The Proof Complexity of SMT Solvers

Robert Robere[1(✉)], Antonina Kolokolova[2], and Vijay Ganesh[3]

[1] University of Toronto, Toronto, Canada
robere@cs.toronto.edu
[2] Memorial University of Newfoundland, St. John's, Canada
kol@mun.ca
[3] University of Waterloo, Waterloo, Canada
vijay.ganesh@uwaterloo.ca

Abstract. The resolution proof system has been enormously helpful in deepening our understanding of conflict-driven clause-learning (CDCL) SAT solvers. In the interest of providing a similar proof complexity-theoretic analysis of satisfiability modulo theories (SMT) solvers, we introduce a generalization of resolution called Res(T). We show that many of the known results comparing resolution and CDCL solvers lift to the SMT setting, such as the result of Pipatsrisawat and Darwiche showing that CDCL solvers with "perfect" non-deterministic branching and an asserting clause-learning scheme can polynomially simulate general resolution. We also describe a stronger version of Res(T), Res*(T), capturing SMT solvers allowing introduction of new literals. We analyze the theory EUF of equality with uninterpreted functions, and show that the Res*(EUF) system is able to simulate an earlier calculus introduced by Bjørner and de Moura for the purpose of analyzing DPLL(EUF). Further, we show that Res*(EUF) (and thus SMT algorithms with clause learning over EUF, new literal introduction rules and perfect branching) can simulate the Frege proof system, which is well-known to be far more powerful than resolution. Finally, we prove under the Exponential Time Hypothesis (ETH) that *any* reduction from EUF to SAT (such as the Ackermann reduction) must, in the worst case, produce an instance of size $\Omega(n \log n)$ from an instance of size n.

1 Introduction

It is common practice in formal verification literature to view SAT/SMT solver algorithms as proof systems and study their properties, such as soundness, completeness and termination, using proof-theoretic tools [GHN+04, ORC09, Tin12]. However, much work remains in applying the powerful lens of proof complexity theory in understanding the relative power of these solvers. All too often, the power of SAT and SMT (satisfiability modulo theories) solving algorithms is determined by how they perform at the annual SAT or SMTCOMP competitions [BHJ17, smt]. While such competitions are an extremely useful practical test of the power of solving methods, they do not address fundamental questions

H. Chockler and G. Weissenbacher (Eds.): CAV 2018, LNCS 10982, pp. 275–293, 2018.
https://doi.org/10.1007/978-3-319-96142-2_18

such as which heuristics are truly responsible for the power of these solvers or what are the lower bounds for these methods when viewed as proof systems.

Solvers, by their very nature, are a tangled jumble of heuristics that interact with each other in complicated ways. Many SMT solvers run into hundreds of thousands of lines of code, making them very hard to analyze. It is often difficult to discern which sets of heuristics are universally useful, which sets are tailored to a class of instances, and how their interactions actually help solver performance. A purely empirical approach, while necessary, is far from sufficient in deepening our understanding of solver algorithms. What is needed is an appropriate combination of empirical and theoretical approaches to understanding the power of solvers. Fortunately, proof complexity theory provides a powerful lens through which to mathematically analyze solver algorithms as proof systems and to understand their relative power via lower bounds. The value of using proof complexity theory to better understand solving algorithms as proof systems is three-fold: first, it allows us to identify key ingredients of a solving algorithm and prove lower bounds to non-deterministic combinations of such ingredients. That is, we can analyze the countably many variants of a solving algorithm in a unified manner via a single analysis, rather than analyzing different configurations of the same set of proof-theoretic ingredients; second, proof complexity-theoretic tools allow us to recognize the relative power of two proof systems, via appropriate lower bounds, even if both have worst-case exponential time complexity; finally, proof complexity theory brings with it a rich literature and connections to other sub-fields of complexity theory (e.g., circuit complexity) that we may be able to leverage in analyzing solver algorithms. Many proof complexity theorists and logicians have long recognized this, and there is rich literature on the analysis of SAT solving algorithms such as DPLL and conflict-driven clause-learning (CDCL) solvers [PD11,BKS04,BBJ14,AFT11]. In this paper, we lift some of these results to the setting of SMT solvers, following the work of Bjørner and de Moura [BM14].

Our focus is primarily the proof complexity-theoretic analysis of the "DPLL(T) method"[1], the prime engine behind many modern SMT solvers [GHN+04,Tin12]. (While other approaches to solving first-order formulas have been studied, DPLL(T) remains a fundamental and dominant approach.) A DPLL(T)-based SMT solver takes as input a Boolean combination of first-order theory T atoms or their negation (aka, theory literals), and decides whether such an input is satisfiable. Informally, a typical DPLL(T)-based SMT solver S

[1] Prior to mid 2000's, SAT researchers and complexity theorists confusingly used the term DPLL to refer to both the original algorithm proposed by Davis, Putnam, Loveland, and Loeggemann in 1960, as well as the newer algorithm by Joao Marques-Silva and Karem Sakallah that added clause learning to DPLL (proposed in 1996), even though they are vastly different in power as proof systems. We will follow the literature and use DPLL(T) to indicate a "modern" SMT solver with clause learning and restarts, but, we urge SMT solver researchers to use the more appropriate term CDCL(T) rather than DPLL(T) to refer to the lazy approach to SMT.

is essentially a CDCL Boolean SAT solver that calls out a theory solver T_s during its search to perform *theory propagations* and *theory conflict-clause learning*. The typical theory solver T_s is designed to accept only quantifier-free conjunction of theory T literals (the T in the term DPLL(T)), while the SAT solver "handles" the Boolean structure of input formulas. Roughly speaking, the SMT solver S works as follows: First, it constructs a Boolean abstraction B_F of the input formula F, by replacing theory literals by Boolean variables. If B_F is UNSAT, S returns UNSAT. Otherwise, satisfying assignments to the Boolean abstraction B_F are found, which in turn correspond to conjunctions of theory literals. Such conjunctions are then input to the theory solver T_s, which may deduce new implied formulas (via theory propagation and conflict clause learning) that are then used to help prune the search space of assignments to F. The solver S returns SAT upon finding a satisfying theory assignment to the input F, and UNSAT otherwise. (For further details, we refer the reader to the excellent exposition on this topic by Tinelli [Tin12].)

A Brief Description of the $\mathsf{Res}(T)$ Proof System: To abstractly model a DPLL(T)-based SMT solver S, we define a proof system $\mathsf{Res}(T)$ below for a given first-order theory T. The Res in $\mathsf{Res}(T)$ refers to the general resolution proof system for Boolean logic. Without loss of generality, we assume that $\mathsf{Res}(T)$ accepts theory formulas in conjunctive normal form (CNF). Let $\mathcal{F}$ denote a CNF with propositional variables representing atoms from an underlying theory T, and for any clause C in FF let $\mathsf{vars}(F)$ denote the set of propositional atoms occurring in F. The proof rules of $\mathsf{Res}(T)$ augment the resolution proof rule as follows: A proof in $\mathsf{Res}(T)$ is a general resolution refutation of $\mathcal{F}$, where at any step the theory T-solver can add to the set of clauses an arbitrary clause C such that $T \models C$ and every propositional atom in $\mathsf{vars}(C)$ occurs in the original formula. That is, each line of a $\mathsf{Res}(T)$ proof is deduced by one of the following rules:

Resolution. $C \vee \ell, D \vee \overline{\ell} \vdash C \vee D$, for previously derived clauses C and D.
Theory Derivation. $\vdash C$ for any clause C such that $T \models C$ and for which every theory literal in C occurs in the input formula.

