On the proof complexity of MCSAT

Gereon Kremer
RWTH Aachen University
gereon.kremer@cs.rwth-aachen.de

Erika Ábrahám
RWTH Aachen University
eab@cs.rwth-aachen.de

Vijay Ganesh
University of Waterloo
vijay.ganesh@uwaterloo.ca

Abstract

Satisfiability Modulo Theories (SMT) and SAT solvers are critical components in many formal software tools, primarily due to the fact that they are able to easily solve logical problem instances with millions of variables and clauses. This efficiency of solvers is in surprising contrast to the traditional complexity theory position that the problems that these solvers address are believed to be hard in the worst case. In an attempt to resolve this apparent discrepancy between theory and practice, theorists have proposed the study of these solvers as proof systems that would enable establishing appropriate lower and upper bounds on their complexity. For example, in recent years it has been shown that (idealized models of) SAT solvers are polynomially equivalent to the general resolution proof system for propositional logic, and SMT solvers that use the CDCL(T) architecture are polynomially equivalent to the Res*(T) proof system.

In this paper, we extend this program to the MCSAT approach for SMT solving by showing that the MCSAT architecture is polynomially equivalent to the Res*(T) proof system. Thus, we establish an equivalence between CDCL(T) and MCSAT from a proof-complexity theoretic point of view. This is a first and essential step towards a richer theory that may help (parametrically) characterize the kinds of formulas for which MCSAT-based SMT solvers can perform well.

1 Introduction

In this work we are interested in proof systems to decide the satisfiability of (quantifier-free first-order logic) formulas. Well-known proof systems include variants of the *resolution proof system* for propositional logic, but also for first-order logic as presented in [RKG]. An interesting measure to compare proof systems is their *proof complexity*: if we can show that in a proof system we can generate shorter proofs (with less proof steps) than in another, we consider the former proof system *more powerful*.

One particular problem that concerns quantifier-free first-order logic formulas is to determine their satisfiability. Answering this question (at least for certain theories) has spawned an active field of research called

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

satisfiability modulo theories (SMT) solving [BHvMW09, KS08]. The predominant approach is called CDCL(T) and works by combining a solver for propositional logic (a *SAT solver*) with a solver that checks a set of theory constraints for consistency (a *theory solver*).

A significantly different solving technique called MCSAT was presented in [JdM, dMJ] which provides a significantly tighter integration of the Boolean and the theory reasoning. This solver performs extremely well in practice, at least for "hard" theories like non-linear real arithmetic, thus naturally raising the question whether this MCSAT implementation only happens to include better heuristics, or whether MCSAT itself has some more fundamental advantage over CDCL(T).

We tackle this question from the perspective of proof complexity, allowing us to abstract from certain practical aspects very naturally, for example the impact of heuristics. We can understand both CDCL(T) and MCSAT as proof systems [NOT06, dMJ]. The proof complexity of CDCL(T) was found in [RKG] to be equivalent to the Res*(T) proof system. In this paper, we show that also the MCSAT architecture is polynomially equivalent to the Res*(T) proof system. Thus, we establish an equivalence between CDCL(T) and MCSAT from a complexity-theoretic view.

It is important to realize, that there is a subtle but important difference between what we call DPLL – referring to the original DPLL method from [DLL62] – and the version extended with clause learning – proposed in [MS] – that we now call CDCL. Compared to CDCL(T) (and Res*(T)), DPLL(T) is weaker in terms of proof complexity and we only argue about CDCL(T) throughout this paper.

2 Preliminaries

2.1 Proof Systems

We assume the reader to be familiar with deductive proof systems and give in the following just a short introduction to notation. For readability, in the following we will use *conditional* proof rules of the form

$$\texttt{PR}\colon \frac{A_1 \ \dots \ A_n}{C} \qquad\qquad \textbf{if}\ \{\ \texttt{S}_1, \dots,\ \texttt{S}_m$$

to form proof systems. Such a rule with name PR allows to derive the consequent C from the antecedents $A_1, \dots, A_n$ *and some side conditions* $\texttt{S}_1, \dots, \texttt{S}_m$. We will typically describe some *state* in a single antecedent A and some conditions on this state in $\texttt{S}_1, \dots, \texttt{S}_m$. Note that the above conditional proof rule is equivalent to one without side-condition but with the antecedents $A_1, \dots, A_n, \texttt{S}_1, \dots, \texttt{S}_m$.

