conditions of Definition 5.5 for the macrostep R_x. Recall that R_x is only defined for formulas φ in CNF such that $C(x) + C(\neg x) \leq 1$ for every clause C. Since arithmetisations of such formulas only have an x^3 term, an x term, and a constant term, it suffices to define γ_x for polynomials of the form $a_3x^3 + a_1x + a_0$.

Lemma 6.5. *Let $\gamma_x \colon \mathcal{P} \to \mathcal{P}$ be the partial mapping defined by $\gamma_x(a_3x^3 + a_1x + a_0) := -a_3a_1 + a_1 + a_0$. The mapping γ_x witnesses that $\mathcal{B}$ is polynomially compatible with the full resolution macrostep R_x.*

Proof. We show that γ_x satisfies all properties of Definition 5.5. Let φ be a formula in CNF such that $C(x) + C(\neg x) \leq 1$ for every clause C (see Definition 4.1). Then φ is of the form

$$\varphi = \Big(\bigwedge_{i \in [k]} x \vee a_i \Big) \wedge \Big(\bigwedge_{j \in [l]} \neg x \vee b_j \Big) \wedge c$$

where a_i, b_j are disjunctions not depending on x and c is a conjunction of clauses not depending on x. We have $R_x(\varphi) = \bigwedge_{i \in [k],\, j \in [l]} (a_i \vee b_j) \wedge c$. Now

$$\mathcal{B}(\varphi) = \sum_{i \in [k]} (1 - x)a_i + \sum_{j \in [l]} x^3 b_j + c = \sum_{j \in [l]} b_j x^3 - \sum_{i \in [k]} a_i x + \sum_{i \in [k]} a_i + c$$

and thus

$$\begin{aligned} \gamma_x(\mathcal{B}(\varphi)) &= \Big(\sum_{j \in [l]} b_j \Big) \Big(\sum_{i \in [k]} a_i \Big) - \sum_{i \in [k]} a_i + \sum_{i \in [k]} a_i + c \\ &= \sum_{i \in [k],\, j \in [l]} a_i b_j + c = \mathcal{B}(R_x(\varphi)). \end{aligned}$$

This proves (a). Since γ_x does not depend on variables other than x, (b) is also given. (c) and (d) are trivial. □

6.1.2. *$\mathcal{B}$ is compatible with C_x.* We exhibit a mapping $\delta_x \colon \mathcal{P} \to \mathcal{P}$ satisfying the conditions of Definition 5.5 for the cleanup macrostep C_x. Recall that C_x is only defined for formulas φ in CNF such that $C(x) + C(\neg x) \leq 2$ for every clause C. Arithmetisations of such formulas are polynomials of degree at most 6 in each variable, and so it suffices to define δ_x for these polynomials. Additionally, one can see by exhausting all possibilities that $\mathcal{B}$ never produces a x^5 term due to the restriction $C(x) + C(\neg x) \leq 2$. Hence, δ_x can ignore the a_5 coefficient.

Lemma 6.6. *Let $\delta_x \colon \mathcal{P} \to \mathcal{P}$ be the partial mapping defined by*

$$\delta_x(a_6x^6 + a_5x^5 + \cdots + a_1x + a_0) := (a_6 + a_4 + a_3)x^3 + (a_2 + a_1)x + a_0.$$

The mapping δ_x witnesses that $\mathcal{B}$ is polynomially compatible with C_x.

Proof. We show that δ_x satisfies all properties of Definition 5.5. We start with (a). Since $\mathcal{B}(C \wedge C') = \mathcal{B}(C) + \mathcal{B}(C')$ for clauses C, C' and $\delta_x(p_1 + p_2) = \delta_x(p_1) + \delta_x(p_2)$, it suffices to show that $\delta_x(\mathcal{B}(C)) = \mathcal{B}(C_x(C))$ for all clauses C of φ. Now let C be a clause of φ. We assume that $C(x) + C(\neg x) \leq 2$ (see Definition 4.1).

- If $C(x) + C(\neg x) \leq 1$, then $\delta_x(\mathcal{B}(C)) = \mathcal{B}(C) = \mathcal{B}(C_x(C))$.
- If $C = x \vee x \vee C'$, then $\mathcal{B}(C) = (1 - x)^2\mathcal{B}(C') = (1 - 2x + x^2)\mathcal{B}(C')$, so $\delta_x\mathcal{B}(C) = (1 - 2x + x)\mathcal{B}(C') = (1 - x)\mathcal{B}(C') = \mathcal{B}(x \vee C') = \mathcal{B}(C_x(C))$.
- If $C = \neg x \vee \neg x \vee C'$, then $\mathcal{B}(C) = x^6\mathcal{B}(C')$, so $\delta_x\mathcal{B}(C) = x^3\mathcal{B}(C') = \mathcal{B}(\neg x \vee C') = \mathcal{B}(C_x(C))$.