For example, a theory of linear arithmetic may introduce a clause $(x \geq 5 \vee y \geq 7 \vee x + y < 12)$, which can then be used in the subsequent steps of a resolution proof, provided each of those literals occurred in the original CNF formula $\mathcal{F}$. The filter on the theory rule of $\mathsf{Res}(T)$ models the fact that in many modern SMT solvers, the "theory solver" is only allowed to reason about literals which already occur in the formula. Recent solvers such as Z3, Yices [Z3, Yic] break this rule and allow the theory solver to introduce new propositional atoms; to model this we introduce the stronger variant $\mathsf{Res}^*(T)$ with a strengthened theory rule:

Strong Theory Derivation: $\vdash C$ for any clause C such that $T \models C$.

1.1 Our Contributions

We prove the following results about the two systems $\mathsf{Res}(T)$, $\mathsf{Res}^*(T)$ and the complexity of SMT solving.

1. We show that $\mathsf{DPLL}(T)$ with an arbitrary asserting clause learning scheme and non-deterministic branching and theory propagation is equivalent (as a proof system) to $\mathsf{Res}(T)$ for *any* theory T. More precisely: if the theory solver in $\mathsf{DPLL}(T)$ can only reason about literals in the input, then it is equivalent to $\mathsf{Res}(T)$; if it can reason about arbitrary literals then it is equivalent to $\mathsf{Res}^*(T)$. (See Sect. 3)
2. When the theory T is E, the theory of pure equalities, $\mathsf{Res}^*(\mathrm{E})$ is equivalent to the $SP(\mathrm{E})$ system of Bjørner et al. [BDdM08], which seems to have no efficient proofs of the PHP. (See Sect. 5.1)
3. When the theory T is EUF (equality with uninterpreted function symbols), the proof system $\mathsf{Res}^*(\mathrm{EUF})$ can simulate $\mathsf{E\text{-}Res}$, a different generalization of resolution introduced by Bjorner and de Moura [BM14] for the purpose of simulating standard implementations of $\mathsf{DPLL}(\mathrm{EUF})$. Furthermore, $\mathsf{Res}^*(\mathrm{EUF})$ can simulate the powerful Frege proof system. (See Sect. 5.2)
4. When T is LA, a theory of linear arithmetic over a set of numbers containing integers, $\mathsf{Res}(\mathrm{LA})$ can polynomially simulate the system $\mathsf{R(lin)}$ of Raz and Tzameret [RT08], and thus has polynomial size proofs of several hard tautologies such as the pigeonhole principle and Tseitin tautologies. (See Sect. 5.3)
5. Finally, we prove under the Exponential Time Hypothesis (ETH) that *any* reduction from EUF to SAT (such as the Ackermann reduction) must, in the worst case, produce an instance of size $\Omega(n \log n)$ from an instance of size n. (See Sect. 6)

These results seem to suggest that our generalization is the "right" proof system corresponding to $\mathsf{DPLL}(T)$, as it characterizes proofs produced by $\mathsf{DPLL}(T)$ and it can simulate other proof systems introduced in the literature to capture $\mathsf{DPLL}(T)$ for particular theories T.

1.2 Previous Work

Among the previous proof systems combining resolution with non-propositional reasoning are $\mathsf{R(CP)}$ proof system of [Kra98], where propositional variables are replaced with linear inequalities, and $\mathsf{R(lin)}$ introduced by Raz and Tzameret [RT08], which reasons with linear equalities, modifying the resolution rule. $\mathsf{R(lin)}$ polynomially simulates $\mathsf{R(CP)}$ when all coefficients in an $\mathsf{R(CP)}$ proof are polynomially bounded. In the SMT community, Bjørner et al. [BDdM08,BM14] introduced calculi capturing the power of resolution over the theory of equality and equality with uninterpreted functions. They show that these systems capture the power of resolution over the corresponding theories, extended with rules for introducing new atoms. Our results supersede previous work since our simulations hold for any first-order theory T.

2 Preliminaries

2.1 Propositional Proof Systems

In this paper, all proof systems are defined by a set of "allowed lines" equipped with a list of deduction rules that allow us to deduce new lines from old ones. We first recall the *resolution* system, which is a refutation system for propositional formulas in CNF (product of sums) form. The lines of a resolution proof are disjunctions of boolean literals called *clauses*, and these lines are equipped with a single deduction rule called the *resolution rule*: given two clauses of the form $C \vee \ell$, $D \vee \overline{\ell}$ we deduce the clause $C \vee D$. If $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ is an unsatisfiable CNF formula then a resolution refutation of ϕ is a sequence of clauses $C_1, C_2, \ldots, C_m, C_{m+1}, \ldots, C_t$ where C_t is the empty clause and all clauses C_i with $i > m$ are deduced from earlier clauses by applying the resolution rule.

Observe that clauses satisfy a *subsumption principle*: if C, D are clauses such that $C \subseteq D$ then every assignment satisfying C also satisfies D. This implies that we can safely add a *weakening rule* to resolution which, from a clause C, derives the clause $C \vee x$ for any literal x not already occurring in C. The subsumption principle implies that this weakening rule does not change the power of resolution, as any use of a clause $D \supseteq C$ can be eliminated or replaced with C.

We also consider the *Frege* proof system, which captures standard "textbook-style" proofs. The lines of a Frege system are given by arbitrary boolean formulas, and from two boolean formulas we can deduce any new boolean formula which follows under typical boolean reasoning (e.g. deducing the conjunction of two formulas, the disjunction of their negation, and so on). Crucially, Frege proofs allow applying a generalized "resolution rule" to arbitrary polynomial-size formulas.

The power of different propositional proof systems are compared using the notion of an *polynomial simulation (p-simulation)*. Proof system A *polynomially simulates* (or p-simulates) proof system B if, for every unsatisfiable formula $\mathcal{F}$, the shortest refutation proof of $\mathcal{F}$ in A is at most polynomially longer than the shortest refutation proof of a formula $\mathcal{F}$ in B. For example, the Frege proof system p-simulates the Resolution proof system, but the converse is widely conjectured not to hold.

2.2 First-Order Theories

In this paper we study proof systems for first-order theories. For the sake of completeness we recall some relevant definitions from first-order logic, but remark that this is essentially standard fare.

Let $\mathcal{L}$ be a first-order signature (a list of constant symbols, function symbols, and predicate symbols). Given a set of $\mathcal{L}$-sentences $\mathcal{A}$ and an $\mathcal{L}$-sentence B we write $\mathcal{A} \models B$ if every model of $\mathcal{A}$ is also a model of B. A *first order theory* (or simply a *theory*) is a set of $\mathcal{L}$-sentences that is consistent (that is, it has a model) and is closed under $\models$. The *decision problem* for a theory T is the following: given a set S of literals over $\mathcal{L}$, decide if there is a model $\mathcal{M}$ of T such that $\mathcal{M} \models S$.