A *(deductive) proof system* is a set of such (conditional) proof rules. By a *derivability statement* $\Gamma \vdash C$ we denote that we can *prove* C from a set Γ of antecedents (and side conditions), i.e., that C is derivable from Γ by finitely many proof rule applications. By $\Gamma \models C$ we denote that Γ assures the truth of C (with respect to an underlying semantics). Qualitative properties of proof systems include soundness (if $\Gamma \vdash C$ then $\Gamma \models C$) and completeness (if $\Gamma \models C$ then $\Gamma \vdash C$). All the proof systems that we consider in this paper are sound and complete; we refer to the respective references for details.

The *length* of a proof in a proof system is the number of proof rule applications used in it; a proof is *shorter* then another one if its length is smaller. The *proof complexity* for $\Gamma \vdash C$ is the length of the *shortest* proof that derives C from Γ. Our aim in this paper is to compare for certain classes of derivability statements their proof complexities in different proof systems.

Assume two proof systems P_1 and P_2, and a metric to measure the size of derivability statements that are true in both systems. We say that P_1 is *more powerful* than P_2 on a set of such statements if the proof complexity grows (at most) polynomially with the statement (input) size in P_1 but exponentially in P_2. We call P_1 and P_2 *comparable* if one of them is more powerful than the other on *all* such classes, and *incomparable* otherwise. We refer to [RKG] for a discussion of existing proof systems and their relations in terms of proof complexity.

2.2 Model-constructing Satisfiability Calculus (MCSAT)

In the following we recall the MCSAT proof system for the satisfiability check of quantifier-free first-order logic formulas as defined in [dMJ]. Following [dMJ], we present the proof rules in three categories: *Boolean reasoning* (fig. 1), *conflict analysis* (fig. 2) and *theory reasoning* (fig. 3). The *Boolean reasoning* and *conflict analysis* parts essentially constitute a regular CDCL-style SAT solver while the *theory reasoning* part enhances the proof system to allow for SMT-style theory computations.

$$\texttt{Decide:}\frac{\langle M, \mathcal{C}\rangle}{\langle [\![M, L]\!], \mathcal{C}\rangle} \qquad \textbf{if} \begin{cases} L \in \mathit{Basis} \\ \text{value}(L, M) = \mathit{undef} \end{cases}$$

$$\texttt{Propagate:}\frac{\langle M, \mathcal{C}\rangle}{\langle [\![M, C \to L]\!], \mathcal{C}\rangle} \qquad \textbf{if} \begin{cases} C = (L_1 \vee \cdots \vee L_n \vee L) \in \mathcal{C}, \\ \forall i.\, \text{value}(L_i, M) = \mathit{false}, \text{value}(L, M) = \mathit{undef} \end{cases}$$

$$\texttt{Conflict:}\frac{\langle M, \mathcal{C}\rangle}{\langle M, \mathcal{C}\rangle \Vdash C} \qquad \textbf{if} \begin{cases} C = (L_1 \vee \cdots \vee L_n) \in \mathcal{C}, \\ \forall i.\, \text{value}(L_i, M) = \mathit{false} \end{cases}$$

$$\texttt{Sat:}\frac{\langle M, \mathcal{C}\rangle}{\texttt{SAT}} \qquad \textbf{if} \begin{cases} M \textit{ is complete}, \\ \forall C = (L_1 \vee \ldots \vee L_n) \in \mathcal{C}. \exists i.\, \text{value}(L_i, M) = \mathit{true} \end{cases}$$

$$\texttt{Forget:}\frac{\langle M, \mathcal{C}\rangle}{\langle M, \mathcal{C} \setminus \{C\}\rangle} \qquad \textbf{if} \left\{ C \in \mathcal{C} \textit{ is a learned clause} \right.$$

$$\texttt{Restart:}\frac{\langle M, \mathcal{C}\rangle \Vdash C}{\langle [\![]\!], \mathcal{C}\rangle} \qquad \textbf{if} \left\{ \mathit{true} \right.$$