- If $C = x \vee \neg x \vee C'$, then $\mathcal{B}(C) = (1-x)x^3\mathcal{B}(C') = x^3\mathcal{B}(C') - x^4\mathcal{B}(C')$, so $\delta_x\mathcal{B}(C) = x^3\mathcal{B}(C') - x^3\mathcal{B}(C') = 0 = \mathcal{B}(C_x(C))$.

This proves (a). Since δ_x does not depend on variables other than x, (b) is also given. Parts (c) and (d) are trivial. □

As observed earlier, DavisPutnam does not remove duplicate clauses; that is, Prover maintains a multiset of clauses that may contain multiple copies of a clause. We show that the number of copies is at most double-exponential in $|\varphi|$.

Lemma 6.7. *Let φ be the input formula, and let φ_k be the last formula computed by* DavisPutnam. *Then $\mathcal{A}(\varphi_k) \in 2^{2^{\mathcal{O}(|\varphi|)}}$.*

Proof. Let $n_C(\psi)$ be the number of clauses in a formula ψ, let x be a variable. Then $n_C(C_x(\psi)) \leq n_C(\psi)$ because a cleanup step can only change or delete clauses. Moreover, $n_C(R_x(\psi)) = n_x n_{\neg x} - n_x - n_{\neg x} + n_C(\psi)$ where n_x and $n_{\neg x}$ are the numbers of clauses in ψ which contain x and $\neg x$, respectively. We get $n_C(R_x(\psi)) \leq (n_x + n_{\neg x})^2 - (n_x + n_{\neg x}) + n_C(\psi)$. Since $n_x + n_{\neg x} \leq n_C(\psi)$, it follows that $n_C(R_x(\psi)) \leq (n_C(\psi))^2$. Now let n be the number of variables. Since φ_k is reached after n resolution steps, it follows that $\mathcal{B}(\varphi_k) = n_C(\varphi_k) \leq n_C(\varphi)^{2^n} \in 2^{2^{\mathcal{O}(|\varphi|)}}$. □

Proposition 6.8. *There exists an interactive protocol for UNSAT that is competitive with* DavisPutnam.

Proof. We show that the $\mathcal{B}$ satisfies all properties of Theorem 5.6. On an input formula φ over n variables, DavisPutnam executes n resolution steps R_x and $n(n-1)/2$ cleanup steps C_x, which gives $n(n+1)/2$ macrosteps in total and proves (a).

Since φ does not contain any variable more than once per clause and since cleanup steps w.r.t. all remaining variables are applied after every resolution step, resolution steps can only increase the maximum degree of $\mathcal{B}(\varphi_i)$ to at most 6 (from 3). Hence the maximum degree of $\mathcal{B}(\varphi_i)$ is at most 6 for any i, showing (b).

Furthermore, since $R_x(\varphi_i)$ does not contain any occurrence of x, and resolution steps are performed w.r.t. all variables, φ_k does not contain any variables, so $\varphi_k = \{a \cdot \square\}$ for some $a \in \mathbb{N}$ where $\square$ is the empty clause. Together with Lemma 6.7, (c) follows. □

Instantiating Theorem 5.6 with $\mathcal{B}$ yields an interactive protocol competitive with DavisPutnam. Table 3 shows a run of this protocol on the formula φ of Table 1. Initially, Prover runs DavisPutnam on φ, computing the formulas $\varphi_1, ..., \varphi_6$. Then, during the run of the protocol, it sends to Verifier polynomials of the form $\Pi_{\sigma'}\mathcal{B}_q(\varphi_{i-1})$ for the assignments σ' chosen by Verifier.

7. Implementation and Evaluation

We have implemented DavisPutnam as `icdp` (Interactively Certified Davis-Putnam) [5], which executes both the Prover and Verifier sides of the interactive protocol. The tool computes a resolution proof of an unsatisfiable SAT instance, and interactively certifies that the proof is correct. The Prover is roughly 1000 lines of C++ code, the Verifier 100 lines, and in total the tool has about 2500 lines of code (all counts excluding blanks and comments). We use our implementation to empirically investigate three questions:

[5] https://gitlab.lrz.de/i7/icdp

Table 3: Run of the instance of the interactive protocol of Table 2 for DavisPutnam, using the arithmetisation $\mathcal{B}$ of Definition 6.2.