The *satisfiability problem* for T, also denoted T-SAT, is the following: given a quantifier-free formula $\mathcal{F}$ in T in conjunctive normal form (CNF), decide if there is a model $\mathcal{M}$ of T such that $\mathcal{M} \models F$.

A simple example of a theory is E, the conjunctive theory of equality. The signature of E contains a single predicate symbol $=$ and an infinite list of constant symbols. It is axiomatized by the standard axioms of equality (reflexivity, symmetry, and transitivity), and a sample sentence in E would be the formula $a \neq b \vee b \neq c \vee a = c$, which encodes the transitivity of equality between the constant symbols a, b, and c. Following the SMT literature, we will call terms from the theory (such as a and b) *theory variables*, and the atoms derived from these terms (such as $a \neq b$ or $a = c$) will be called *theory literals* or just *literals*. We note that the decision problem for E can be decided very efficiently [DST80]; in contrast, the satisfiability problem for E is easily seen to be NP-complete.

3 Res(T): Resolution Modulo Theories

We now define a generalization of resolution which captures the type of reasoning modulo a first-order theory that is common in SMT solvers. We give two variants: the first, denoted $\mathsf{Res}(T)$, allows the deduction of any clause C of theory literals such that $T \models C$ and for which every literal in C already occurs in the input formula. This is intended to model "standard" lazy SMT solvers [NOT06] which only reason about literals in the input formula.

The second, more powerful variant is denoted $\mathsf{Res}^*(T)$, and allows the deduction of any clause of literals C such that $T \models C$, *even if* the new clause contains literals which do not occur in the input formula. We introduce this to explore the power of lazy SMT solvers that are allowed to introduce new literals from the theory, and note that there are well-known examples in the SMT literature which show that introducing new literals can drastically decrease the length of refutations (e.g. the *diamond equalities* [BDdM08]). Indeed, in Sect. 5.2 we show that this power can drastically increase the proof theoretic strength of SMT solvers.

Definition 1 ($\mathsf{Res}(T), \mathsf{Res}^*(T)$)**.** *Let T be a theory and let $\mathcal{F}$ be an quantifier-free CNF formula over T. The lines of a $\mathsf{Res}(T)$ ($\mathsf{Res}^*(T)$) proof are quantifier-free clauses of theory literals deduced from $\mathcal{F}$ and T by the following derivation rules.*

Resolution. $C \vee \ell, D \vee \overline{\ell} \vdash C \vee D$.

Weakening. $C \vdash C \vee \ell$ *for any theory literal ℓ occurring in the input formula.*

Theory Derivation ($\mathsf{Res}(T)$)**.** $\vdash C$ *for any clause C satisfying $T \models C$ and for which every literal in C occurs in the input formula.*

Strong Theory Derivation ($\mathsf{Res}^*(T)$)**.** $\vdash C$ *for any clause C satisfying $T \models C$.*

A refutation *of $\mathcal{F}$ is a proof in which the final line is the empty clause.*

It is easy to see that both $\mathsf{Res}(T)$ and $\mathsf{Res}^*(T)$ are sound since all rules are sound, and completeness follows from a straightforward modification of the usual proof of resolution completeness (see, e.g. Jukna [Juk12]).

Technically speaking, $\mathsf{Res}(T)$ is *not* a (formal) propositional proof system as defined by Cook and Reckhow [CR79] since the proofs may not be efficiently verifiable if deductions from the theory T are computationally difficult to verify. However, all theories considered in this paper (cf. Sect. 5) are very efficiently decidable, and thus the corresponding $\mathsf{Res}(T)$ proofs are efficiently verifiable.

Note that the clauses introduced by the theory derivations are arbitrary theorems of T; this means there is no direct information exchange between the resolution proof and the theory. It is enough to derive clauses in the theory derivation rules rather than arbitrary formulas since every axiom can be written in CNF form, and introduced as a sequence of clauses. The strong theory derivation rule can introduce new theory literals which might not have been present in the initial formula—we emphasize that the new theory literals can even contain theory *variables* (i.e. first-order terms) that did not occur in the original formula. We will see that this ability to introduce new literals seems to give $\mathsf{Res}^*(T)$ extra power over general resolution.

4 Lazy SMT Solvers and $\mathsf{Res}(T)$

In this section we show that lazy SMT solvers and resolution modulo theories are polynomially-equivalent as proof systems, provided that the SMT solvers are given a set of branching and restart decisions *a priori*.

We model SMT solvers by the algorithm schema[2] $\mathsf{DPLL}(T)$, which is given in Algorithm 1. Using this schema we prove two results: first, if the theory solver in $\mathsf{DPLL}(T)$ can only reason about literals occurring in its input formula, then $\mathsf{DPLL}(T)$ is polynomially equivalent to $\mathsf{Res}(T)$. Second, if the theory solver is strengthened so that it is allowed to introduce new literals then the resulting solver can polynomially simulate $\mathsf{Res}^*(T)$. The proofs of these results use techniques developed for comparing Boolean CDCL solvers and resolution by Pipatsrisawat and Darwiche [PD11].

[2] In the literature, SMT solvers are typically defined as abstract state-transition systems (see, for instance, [GHN+04, BM14]); we have chosen to define it instead as an algorithm schema (cf. Algorithm 1) inspired by the abstract definition of a CDCL solver by Pipatsrisawat and Darwiche [PD11].

Algorithm 1. DPLL(T)

Input: CNF formula $\mathcal{F}$ over T-literals;
Output: SAT or UNSAT
Let $\sigma = \emptyset$ be an initially empty partial assignment of T-literals;
Let Γ be an initially empty collection of learned clauses;
while *true* **do**
  if $\mathcal{F} \wedge \Gamma \wedge \sigma \vdash_1 \emptyset$ **then**
    if $\sigma = \emptyset$ **then**
      return UNSAT;
    Apply the **clause learning scheme** to learn a conflict clause C, add it to Γ;
    Backjump σ to the second highest decision level in C;
  else if $\sigma \vDash^T \emptyset$ **then**
    Apply the T**-conflict scheme** to learn a conflict clause C, add it to Γ;
    Backjump σ to the second highest decision level in C;
  else
    if σ *satisfies* $\mathcal{F}$ **then**
      return SAT;
    Apply the **restart scheme** to decide whether or not to restart;
    if *restart* **then**
      Set $\sigma = \emptyset$;
      Restart loop;
    Apply the T**-propagate scheme**;
    Unit propagate literals to completion and update σ accordingly;
    Apply the **branching scheme** to choose a decision literal ℓ, set $\sigma = \sigma \cup \{\ell\}$;

If T is a theory and A, B are formulas over T then we write $A \vDash^T B$ as a shorthand for $T \cup \{A\} \vDash B$ (i.e. every model of the theory T that satisfies A also satisfies B). We also define *unit resolution*, which describes the action of the *unit propagator*.