Figure 1: The MCSAT proof rules I: Boolean reasoning

The only modification to [dMJ] is the addition of the `Restart` rule (fig. 1), which is sound and also does not introduce non-termination as long as we ensure that it is applied with increasing periodicity [NOT06]. Restarts are a standard technique both in CDCL-style SAT solving and in SMT solving, but such a rule was not included in the original MCSAT proof system. However, we will need this rule for the proof complexity results in the next section.

Due to space restrictions, in the following we discuss the rules only briefly and refer to [dMJ] for more detailed explanations.

The proof system works on states of the form $\langle M, \mathcal{C}\rangle$ consisting of a *trail* M and a set $\mathcal{C}$ of clauses whose satisfaction we are interested in, where the literals of the clauses are constraints from a given theory T. The trail is a list of Boolean decisions, theory decisions, Boolean propagations and theory propagations. Decisions represent exploration by enumeration: a Boolean decision L assigns the Boolean value *true* to the literal L (fig. 1, rule `Decide`; the *Basis* set and the value function will be explained a bit later), whereas a theory decision $x \mapsto \alpha_x$ assigns the theory value α_x to the theory variable x (fig. 3, rule `T-Decide`; consistency of a trail is explained below). A Boolean propagation $C \to L$ states that the satisfaction of $\mathcal{C}$ under previous decisions is possible only if L is *true* with the reason being the clause C (fig. 1, rule `Propagate`). Analogously, a theory propagation $E \to L$ states an implication based on a lemma, i.e. a tautology in the theory (`T-Propagate`, fig. 3). We write $[\![M_1, \ldots, M_k]\!]$ to explicitly denote individual elements of the trail, and we call a trail *complete* if it assigns a Boolean value to each variable or its negation (either by Boolean decision or propagation) as well as a theory value to each theory variable. Satisfiability of a CNF formula can be proven by deriving the `SAT` state, based on a complete trail that satisfies each clause (fig. 1, rule `Sat`).

For theory variables x, if there is some theory value α_x such that $x \mapsto \alpha_x$ is in the trail M then the value value(x, M) of a theory variable x in trail M is α_x, and *undef* otherwise.

A literal L, which is either a theory constraint c or the negation $\neg c$ of a constraint c, can be evaluated by two semantics: c or $\neg c$ can enter the trail by Boolean decision or propagation, but also the theory variables in c can be assigned a value from the theory domain. The proof system argues in both the Boolean and the theory domain concurrently, but it assures that they stay *consistent*, meaning that the Boolean and the theory evaluation of a constraint never contradict. Thus the value value(L, M) of a literal L in a trail M is *true* if L is decided or propagated in M, or if L evaluates to true when we substitute the theory values from M in L; it is *false* if $\neg L$ is decided or propagated in M, or if L evaluates to *false* under the theory assignments; and *undef* otherwise. We call a consistent trail M *feasible* if the trail's theory variable assignments can be extended to satisfy all clauses $C \in \mathcal{C}$ with value$(C, M) = \mathit{true}$; by infeasible(M) we denote that M is not feasible.

From a *search state* $\langle M, \mathcal{C}\rangle$ we enter the *conflict state* $\langle M, \mathcal{C}\rangle \Vdash C$ when a violated *conflict clause* (i.e., a clause whose literals all evaluate to *false* under the current trail M) is detected (fig. 1, rule `Conflict`), or the conflict state $\langle M, \mathcal{C}\rangle \Vdash E$ when the trail is infeasible and E is a theory lemma explaining this fact. The

$$\texttt{Resolve}: \frac{\langle [\![M, D \to L]\!], \mathcal{C}\rangle \Vdash C}{\langle M, \mathcal{C}\rangle \Vdash R} \qquad \textbf{if} \begin{cases} \neg L \in C, \\ R = \text{resolve}(C, D, L) \end{cases}$$