Round	Honest Prover	Verifier
Initial	$q := 15871$ $p_6 := \mathcal{B}_q(\varphi_6) = 2$ send q, p_6	$w := p_6 = 2$ $\sigma := \{x \mapsto 3, y \mapsto 4, z \mapsto 3\}$ send σ
$k = 6$	$\sigma' := \{x \mapsto 3, y \mapsto 4\}$ $p_5 := \Pi_{\sigma'}(\mathcal{B}_q(\varphi_5))$ $= 2z^3 - z + 1$ send p_5	$\pi_{[z:=3]}\gamma_z(p_5) = \pi_{[z:=3]}2 \stackrel{?}{=} 2$ $\sigma(z) := 4$ $w := \pi_{[z:=4]}p_5 = 125$ send σ
$k = 5$	$\sigma' := \{x \mapsto 3, y \mapsto 4\}$ $p_4 := \Pi_{\sigma'}(\mathcal{B}_q(\varphi_4))$ $= 2z^6 - 3z^4 + 3z^3 + z^2 - 2z + 1$ send p_4	$\pi_{[z:=4]}\delta_z(p_4) = \pi_{[z:=4]}2z^3 - z + 1 \stackrel{?}{=} 125$ $\sigma(z) := 2$ $w := \pi_{[z:=2]}p_4 = 105$ send σ
$k = 4$	$\sigma' := \{x \mapsto 3, z \mapsto 2\}$ $p_3 := \Pi_{\sigma'}(\mathcal{B}_q(\varphi_3))$ $= 15y^3 - 7y + 7$ send p_3	$\pi_{[y:=4]}\gamma_y(p_3) = \pi_{[y:=4]}105 \stackrel{?}{=} 105$ $\sigma(y) := 2$ $w := \pi_{[y:=2]}p_3 = 113$ send σ
$k = 3$	$\sigma' := \{x \mapsto 3, y \mapsto 2\}$ $p_2 := \Pi_{\sigma'}(\mathcal{B}_q(\varphi_2))$ $= 16z^6 - z^3 - 7z + 7$ send p_2	$\pi_{[z:=2]}\delta_z(p_2) = \pi_{[z:=2]}15z^3 - 7z + 7 \stackrel{?}{=} 113$ $\sigma(z) := 3$ $w := \pi_{[z:=3]}p_2 = 11623$ send σ
$k = 2$	$\sigma' := \{x \mapsto 3, z \mapsto 2\}$ $p_1 := \Pi_{\sigma'}(\mathcal{B}_q(\varphi_1))$ $= 729y^6 - 27y^4 + 754y^3 - 25y + 25$ send p_1	$\pi_{[y:=2]}\delta_y(p_1) = \pi_{[y:=2]}1456y^3 - 25y + 25 \stackrel{?}{=} 11623$ $\sigma(y) := 1$ $w := \pi_{[y:=1]}p_1 = 1456$ send σ
$k = 1$	$\sigma' := \{y \mapsto 1, z \mapsto 2\}$ $p_0 := \Pi_{\sigma'}(\mathcal{B}_q(\varphi_0))$ $= 54x^3 - 27x + 25$ send p_0	$\pi_{[x:=3]}\gamma_x(p_0) = \pi_{[x:=3]}1456 \stackrel{?}{=} 1456$ $\sigma(x) := 2$ $w := \pi_{[x:=2]}p_0 = 493$ send σ
Final		$\Pi_\sigma \mathcal{B}_q(\varphi) \stackrel{?}{=} 493$

- Interactive vs. conventional certification for DavisPutnam.
  We compare the resource consumption and communication complexity of (a) interactive certification as carried out by `icdp` and (b) conventional certification of DavisPutnam, in which Prover sends Verifier (an external routine) the resolution proof produced by DavisPutnam, and Verifier sequentially checks the correctness of each resolution step.
- `icdp` vs. general-purpose interactive certification.
  Proofs of the IP = PSPACE theorem provide interactive certification procedures for any PSPACE problem, and so in particular for UNSAT. This raises the question of how this general-purpose procedures compare to DavisPutnam.
- `icdp` vs. modern DRAT certification.
  We compare the resource consumption and communication complexity of (a) interactive certification as carried out by `icdp` and (b) conventional certification as carried out by `kissat`, a modern SAT solver that emits DRAT certificates of unsatisfiability [Heu16]. Like resolution proofs, DRAT certificates can be checked by a verifier in linear time and