Definition 2 (Unit Resolution). *Let $\mathcal{F}$ be a collection of clauses over an arbitrary theory T. A clause C is derivable from $\mathcal{F}$ by* unit resolution *if there exists a resolution proof from $\mathcal{F}$ of C such that in each application of the resolution rule, one of the clauses is a unit clause. If C is derivable from $\mathcal{F}$ by unit resolution then we write $\mathcal{F} \vdash_1 C$. If $\mathcal{F} \vdash_1 \emptyset$ then we say $\mathcal{F}$ is* unit refutable, *otherwise it is* unit consistent.

A DPLL(T) algorithm is defined by specifying algorithms for each of the bolded "schemes" in Algorithm 1:

Clause Learning Scheme. When a clause in the database is falsified by the current partial assignment, the **Clause Learning Scheme** is applied to learn a new clause C which is added to the database of stored clauses.

Restart Scheme. The solver applies the **Restart Scheme** to decide whether or not to restart its search, discarding the current partial assignment σ and saving the list of learned clauses.

Branching Scheme. The **Branching Scheme** is applied to choose an unassigned variable from the formula $\mathcal{F}$ or from the learned clauses Γ and assign the variable a Boolean value.

T-Propagate Scheme. During search, the $\mathsf{DPLL}(T)$ solver can hand the theory solver the current partial assignment σ and ask whether or not it should unit-propagate a literal; if a unit propagation is possible the theory solver will return a clause C from the theory witnessing this unit propagation.

T-Conflict Scheme. When the theory solver detects that the current partial assignment σ contradicts the theory, the **T-Conflict Scheme** is applied to learn a new clause of literals C, $\neg C \subseteq \sigma$, which is added to the clause database.

We pay particular interest to the specification of the T-propagate scheme. The next definition describes two types of propagation schemes: a *weak* propagation scheme is only allowed to return clauses which propagate literals in the formula, while the more powerful *strong* propagation scheme returns a clause of literals from the theory that may contain new literals.

Definition 3. *A* weak T-propagate scheme *is an algorithm which takes as input a conjunction of theory literals σ over T and returns (if possible) a clause $C = \neg\sigma \vee \ell$ where $T \models C$ and the literal ℓ occurs in the input formula of the* $\mathsf{DPLL}(T)$ *algorithm.*

A strong T-propagate scheme *is an algorithm which takes as input a conjunction of literals σ over T, and if possible returns a clause C of literals from T such that $T \models C$ and $\neg\sigma \subseteq C$. An algorithm equipped with a strong T-propagate scheme will be called a* $\mathsf{DPLL}^*(T)$ *solver.*

A $\mathsf{DPLL}(T)$ algorithm equipped with a weak T-propagation scheme is equivalent to the basic theory propagation rules found in SMT solvers (see, for example, [BM14, NOT06]). For technical convenience we assume that the weak T-propagate scheme adds a clause to the database "certifying" the unit propagation, while in actual implementations the clause would likely not be added and the literal would simply be propagated. Recent SMT solvers [Yic, Z3] have strengthened the interaction between the SAT solver and the theory solver, allowing the theory solver to return constraints over new variables; this is modelled very generally by strong T-propagate schemes.

4.1 $\mathsf{DPLL}(T)$ and $\mathsf{Res}(T)$

We now prove the main result of this section, after introducing some preliminaries from [PD11] that are suitably modified for our setting. Fix a theory T. An *assignment trail* is a sequence of pairs $\sigma = \{(\ell_i, d_i)\}_{i=1}^{t}$ where each literal ℓ_i is a literal from the theory and each $d_i \in \{\mathsf{d}, \mathsf{p}\}$, indicating that the literal was set by a decision or a unit propagation. The *decision level* of a literal ℓ_i in σ

is the number of decision literals occurring in σ up to and including ℓ_i. Given an assignment trail σ and a clause C we say that C is *asserting* if it contains exactly one literal occurring in σ at the highest decision level. A clause learning scheme is *asserting* if all conflict clauses produced by the scheme are asserting with respect to the assignment trail at the time of conflict.

An *extended branching sequence* is an ordered sequence $B = \{\beta_1, \beta_2, \ldots, \beta_t\}$ where each β_i is either (1) a literal from the theory, (2) a symbol $x \in \{\mathsf{R}, \mathsf{NR}\}$, to denote a restart or no-restart, respectively, or (3) a clause C such that $T \vDash C$. Intuitively, extended branching sequences are used to provide a $\mathsf{DPLL}(T)$ solver with a list of instructions for how to proceed in its execution. For instance, whenever the solver calls the Branching Scheme, we consume the next β_i from the sequence, and if it is a literal from the theory then the solver assigns that literal. Similarly, when the $\mathsf{DPLL}(T)$ solver calls the Restart Scheme it uses the branching sequence to dictate whether or not to restart, and when the solver calls the T-propagate scheme it uses the sequence to dictate which clause to learn. If the symbol does not correctly match the current scheme being called then the solver halts in error, and if the branching sequence is empty, then the algorithm proceeds using the heuristics defined by the algorithm.

We now introduce *absorbed* clauses (and their duals, *empowering* clauses), which were originally defined by Pipatsrisawat and Darwiche [PD11] and independently by Atserias et al. [AFT11]. One should think of the absorbed clauses as being learned "implicitly"—they may not necessarily appear in $\mathcal{F}$, but, if we assign all but one of the literals in the clause to false then unit propagation in $\mathsf{DPLL}(T)$ will set the final literal to true.

Definition 4 (Empowering Clauses). *Let $\mathcal{F}$ be a collection of clauses over an arbitrary theory T and let A be a $\mathsf{DPLL}(T)$ solver. Let α be a conjunction of literals, and let $C = (\neg\alpha \Rightarrow \ell)$ be a clause. We say that C is* empowering with respect to $\mathcal{F}$ at ℓ *if the following holds: (1) $\mathcal{F} \cup T \vDash C$, (2) $\mathcal{F} \wedge \alpha$ is unit consistent, and (3) any execution of A on $\mathcal{F}$ that satisfies α without setting ℓ does not unit-propagate ℓ. The literal ℓ is said to be* empowering. *If item (1), (2) are satisfied but (3) is false then we say that the solver A and $\mathcal{F}$* absorbs *C at ℓ; if A and $\mathcal{F}$ absorbs C at at every literal then the clause is simply* absorbed.

For an example, consider the set of clauses $(x \vee y \vee z), (\neg z \vee a), (\neg a \vee b)$. The clause $(x \vee y \vee b)$ is absorbed by this set of clauses as, for instance, if we falsify x and y then the unit-propagator will force b to be set to true. Thus in the $\mathsf{DPLL}(T)$ algorithm the unit propagator will behave as though this clause is learned even though it is not (if we remove the final clause $\neg a \vee b$, then $(x \vee y \vee b)$ is empowering but not absorbed).

The next lemma shows that for any theory clause C, there is an extended branching sequence which can be applied to absorb that clause.