$$\texttt{Consume}_1: \frac{\langle [\![M, D \to L]\!], \mathcal{C}\rangle \Vdash C}{\langle M, \mathcal{C}\rangle \Vdash C} \qquad \textbf{if} \left\{ \neg L \notin C \right.$$

$$\texttt{Consume}_2: \frac{\langle [\![M, L]\!], \mathcal{C}\rangle \Vdash C}{\langle M, \mathcal{C}\rangle \Vdash C} \qquad \textbf{if} \left\{ \neg L \notin C \right.$$

$$\texttt{Backjump}: \frac{\langle [\![M, N]\!], \mathcal{C}\rangle \Vdash C}{\langle [\![M, C \to L]\!], \mathcal{C}\rangle} \qquad \textbf{if} \begin{cases} C = (L_1 \vee \cdots \vee L_n \vee L), \\ \forall i.\, \text{value}(L_i, M) = \mathit{false}, \\ \text{value}(L, M) = \mathit{undef}, \\ N \textit{ starts with a decision} \end{cases}$$

$$\texttt{Unsat}: \frac{\langle M, \mathcal{C}\rangle \Vdash \mathit{false}}{\texttt{UNSAT}} \qquad \textbf{if} \left\{ \mathit{true} \right.$$

$$\texttt{Learn}: \frac{\langle M, \mathcal{C}\rangle \Vdash C}{\langle M, \mathcal{C} \cup \{C\}\rangle \Vdash C} \qquad \textbf{if} \left\{ C \notin \mathcal{C} \right.$$

Figure 2: The MCSAT proof rules II: conflict analysis

rules `Resolve`, `Consume`$_1$, `Consume`$_2$, `T-Consume`, `Backjump`, `T-Backjump-Decide` and `Learn` allow to implement conflict resolution in the style of conflict-driven clause learning (CDCL), where *resolve*(C, D, L) denotes the result of the (propositional) resolution applied to the clauses C and D with respect to the literal L. Unsatisfiability of a CNF formula can be proven by deriving the `UNSAT` state from the empty trail, based on the derivation of the empty (*false*) clause (fig. 2, rule `Unsat`).

The explain function explains theory-specific propagations and infeasible states. The proof systems assumes the existence of a *finite* set *Basis* of theory constraints such that all literals from all clauses, especially those returned by the explain function, are contained in it. The finiteness provides a nice termination argument as only finitely many (different) clauses exist. Also explanations can be learned and forgotten (rules `Learn` in fig. 2 and `Forget` in fig. 1).

2.3 The Proof Systems Res(T) and Res*(T)

Our reference proof system is (one of the variants of) the resolution proof system. Two resolution proof systems for first-order logic with some theory T called Res(T) and Res*(T) are discussed in [RKG]. They enhance the traditional resolution proof system for propositional logic by a proof rule that allows for the introduction of new clauses that state tautologies from the theory. While Res*(T) allows for *strong theory derivation* which may also introduce new literals (theory constraints), Res(T) is restricted to *(regular) theory derivation* that allows only literals that occur in the formula already.

Introducing new literals can make a significant difference for certain problem classes as discussed in [RKG]. As the MCSAT proof system may as well introduce new literals – the explain method makes heavy use of this in many cases – we use Res*(T) for the following comparison, consisting of the `Resolution` rule and the `Strong Theory Derivation` rule.

Note that these proof systems lack dedicated final states – like `SAT` and `UNSAT` from MCSAT. We implicitly derive `UNSAT` if `Resolution` produces the empty clause and `SAT` if no new clauses can be derived (with `Resolution` or `(Regular) Theory Derivation`), thus essentially when a fixed point is reached. Note that this proof system does not (directly) allow to extract a satisfying assignment.

3 Content

We now state and afterwards prove our core theorem.