Table 4: Families of formulae used in the benchmark set

Name	Count	Size of largest instance (in KiB)	Description
bphp	5	6.1	Binary pigeon-hole principle
cliquecoloring	4	2.5	Some graph is both k-colourable and has a $(k+1)$-clique
count	8	16.6	A set of elements can be partitioned into k subsets
domset	12	6.3	Existence of a dominating set
ec	3	5.0	The edges of a graph can be split into two parts s.t. each vertex has an equal number of incident edges in both
kclique	9	5.3	Existence of a clique
matching	11	6.4	Existence of a perfect matching
parity	4	8.9	A set can be grouped into pairs
php	5	3.2	Pigeon-hole principle
ram	2	6.5	Ramsey numbers
randkcnf	11	1.4	Random 3-CNF formulae close to the satisfiability threshold
rphp	4	3.7	Relativised pigeonhole principle
subsetcard	9	2.3	2-coloured edges in a bipartite graph, each colour has a majority in each vertex of the left and right side, respectively
tseitin	18	2.4	Tseitin formulae
vdw	3	2.3	Van der Waerden numbers

have exponential worst-case size in the size of the formula, but they are more compact in practice[6].

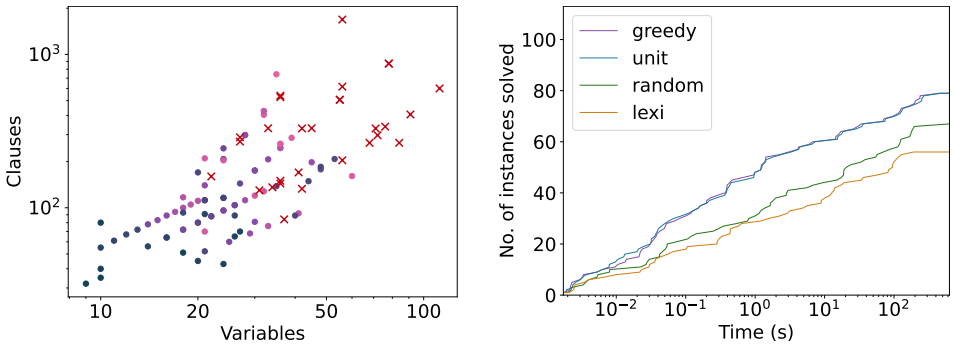
All experiments are carried out on an AMD Ryzen 9 5900X CPU with 64 GiB of total system memory under Linux, with a timeout of 20 minutes and a memory limit of 48 GiB.

7.1. **Benchmarks and variable order.** To obtain our benchmarks, we first use `cnfgen`[7] to generate 108 unsatisfiable instance formulas from well-known families known to be challenging for many SAT solvers, often encoding combinatorial problems. Table 4 gives an overview of the families used, and Figure 1(a) shows the number of variables and clauses of each instance. Dots represent instances solved by `icdp` within 20 minutes (see below for more details), and crosses represent instances for which `icdp` times out.

The runtime of `icdp` depends on the variable order. (Recall that Davis-Putnam performs one resolution phase for each variable and its performance depends on the order — although not its correctness.) We compare four different orders; `lexi`: the variables are ordered according to the provided instances; `random`: a variable order is chosen uniformly at random;

[6]The related LRAT format [CHJ+17] trades a slight increase in size for faster certification time and would be an interesting comparison as well. We leave this for future work, as DRAT is still the only supported format of almost all modern solvers.

[7]https://github.com/MassimoLauria/cnfgen



(a) (b)