Lemma 5. *Let $\mathcal{F}$ be an unsatisfiable CNF over a theory T and let Π be any $\mathsf{Res}(T)$ proof from $\mathcal{F}$. Let $\Pi_T \subseteq \Pi$ be the set of clauses in Π derived using the theory rule. For any $\mathsf{DPLL}(T)$ algorithm A there is an extended branching sequence B such that after applying B to the solver A every clause in Π_T will be absorbed.*

Proof. Order Π_T arbitrarily as $C_1, C_2, \ldots, C_t$ and remove any clause that is absorbed or already in $\mathcal{F}$, as these are clearly already absorbed. We construct B directly: add the negations of literals in C_1 to B until one literal remains, and then add the clause C_1 to the extended branching sequence. By definition the weak T-propagator will be called and will return C_1, adding it to the clause database. Restart and continue to the next theory clause in order.

Our proof of mutual simulations between $\mathsf{Res}(T)$ and $\mathsf{DPLL}(T)$ crucially relies on the following technical lemma (which is a modified version of a lemma from [PD11]).

Lemma 6. *Let $\mathcal{F}$ be an unsatisfiable, unit-consistent CNF over literals from a theory T and let Π be any $\mathsf{Res}(T)$ proof from $\mathcal{F}$. Let Π_T be the set of clauses in Π derived using the theory rule. Then there exists a clause C in Π that is both empowering and unit-refutable with respect to $\mathcal{F} \cup \Pi_T$.*

Proof. Let Π denote a $\mathsf{Res}(T)$-refutation of $\mathcal{F}$ and assume without loss of generality (by applying Lemma 5) that the first derived clauses in Π are in Π_T. If every clause in Π is unit-refutable from $\mathcal{F}$, then the empty clause is unit-refutable and thus $\mathcal{F}$ is not unit-consistent, which is a contradiction. So, assume that there exists a clause C_i which is the first clause in Π by this ordering such that it is not unit-refutable. Since Π is a $\mathsf{Res}(T)$-proof, C_i is one of three types: either it is a clause in $\mathcal{F}$, it is a clause derived from the theory rule, or C_i was derived by applying the resolution rule to two clauses C_j, C_k. If $C_i \in \mathcal{F}$ then it is clearly unit-refutable, which is a contradiction. If C_i was derived from the theory rule then it is unit-refutable with respect to Π_T, which is again a contradiction. Finally, suppose that C_i was derived by applying the resolution rule to clauses C_j and C_k, and write $C_j = (\alpha \Rightarrow \ell)$, $C_k = (\beta \Rightarrow \overline{\ell})$ where ℓ is the resolved literal and $j, k < i$ in the ordering of clauses in Π. Since C_j and C_k are both unit-refutable, assume by way of contradiction that neither C_j nor C_k are empowering. It follows by definition that both clauses are absorbed at every literal. Thus, if we consider $\mathcal{F} \wedge \alpha \wedge \beta$, it follows by the absorption property that $\mathcal{F} \wedge \alpha \wedge \beta \vdash_1 \ell, \mathcal{F} \wedge \alpha \wedge \beta \vdash_1 \neg\ell$ which implies that $\mathcal{F} \wedge \alpha \wedge \beta \vdash_1^T \emptyset$. However, $\overline{C}_i = \alpha \wedge \beta$, and thus we have concluded C_i is unit-refutable, which is a contradiction! Thus at least one of C_j or C_k is both empowering and unit-refutable.

The gist of the Lemma 6 is simple: if clauses $C \vee \ell$ and $D \vee \overline{\ell}$ are both absorbed by a collection of clauses $\mathcal{C}$, then asserting $\overline{C} \wedge \overline{D}$ in the DPLL solver will hit a conflict since it will unit-imply both ℓ and $\overline{\ell}$. In the main theorem, proved next, we show that empowering and unit-refutable clauses will be absorbed by the solver after sufficiently many restarts.

Theorem 7. *The $\mathsf{DPLL}(T)$ system with an asserting clause learning scheme, non-deterministic branching and T-propagation polynomially simulates $\mathsf{Res}(T)$. Equivalently: for any unsatisfiable CNF $\mathcal{F}$ over a theory T, and any $\mathsf{Res}(T)$ refutation Π of $\mathcal{F}$ there exists an extended branching sequence B such that running a $\mathsf{DPLL}(T)$ algorithm on input $\mathcal{F}$ using B will refute $\mathcal{F}$ in time polynomial in the length of $|\Pi|$.*

Proof. Let $\mathcal{F}$ be an unsatisfiable CNF over the theory T, and let Π be a $\mathsf{Res}(T)$ refutation of $\mathcal{F}$. Let $\Pi_T \subseteq \Pi$ be the set of clauses in Π derived using the theory rule, and write $\Pi = C_1, C_2, \ldots, C_m$. As a first step, apply Lemma 5 and construct an extended branching sequence B' which leads to the absorbtion of all clauses in Π_T. We prove the following claim, from which the theorem directly follows.

Claim. Let C be any unit-refutable and empowering clause with respect to $\mathcal{F}$. Then there exists an extended branching sequence B of polynomial size such that after applying B the clause C will be absorbed.

Let ℓ be any empowering literal of C, and write $C = (\alpha \Rightarrow \ell)$. Let B be any extended branching sequence in which all literals in α are assigned. Since C is empowering, it follows that $\mathcal{F} \wedge \alpha$ is unit-consistent. Extending B with the decision literal $\neg\ell$ will therefore cause a conflict since C is unit-refutable. Let C' be the asserting clause obtained by applying the clause learning scheme to $B \cup \{\neg\ell\}$. If $\mathcal{F} \wedge C'$ absorbs C at ℓ, then we are done and we continue to the next empowering literal. Otherwise, we resolve whatever conflicts the solver needs to resolve (possibly adding more learned clauses along the way) until the branching sequence is unit-consistent.

Observe that after this process we must have that $\mathcal{F} \wedge C' \vdash_1 \ell'$ where ℓ' is some literal at the same decision level as ℓ, since the clause learning scheme is asserting. Thus the number of literals at the maximum decision level has reduced by one. At this point, we restart and do exactly the same sequence of branchings—each time, as argued above, we reduce the number of literals at the maximum decision level by 1. Since ℓ is a literal at the maximum decision level, it implies that after at most $O(n)$ restarts (and $O(n^2)$ learned clauses) we will have absorbed the clause C at ℓ. Repeating this process at most n times for each empowering literal in C we can absorb C, and it is clear that the number of learned clauses is polynomial from the analysis.

We are now ready to finish the proof. Apply the claim repeatedly to the first empowering and unit-refutable clause in Π to absorb that clause—by Lemma 6, such a clause will exist as long as the CNF $\mathcal{F}$ is not unit-refutable; a $\mathsf{DPLL}(T)$ solver can obtain an arbitrary theory clause by setting relevant literals in the branching sequence and using theory propagation. Since the length of the proof Π is finite (length m), it follows that this process must terminate after at most m iterations. At this point, there can not be such an empowering and unit-refutable clause, and so by Lemma 6 it follows that $\mathcal{F}$ (with its learned clauses) is now unit-refutable, and so the $\mathsf{DPLL}(T)$ algorithm halts and outputs UNSAT.