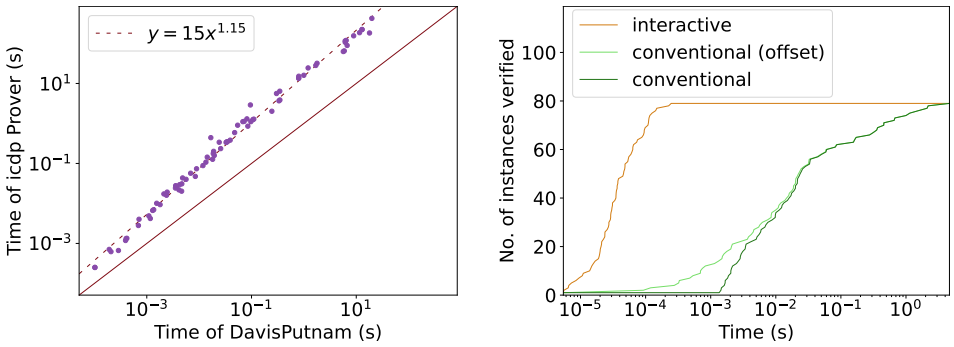
Figure 1: (a) Size of instances (number of variables and clauses). Instances solved by `icdp` within 20 minutes using the `greedy` variable order (see (b)) are shown as dots, the rest as crosses. (b) Number of instances solved by `icdp` with different variable orderings.

`greedy`: at each resolution step, choose the variable that minimises the size of the resulting formula; and `unit`: if a unit clause exists, pick its variable, and otherwise behave as `greedy`. The resuts are shown in Figure 1(b). The `greedy` order is slightly faster overall, and proves unsatisfiability of 79 of the 108 instances with a timeout of 20 minutes. These are the 79 instances represented as dots in Figure 1(a), and the ones we use as benchmark set for all subsequent experiments.

7.2. **Interactive vs. conventional certification for** DavisPutnam**.** We compare the time consumption, memory consumption and communication complexity of interactive certification with `icdp`, and conventional certification using the resolution proof of DavisPutnam as certificate. For time and memory consumption, we present results for Prover and Verifier separately.

Certification time. We first report on the runtime of Prover. For interactive certification, this is the total time `icdp`'s Prover needs to *both* solve the instance using DavisPutnam and interact with Verifier. For conventional certification, Prover's runtime is given by the runtime of DavisPutnam, because DavisPutnam already generates a resolution proof that can be used as certificate. The results are shown in Figure 2(a). Recall that, by the definition of a competitive protocol, for every formula φ the overall time needed by `icdp`'s Prover is $\mathcal{O}(\text{poly}(|\varphi|)T(|\varphi|))$, where $T(|\varphi|)$ is the runtime of DavisPutnam on φ. Figure 2(a) indicates that the polynomial $\text{poly}(|\varphi|)$ is almost constant.

Now we consider the runtime of Verifier. For interactive verification, this is the total runtime of `icdp`'s Verifier. This time is not contiguous, as there are multiple rounds of interaction in the protocol. Intuitively, we measure how much time is spent in *trusted code*, i.e. code on which it is sufficient to rely for the correctness of the result. For conventional certification, we let `icdp` output a resolution proof in the DRAT format used by modern SAT solvers, and define Verifier's runtime as the time taken by the `DRAT-trim` proof checker to check it [Heu16]. Due to the design of `DRAT-trim`, in its default configuration it spends roughly 40ms initialising internal data structures. To reduce this overhead, we adjusted the `BIGINIT` parameter to 10000 (from 1000000), after which the initialisation takes only 0.5ms.



(a) (b)

Figure 2: Interactive vs. conventional certification: Time consumption of (a) Prover and (b) Verifier, in seconds.

To better compare the scaling behaviour of the tools, we also include times for `DRAT-trim` that are offset by $T_1 - T_2$, where T_1, T_2 are the minimum times of `icdp` and `DRAT-trim` on any instance, respectively.

The results are shown in Figure 2(b). Since the time used by Verifier for each particular instance is very small, we present a cumulative plot indicating the number of verified instances for a given timeout. The total certification time for all 79 instances is about four orders of magnitude larger for conventional than for interactive certification.

Certification space. On Prover's side, we plot the memory used by `icdp`'s Prover to execute the complete protocol against the memory usage of DavisPutnam. The results are shown in Figure 3. The memory required to simply start `icdp` (without performing any computation) is 15 MiB, and for 27 out of our 79 instances these 15 MiB sufficed for both DavisPutnam and the complete protocol (orange dot). In the rest of the instances, the memory overhead of `icdp`'s Prover is quadratic. The reason is that, while `icdp`'s Prover computes the formulas $\varphi_1, ..., \varphi_k$ (see Table 2) in that order, in the course of certification it needs them in the reverse order, and so `icdp` stores the formulas $\varphi_1, ..., \varphi_k$. (For convenience they are stored in RAM, but they could also be stored on disk; one could also recompute them, trading time for space.) On the contrary, at any given time DavisPutnam only needs to store φ_i and φ_{i+1} for some $1 \leq i \leq k-1$.

We do not provide quantitative results on the memory usage of Verifier. For both conventional and interactive certification, Verifier needs virtually no memory.

Communication complexity. For interactive certification, we count the number of bytes sent by `icdp`'s Prover to Verifier. For conventional certification, we count the number of bytes of the resolution proof produced by DavisPutnam. The results are shown in Figure 4. We observe that the space used by interactive certification seems to grow like the inverse fifth power of the size of the resolution proof. If this power law extends to larger proofs, then in order to interactively certify a petabyte proof Prover only needs to send Verifier a few kilobytes.

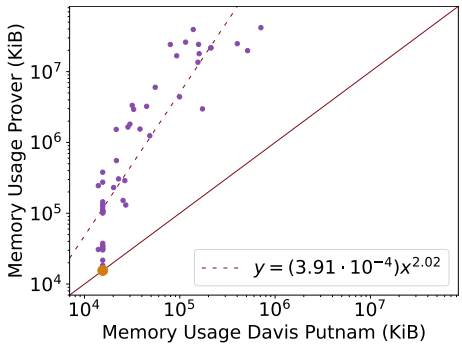


Figure 3: Interactive vs. conventional certification: Memory usage of Prover, in kilobytes.

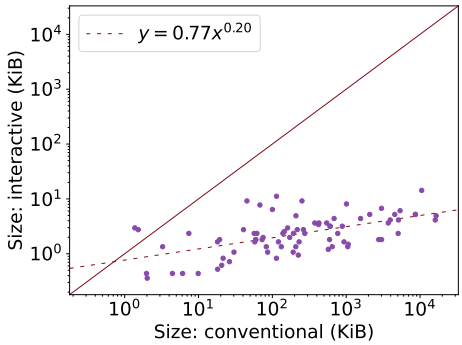


Figure 4: Interactive vs. conventional certification: Communication complexity, in kilobytes.

7.3. `icdp` **vs. general-purpose interactive certification.** Proofs of the IP = PSPACE theorem provide interactive certification procedures for any PSPACE problem, and so in particular for UNSAT. This raises the question of how such general-purpose procedures compare in practice with interactive certification using `icdp`. The answer can be derived from Figure 1(a). As we mentioned in the introduction, the interactive protocols for UNSAT given by the proofs of IP = PSPACE (or IP $\supseteq$ coNP) essentially require that Prover evaluates the formula for all possible truth assignments. Therefore, with a 20-minute timeout for the Prover, and generously assuming that it can check 10^7 assignments per second, general-purpose Provers can only certify formulas with at most 33 variables, while `icdp`'s Prover goes beyond 50.

7.4. `icdp` **vs. modern DRAT certification.** Modern SAT solvers emit DRAT certificates of unsatisfiability [Heu16]. Like resolution proofs, DRAT certificates can be checked by a verifier in linear time and have exponential worst-case size in the size of the formula, but they are more compact in practice[8].

[8]The related LRAT format [CHJ+17] trades a slight increase in size for faster certification time and would be an interesting comparison as well. We leave this for future work, as DRAT is still the only supported format of almost all modern solvers.

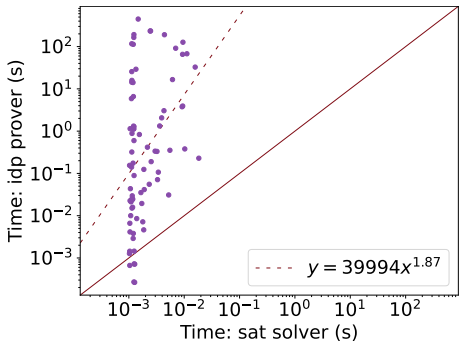


Figure 5: Interactive certification with `icdp` vs. conventional certification with `kissat` and `DRAT-trim`: Solving time, in seconds.

We compare the resource consumption of interactive certification with `icdp` and conventional certification with `kissat`[9], a state-of-the-art SAT solver. For the latter, we run `kissat` to generate DRAT certificates, and use `DRAT-trim` to check their correctness.

Certification time. On Prover's side, we plot the runtime of `icdp`'s Prover against the runtime of `kissat`. The results are shown in Figure 5; `kissat` is about five orders of magnitude faster than `icdp`. While a good part of `kissat`'s advantage may be due to preprocessing and better data structures, we think that the main issue is the low performance of DavisPutnam itself. As we mentioned in the introduction, DavisPutnam was proposed in the 1960s and is not a state-of-the-art procedure.