The reverse direction of the theorem is straightforward, and thus we have the following corollary:

Corollary 8. *The* $\mathsf{DPLL}(T)$ *system with an asserting clause learning scheme, non-deterministic branching and* T*-propagation is polynomially equivalent to* $\mathsf{Res}(T)$*.*

A key point of the above simulation is that it does not depend on whether or not the T-propagation scheme is weak or strong—since the clauses learned

by the scheme are specified in advance by the extended branching sequence the same proof will apply if we began with a $\mathsf{Res}^*(T)$ proof instead. Of course, if we begin with a $\mathsf{Res}^*(T)$ proof instead of a $\mathsf{Res}(T)$ proof we may use the full power of the theory derivation rule, requiring that we use a $\mathsf{DPLL}^*(T)$ algorithm with a strong T-propagation scheme instead. We record this observation as a second theorem.

Theorem 9. *The* $\mathsf{DPLL}^*(T)$ *system with an asserting clause learning scheme, non-deterministic branching and* T*-propagation is polynomially equivalent to* $\mathsf{Res}^*(T)$.

5 Case Studies: Resolution Modulo Common Theories

In this section, we study the power of $\mathsf{Res}(T)$ over theories that are common in the SMT context—namely, we focus on the theory of equality E, the theory of uninterpreted function symbols EUF, and the theory of linear arithmetic LA.

5.1 Resolution over E: A Theory of Equality

We first consider E, the theory of equality. Bjørner et al. [BDdM08] introduced a proof-theoretic calculus called $\mathtt{SP}(\mathrm{E})$ for reasoning over the theory of equality—in a prototype of our main result, they showed that proofs in $\mathtt{SP}(\mathrm{E})$ exactly characterized proofs produced by a simple model SMT solver. In this section we show that the theory $\mathsf{Res}^*(\mathrm{E})$ is polynomially-equivalent to $\mathtt{SP}(\mathrm{E})$, which is evidence that our general framework is the correct way of capturing the power of SMT solvers.

Let us first reproduce the rules of $\mathtt{SP}(\mathrm{E})$ from [BDdM08]: **Cut.** $C \vee \ell$, $D \vee \neg\ell \vdash C \vee D$, **E-Dis.** $C \vee a \neq a \vdash C$, **E-Eqs.** $C \vee a = b \vee a = c \vdash C \vee a = b \vee b \neq c$, **Sup.** $C \vee a = b$, $D[a] \vdash C \vee D[b]$. Observe that the Sup rule allows replacing some occurrences of a term a in atoms of a clause D with b (not necessarily for all occurrences of a). Both the Sup rule and the E-Eqs rule can introduce literals that did not occur in the initial formula.

Proposition 10. $\mathsf{Res}^*(\mathrm{E})$ *and* $\mathtt{SP}(\mathrm{E})$ *are polynomially equivalent.*

Proof (Sketch). Bjørner et al. show that $\mathtt{SP}(\mathrm{E})$ exactly characterizes the proofs produced by a simple theoretical model of an SMT solver, which we will denote by $\mathsf{DPLL}(e + \Delta)$ [BDdM08, Theorem 4.1]. Examining the solver $\mathsf{DPLL}(e + \Delta)$ from [BDdM08], it is not hard to see it is equivalent to the algorithm $\mathsf{DPLL}^*(\mathrm{E})$ (that is, $\mathsf{DPLL}(T)$ with a strong T-propagation rule). The equivalence between $\mathsf{Res}^*(\mathrm{E})$ and $\mathsf{DPLL}^*(\mathrm{E})$ follows by the Corollary of Theorem 9.

In the conclusion of [BDdM08] it is stated that there are no short $\mathtt{SP}(\mathrm{E})$ proofs of the following encoding of the pigeonhole principle (PHP): there are clauses of the form $(d_i = r_1 \vee \ldots d_i = r_n)$, for $i \in [1, n+1]$, enforcing that the ith pigeon must travel to some hole, and clauses of the form $(d_i \neq d_j)$ for

$i, j \in [1, n+1]$ which, when combined with the first family of clauses and the transitivity axioms of E, imply that no two pigeons can travel to the same hole. Since their $\mathtt{SP}(\mathrm{E})$ system is equivalent to $\mathsf{Res}^*(\mathrm{E})$ it follows that the lower bounds on $\mathtt{SP}(\mathtt{E})$ carry over:

Corollary 11. *If* $\mathtt{SP}(\mathrm{E})$ *does not have polynomial-size refutations of the pigeonhole principle, then neither does* $\mathsf{Res}^*(\mathrm{E})$.

5.2 Resolution over EUF: Equality with Uninterpreted Functions

Next, we study the theory EUF, which is an extension of the theory of equality to contain uninterpreted function symbols. The signature of EUF consists of an unlimited set of uninterpreted function symbols and constant symbols; a term in the theory is thus inductively defined as either a constant symbol or an application of a function symbol to a sequence of terms: $f(t_1, \ldots, t_k)$. There is one relational symbol $=$ interpreted as equality between terms, so theory literals of EUF are of the form $t = t'$ for terms t, t'.

The axioms of EUF state that $=$ is an equivalence relation, together with a family of *congruence axioms* for the function symbols stating, for any k-ary function symbol f and any sequences of terms $t_1, t_2, \ldots, t_k$, $t'_1, t'_2, \ldots, t'_k$, if $t_1 = t'_1, \ldots, t_k = t'_k$, then $f(t_1, \ldots, t_k) = f(t'_1, \ldots, t'_k)$. The decision problem for EUF can be decided in time $O(n \log n)$ by the Downey-Sethi-Tarjan congruence closure algorithm [DST80].

Using EUF as a central example, Bjorner and de Moura [BM14] observed that $\mathsf{DPLL}(T)$ suffers some serious limitations in terms of access to the underlying theory. To resolve this, they modified $\mathsf{DPLL}(\mathrm{EUF})$ with a set of non-deterministic rules that allowed it to dynamically introduce clauses corresponding to the congruence and transitivity axioms. To characterize the strength of this new algorithm, they introduced a variant of resolution called $\mathsf{E\text{-}Res}$, extending $\mathtt{SP}(\mathrm{E})$ from [BDdM08] to reasoning over uninterpreted functions. We show that the $\mathsf{Res}^*(\mathrm{EUF})$ proof system can polynomially-simulate the $\mathsf{E\text{-}Res}$ system, which again suggests that we have the "correct" proof system for capturing SMT reasoning. Due to space considerations, we leave the proof to the full version of the paper.

Theorem 12. *The system* $\mathsf{E\text{-}Res}$ *is polynomially simulated by* $\mathsf{Res}^*(\mathrm{EUF})$.

However, unlike the case of $\mathtt{SP}(\mathrm{E})$ the converse direction is not so clear. The theory rule in $\mathsf{Res}^*(\mathrm{EUF})$ is fundamentally *semantic*: it allows one to derive *any* clause which follows from the theory EUF semantically; this is in contrast to the $\mathsf{E\text{-}Res}$ system which is fundamentally syntactic. Thus, to show that $\mathsf{E\text{-}Res}$ polynomially simulates EUF, one would need to show that any use of the theory rule in a $\mathsf{Res}^*(\mathrm{EUF})$ proof could be somehow replaced with a short proof in $\mathsf{E\text{-}Res}$. We leave this as an open problem.

Next, we show that $\mathsf{Res}^*(\mathrm{EUF})$ and $\mathsf{E\text{-}Res}$ can efficiently simulate the Frege proof system, which is a very powerful propositional proof system studied in proof complexity. We note that the simulation crucially relies on the introduction

of new theory literals; this suggests that an SMT solver which can intelligently introduce new theory literals has the potential to be extremely powerful.

Theorem 13. Res*(EUF) *(and, in fact,* E-Res*) can efficiently simulate the Frege proof system.*

Proof Sketch. We show the stronger statement that E-Res simulates Frege. The idea of the proof is to introduce constants $e_0 \neq e_1$ corresponding to FALSE and TRUE; every positive literal x in the original formula is replaced by $x = e_1$, and negative literal $\neg x$ by $x = e_0$. Then introduce uninterpreted function symbols N, O, A, together with constraints that make N, O, A behave as NOT, OR and AND, respectively (such as $N(e_0) = e_1 \wedge N(e_1) = e_0$). So formulas in the Frege refutation are iteratively transformed into expressions of the form $t_F = e_0$ or $t_F = e_1$, where t_F is a term obtained by replacing Boolean connectives in a formula F by N, O, A. As the Frege proof ends with an empty sequent, the corresponding E-Res proof ends with an empty clause. See the full version for details.

5.3 Resolution over LA: A Theory of Linear Arithmetic

Finally, we study the theory of linear arithmetic LA. A formula in the theory LA over a domain D is a conjunction of expressions of the form $\Sigma_{i=1}^{n} a_i x_i \circ b$, where $\circ \in \{=, \leq, <, \neq, \geq, >\}$, and $a_i, x_i \in D$ — usually, D is integers or reals[3]. We show that Res(LA) polynomially simulates the proof system R(lin) introduced by Raz and Tzameret [RT08]. This is interesting, as R(lin) has polynomial-size proofs of several difficult tautologies considered in proof complexity, such as the pigeonhole principle, Tseitin tautologies and the clique-colouring principle.

In the proof system R(lin) propositional variables are linear equations over integers. The input formula is a CNF over such equations, together with $\bigwedge_{i=1}^{n}(x_i = 0 \vee x_i = 1)$ clauses ensuring 0/1 assignment. The rules of inference consist of a modified resolution rule, together with two structural rules, weakening and simplification:

R(lin)**-cut.** Let $(A \vee L_1)$, $(B \vee L_2)$ be two clauses containing linear equalities L_1 and L_2, respectively. From these two clauses, derive a clause $(A \vee B \vee (L_1 - L_2))$.
Weakening. From a (possibly empty) clause A derive $(A \vee L)$ for any equation L.
Simplification. From $(A \vee k = 0)$, where $k \neq 0$ is a constant, derive A.

Proposition 14. Res(LA) *polynomially simulates* R(lin).

[3] Some definitions of linear arithmetic do not include disequalities; however, as disequalities and strict inequalities occur naturally in SMT context, SMT-oriented linear arithmetic solvers do incorporate mechanisms for dealing with them.

Proof. We show how to simulate rules of R(lin) in Res(LA). We can assume, without loss of generality, that Res(LA) has a weakening rule which simulates weakening of R(lin) directly. For the simplification rule, note that LA $\vDash k \neq 0$ for any $k \neq 0$; one application of the resolution rule on $(k \neq 0)$ and $(A \vee k = 0)$ results in A.

Finally, let L_1 be $\Sigma_{i=1}^{n} a_i x_i = b$ and L_2 be $\Sigma_{i=1}^{n} c_i x_i = d$. From $(A \vee L_1)$, $(B \vee L_2)$ we want to derive $(A \vee B \vee L_1 - L2)$. First derive in LA a clause $C = (\Sigma_{i=1}^{n} a_i x_i \neq b \vee \Sigma_{i=1}^{n} c_i x_i \neq d \vee \Sigma_{i=1}^{n} (a_i - c_i) x_i = b - d)$. Resolving $(A \vee L_1)$ with C, and then resolving the resulting clause with $(B \vee L_2)$ gives the desired $(A \vee B \vee (L_1 - L_2))$.

Note that we didn't need to specify whether LA is over the integers, rationals or reals, and hence the proof works for any of them. Also, in order to establish our simulations it is sufficient to consider a fragment of LA with only equalities and inequalities, and produce only unit clauses and width-3 clauses of a fixed form.

Corollary 15. Res(LA) *has polynomial-size proofs of the pigeonhole principle, Tseitin tautologies and a clique-colouring principle for* $k = \sqrt{(n)}$ *size clique and* $k' = (\log n)^2 / 8 \log \log n$ *size colouring.*

6 Lazy vs. Eager Reductions and the Exponential Time Hypothesis

Throughout this paper we have primarily discussed the *Lazy* approach to SMT. In this section, we consider the *Eager* approach, in which an input formula $\mathcal{F}$ over a theory T is reduced to an equisatisfiable propositional formula $\mathcal{G}$, which is then solved using a suitable (Boolean) solver.

The Eager approach is still used in several modern SMT solvers such as the STP solver for bit-vectors and arrays [GD07]. A common eager reduction used when solving equations over the theory of equality, E (or its generalization to uninterpreted function symbols EUF), is the *Ackermann reduction*. Let us first describe a simple version of the Ackermann reduction over the theory E.

Let $\mathcal{F}$ denote a CNF over literals from the theory E—so, each literal is of the form $a = b$ for constant terms a, b—which we will ultimately transform into a Boolean SAT instance. Let n denote the number of constant terms occurring in $\mathcal{F}$, let m denote the number of distinct literals occurring in $\mathcal{F}$, and consider the literal $a = b$ and the literal $b = a$ to be the same. For each literal $a = b$ introduce a Boolean variable $x_{a=b}$, and for each clause of literals $\bigvee_i a_i = b_i$ create a clause $\bigvee_i x_{a_i = b_i}$. To encode the transitivity of equality, for each triple of terms (a, b, c) occurring in the initial CNF $\mathcal{F}$ introduce a clause of the form $\neg x_{a=b} \vee \neg x_{b=c} \vee x_{a=c}$. Note that the final formula will have $O(n^2)$ Boolean variables corresponding to each possible term $a = b$—a potential quadratic blow-up—which is unavoidable using this encoding due to the transitivity axioms. Observe that this blow-up only occurs in the eager approach—in the lazy approach to solving we only need to consider the literals $a = b$ which occur in the original

formula $\mathcal{F}$. It is therefore natural to wonder if this blow-up in the number of input variables can somehow be avoided.

In fact, one can construct a more clever Eager reduction from E-SAT to SAT which only introduces $O(n \log n)$ boolean variables; however, this more clever encoding does not represent the literals $a = b$ as Boolean variables $x_{a=b}$ and instead uses a more complicated pointer construction. This improved reduction turns out to be the best possible under the well-known (and widely believed) *Exponential Time Hypothesis*, which is a strengthening of $\mathtt{P} \neq \mathtt{NP}$.