Let us now consider the time consumption of Verifier. A caveat is that, since `icdp` can only certify formulas it has solved, we can only perform a comparison on formulas that are very small for the standards of modern SAT solvers. We compare the time that `DRAT-trim`—a standard tool for checking DRAT certificates—needs to check the DRAT certificates produced by `kissat` with the runtime of `icdp`'s Verifier. Time is measured internally, using the same high-resolution clock. Figure 6 shows a survival plot of the number of instances checked by Verifier within a given time. As described above, for `DRAT-trim` we also provide times that are offset. Overall, `icdp`'s Verifier checks roughly one order of magnitude faster than `DRAT-trim`, even when the times are offset. For example, the longest time of `DRAT-trim` is 347ms, while `icdp`'s Verifier takes at most 0.2ms on the benchmarked instances[10]. This result is likely due to the theoretical exponential speedup of Verifier compared to conventional certification.

Certification space. We omit the experimental results, because they do not add insight: On Prover's side, we have the same situation as for time consumption: `icdp`'s Prover consumes much more memory than `kissat`. On Verifier's side, the memory consumption of both `icdp`'s Verifier and `DRAT-trim` are negligible.

[9] https://github.com/arminbiere/kissat

[10] Recall that certification times may be faster for the LRAT format, and so the advantage of interactive certification with respect to time might disappear in a comparison with LRAT certification.

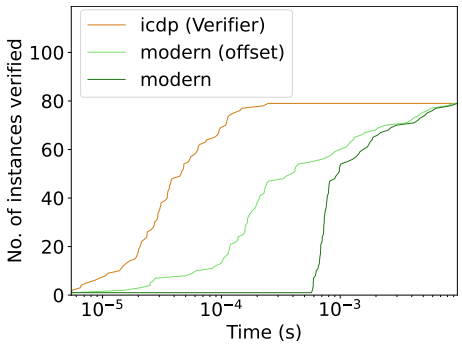


Figure 6: Interactive certification with `icdp` vs. conventional certification with `kissat` and `DRAT-trim`: Verifier's time, in seconds.

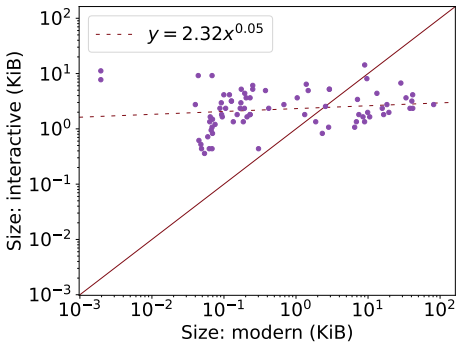


Figure 7: Interactive certification with `icdp` vs. conventional certification with `kissat` and `DRAT-trim`: Communication complexity, in kilobytes.

Communication complexity. We compare the length in bytes of the DRAT certificate produced by `kissat` and the number of bytes sent by `icdp`'s Prover to Verifier during the execution of `icdp`. The results are shown in Figure 7. While DRAT certificates are smaller for many instances, interactive certification scales better, and outperforms conventional certification in the larger instances. In particular, the size of the interactive certificates grows even more slowly in the size of the DRAT certificate than in the size of the resolution proof[11]. Again, this result is due to the theoretical complexity differences between interactive proofs with guaranteed polynomial time for Verifier and conventional certification with exponential-size certificates in the worst case.

7.5. **Final summary.** If we consider DavisPutnam in a distributed setting, where Prover and Verifier run in different machines, interactive certification exhibits clear advantages w.r.t. Verifier's time consumption and communication complexity: both improve by several orders of magnitude with respect to conventional certification. The price to pay on Prover's side is an almost linear time overhead and a quadratic memory overhead.

Our experiments on interactive certification for DavisPutnam vs. conventional certification in `kissat` indicate that, while the advantages in Verifier's time consumption and

[11]DRAT certificates are more compact than LRAT ones, and so the advantage of interactive certification with respect to certificate size should also manifest itself in an experiment with LRAT certificates.

communication complexity still remain, the price to pay in Prover's runtime is very high, due to the much higher solving time.

8. Conclusions

We have presented the first interactive proof system for the Davis-Putnam resolution procedure, one of the oldest algorithms for UNSAT producing a certificate. We have derived it in a systematic way using a procedure to obtain interactive proof systems from arithmetisations satisfying a few commutativity properties. We have shown that, while standard arithmetisations do not satisfy these properties, non-standard arithmetisations do.