Exponential Time Hypothesis (ETH). There is no deterministic or randomized algorithm for SAT running in time $2^{o(n)}$, where n is the number of input variables.

Theorem 16. *Let $\mathcal{F}$ be an instance of* E*-SAT with n distinct terms. For any polynomial-time reduction R from* E*-SAT to SAT, the boolean formula $R(\mathcal{F})$ must have $\Omega(n \log n)$ variables unless ETH fails.*

Proof. By way of contradiction, suppose that ETH holds and let R be a reduction from E-SAT to SAT which introduces $o(n \log n)$ variables. Let 2-CSP denote a constraint satisfaction problem with two variables per constraint. The theorem follows almost immediately from the following result of Traxler [Tra08].

Theorem 17 (Theorem 1 in [Tra08], Rephrased). *Consider any 2-CSP $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ over an alphabet Σ of size d, where each constraint is of the form $x \neq a \vee y \neq b$ for variables x, y and constants $a, b \in \Sigma$. Unless ETH fails, every algorithm for this problem requires time d^{cn} for some universal constant $c > 0$.*

There is a simple reduction from the restriction of 2-CSP described in the above theorem to E-SAT. Introduce terms $e_1, e_2, \ldots, e_d$, each intended to represent a symbol from the universe Σ, and also terms $x_1, x_2, \ldots, x_n$ for each variable x occurring in the original CSP instance. Now, for each $i \neq j$ introduce unit clauses $e_i \neq e_j$, and similarly for each $i \in [n]$ add a clause of the form $x_i = e_1 \vee x_i = e_2 \vee \cdots \vee x_i = e_d$. Finally, for each constraint in the 2-CSP of the form $x_i \neq a \vee x_j \neq b$ introduce a clause $x_i \neq e_a \vee x_j \neq e_b$, where e_a, e_b are the terms corresponding to the symbols a, b. Let $\mathcal{F}'$ denote the final E-SAT instance, and it is clear that $\mathcal{F}'$ is satisfiable if and only if the original 2-CSP is satisfiable, and also that $\mathcal{F}'$ has $n + d$ constant terms.

Now, apply the Ackermann reduction R to $\mathcal{F}'$, obtaining a SAT instance $R(\mathcal{F}')$. By assumption the final SAT instance has $o((n+d)\log(n+d))$ variables; running the standard brute-force algorithm for SAT gives an algorithm running in $2^{o((n+d)\log(n+d))}$ time for the 2-CSP variant described above. However, by the above theorem, every algorithm for this 2-CSP variant requires time at least $d^{cn} = 2^{cn \log d}$, which violates ETH if $d \approx n$.

7 Conclusion

In this paper, we studied SMT solvers through the lens of proof complexity, introducing a generalization of the resolution proof system and arguing that it

correctly models the "lazy" SMT framework $\mathsf{DPLL}(T)$ [NOT06]. We further presented and analyzed a stronger version $\mathsf{Res}^*(T)$ that allows for the introduction of new literals, and showed that it models $\mathsf{DPLL}^*(T)$, which is a modification of an SMT solver that can introduce new theory literals; this captures the new literal introduction in solvers such as Yices and Z3 [Z3, Yic].

There are many natural directions to pursue. First, although we have not considered it here, it is natural to introduce an *intermediate* proof system between $\mathsf{Res}(T)$ and $\mathsf{Res}^*(T)$ which is allowed to introduce new theory *literals* but *not* new theory *variables*. For instance, if we have the formula $a = f(b) \wedge a = c$ in EUF, then this intermediate proof system could introduce the theory literal $c = f(b)$ but *not* the theory literal $f(c) = f(a)$, whereas both are allowed to be introduced by $\mathsf{Res}^*(T)$. It is not clear to us if this intermediate system can simulate Frege, and we suggest studying it in its own right.

A second direction that we believe is quite interesting is extending our results on EUF to capture the *extended Frege* system, which is the most powerful proof system typically studied in proposition proof complexity. Intuitively, it seems that EUF by itself is not strong enough to capture extended Frege; we consider finding a new theory T which can capture it an interesting open problem.

References

[AFT11] Atserias, A., Fichte, J.K., Thurley, M.: Clause-learning algorithms with many restarts and bounded-width resolution. J. Artif. Intell. Res. **40**, 353–373 (2011)

[BBJ14] Bonet, M.L., Buss, S., Johannsen, J.: Improved separations of regular resolution from clause learning proof systems. J. Artif. Intell. Res. **49**, 669–703 (2014)

[BDdM08] Bjørner, N., Dutertre, B., de Moura, L.: Accelerating lemma learning using joins - DPLL(Join). In: 15th International Conference on Logic for Programming Artificial Intelligence and Reasoning, LPAR 2008 (2008)

[BHJ17] Balyo, T., Heule, M.J.H., Järvisalo, M.: SAT competition 2016: recent developments. In: Singh, S.P., Markovitch, S. (eds.) Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, 4–9 February 2017, San Francisco, California, USA, pp. 5061–5063. AAAI Press (2017)

[BKS04] Beame, P., Kautz, H.A., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. J. Artif. Intell. Res. **22**, 319–351 (2004)

[BM14] Bjørner, N., de Moura, L.: Tractability and modern SMT solvers. In: Bordeaux, L., Hamadi, Y., Kohli, P. (eds.) Tractability: Practical Approaches to Hard Problems, pp. 350–377. Cambridge University Press (2014)

[CR79] Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. J. Symb. Log. **44**(1), 36–50 (1979)

[DST80] Downey, P.J., Sethi, R., Tarjan, R.E.: Variations on the common subexpression problem. J. ACM (JACM) **27**(4), 758–771 (1980)

[GD07] Ganesh, V., Dill, D.L.: A Decision procedure for bit-vectors and arrays. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 519–531. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_52

[GHN+04] Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: DPLL(T): fast decision procedures. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 175–188. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27813-9_14

[Juk12] Jukna, S.: Boolean Function Complexity: Advances and Frontiers. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-24508-4

[Kra98] Krajíček, J.: Discretely ordered modules as a first-order extension of the cutting planes proof system. J. Symb. Log. **63**(04), 1582–1596 (1998)

[NOT06] Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories. J. ACM **53**(6), 937–977 (2006)

[ORC09] Oliveras, A., Rodr1guez-Carbonell, E.: Combining decision procedures: the Nelson-Oppen approach. Techniques (2009)

[PD11] Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers as resolution engines. Artif. Intell. **175**(2), 512–525 (2011)

[RT08] Raz, R., Tzameret, I.: Resolution over linear equations and multilinear proofs. Annals Pure Appl. Log. **155**(3), 194–224 (2008)

[smt] The Annual SMTCOMP Competition Website. http://www.smtcomp.org

[Tin12] Tinelli, C.: Foundations of Lazy SMT and DPLL(T) (2012)

[Tra08] Traxler, P.: The time complexity of constraint satisfaction. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 190–201. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79723-4_18

[Yic] The Yices SMT Solver. http://yices.csl.sri.com/

[Z3] The Z3 Theorem Prover. https://github.com/Z3Prover