Our work shows that interactive certification with small Prover overhead exist not only for brute-force algorithms, but for more sophisticated ones. At the same time, Davis-Putnam is several orders of magnitude slower than current techniques, like CDCL. Whether the results we have obtained for Davis-Putnam can be extended to these techniques remains open.

Lovasz et al. have shown that given a refutation by the Davis-Putnam resolution procedure, one can extract a multi-valued decision diagram, polynomial in the size of the refutation, in which the path for a given truth assignment leads to a clause false under that assignment (that is, to a clause witnessing that the assignment does not satisfy the formula) [LNNW95]. This suggests a possible connection between our work and the work of Couillard et al. in [CCEM23]. As mentioned in the introduction, [CCEM23] presents an interactive proof system competitive with the algorithm for UNSAT that iteratively constructs a BDD for the formula (starting at the leaves of its syntax tree, and moving up at each step), and returns "unsatisfiable" iff the BDD for the root of the tree only contains the node 0. We conjecture that a future version of our systematic derivation technique could subsume both [CCEM23] and this paper.

Acknowledgments. We thank the anonymous reviewers for their comments and Albert Atserias for helpful discussions.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[Bab85] 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. doi:10.1145/22145.22192.

[BN21] Sam Buss and Jakob Nordström. Proof complexity and SAT solving. 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 233–350. IOS Press, 2021. doi:10.3233/FAIA200990.

[BRK+22] 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. doi:10.1007/978-3-031-10769-6_3.

[BT88] Samuel R. Buss and György Turán. Resolution proofs of generalized pigeonhole principles. *Theor. Comput. Sci.*, 62(3):311–317, 1988. doi:10.1016/0304-3975(88)90072-2.

[CCEM23] Eszter Couillard, Philipp Czerner, Javier Esparza, and Rupak Majumdar. Making IP = PSPACE practical: Efficient interactive protocols for BDD algorithms. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III*, volume 13966 of *Lecture Notes in Computer Science*, pages 437–458. Springer, 2023. doi:10.1007/978-3-031-37709-9_21.

[CEK24] Philipp Czerner, Javier Esparza, and Valentin Krasotin. A resolution-based interactive proof system for UNSAT. In Naoki Kobayashi and James Worrell, editors, *Foundations of Software Science and Computation Structures - 27th International Conference, FoSSaCS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part II*, volume 14575 of *Lecture Notes in Computer Science*, pages 116–136. Springer, 2024. doi:10.1007/978-3-031-57231-9_6.

[CHJ+17] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. 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 220–236. Springer, 2017. doi:10.1007/978-3-319-63046-5_14.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960. doi:10.1145/321033.321034.

[GMR85] 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. doi:10.1145/22145.22178.

[Hak85] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

[Har09] John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

[Heu16] Marijn J. H. Heule. The DRAT format and drat-trim checker. *CoRR*, abs/1610.06229, 2016.

[Heu21] 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. doi:10.3233/FAIA200998.

[HJKW17] Marijn Heule, Warren A. Hunt Jr., Matt Kaufmann, and Nathan Wetzler. Efficient, verified checking of propositional proofs. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings*, volume 10499 of *Lecture Notes in Computer Science*, pages 269–284. Springer, 2017. doi:10.1007/978-3-319-66107-0_18.

[HJM+02] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, George C. Necula, Grégoire Sutre, and Westley Weimer. Temporal-safety proofs for systems code. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 526–538. Springer, 2002. doi:10.1007/3-540-45657-0_45.

[HKM16] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. *CoRR*, abs/1605.00723, 2016.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. doi:10.1145/146585.146605.

[LNNW95] László Lovász, Moni Naor, Ilan Newman, and Avi Wigderson. Search problems in the decision tree model. *SIAM J. Discret. Math.*, 8(1):119–132, 1995. doi:10.1137/S0895480192233867.

[Nam01] Kedar S. Namjoshi. Certifying model checkers. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *Lecture Notes in Computer Science*, pages 2–13. Springer, 2001. doi:10.1007/3-540-44585-4_2.

[Nec97] George C. Necula. Proof-carrying code. In Peter Lee, Fritz Henglein, and Neil D. Jones, editors, *Conference Record of POPL'97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles*

of Programming Languages, Papers Presented at the Symposium, Paris, France, 15-17 January 1997, pages 106–119. ACM Press, 1997. doi:10.1145/263699.263712.

[Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992. doi:10.1145/146585.146609.