Theorem 1. *The Res*(T) proof system and the MCSAT proof system are* equivalent *with respect to their proof complexity on* first-order logic with any theory*.*

$$\texttt{T-Propagate:}\frac{\langle M,\mathcal{C}\rangle}{\langle[\![M,E\rightarrow L]\!],\mathcal{C}\rangle}\qquad\textbf{if}\begin{cases}L\in \mathit{Basis},\\ \text{value}(L,M)=\mathit{undef},\\ \text{infeasible}([\![M,\neg L]\!]),\\ E=\text{explain}([\![M,\neg L]\!])\end{cases}$$

$$\texttt{T-Decide:}\frac{\langle M,\mathcal{C}\rangle}{\langle[\![M,x\mapsto\alpha_x]\!],\mathcal{C}\rangle}\qquad\textbf{if}\begin{cases}x\ \mathit{is\ a\ theory\ variable\ in}\ \mathcal{C},\\ \text{value}(x,M)=\mathit{undef},\\ [\![M,x\mapsto\alpha_x]\!]\ \mathit{is\ consistent}\end{cases}$$

$$\texttt{T-Conflict:}\frac{\langle M,\mathcal{C}\rangle}{\langle M,\mathcal{C}\rangle\Vdash E}\qquad\textbf{if}\begin{cases}\text{infeasible}(M),\\ E=\text{explain}(M)\end{cases}$$

$$\texttt{T-Consume:}\frac{\langle[\![M,x\mapsto\alpha_x]\!],\mathcal{C}\rangle\Vdash C}{\langle M,\mathcal{C}\rangle\Vdash C}\qquad\textbf{if}\begin{cases}\text{value}(C,M)=\mathit{false}\end{cases}$$

$$\texttt{T-Backjump-Decide:}\frac{\langle[\![M,x\mapsto\alpha_x,N]\!],\mathcal{C}\rangle\Vdash C}{\langle[\![M,L]\!],\mathcal{C}\rangle}\qquad\textbf{if}\begin{cases}C=(L_1\vee\cdots\vee L_n\vee L),\\ \exists i.\,\text{value}(L_i,M)=\mathit{undef},\\ \text{value}(L,M)=\mathit{undef}\end{cases}$$

Figure 3: The MCSAT proof rules III: theory reasoning

$$\texttt{Resolution:}\frac{(C\vee l)\quad(D\vee\neg l)}{(C\vee D)}\qquad\textbf{if}\begin{cases}\mathit{true}\end{cases}$$

$$\texttt{(Regular) Theory Derivation:}\frac{\varphi}{\varphi\wedge C}\qquad\textbf{if}\begin{cases}T\models C,\\ l\in\varphi\ \text{for all}\ l\in C\end{cases}$$

$$\texttt{Strong Theory Derivation:}\frac{\varphi}{\varphi\wedge C}\qquad\textbf{if}\begin{cases}T\models C\end{cases}$$

Figure 4: The Res(T) and Res*(T) proof systems

We give our proof in two steps: first we show that MCSAT can simulate Res*(T) – meaning that for any Res*(T) proof MCSAT can construct a proof that is at most polynomially longer – and finally show the reverse statement, that Res*(T) can simulate MCSAT. What we show is actually a slightly stronger statement: instead of constructing *some* proof (that is at most polynomially longer) we construct a logically equivalent proof, yielding something we could describe as *algorithmic equivalency.* Of course these proofs will not be *syntactically identical*, but they describe *logically equivalent* proof steps.

3.1 MCSAT simulates Res*(T)

To show that MCSAT can simulate Res*(T), we need to show that MCSAT can simulate both the `Resolution` rule and the `Strong Theory Derivation` rule with at most polynomial overhead.

`Resolution`.

Assuming that our set of clauses $\mathcal{C}$ contains the clauses $(C\vee L)$ and $(D\vee\neg L)$, we need to add $(C\vee D)$ to $\mathcal{C}$. Let us first handle a special case. Assume that we have a literal in C whose negation is in D. In this case $(C\vee D)\equiv \mathit{true}$ and there is nothing to do. Similarly, if $(C\vee D)\in\mathcal{C}$ then we do not need to add it. From here on we assume that $(C\vee D)\not\equiv \mathit{true}$ and $(C\vee D)\notin\mathcal{C}$. Starting from an empty trail, the clause $(C\vee D)$ can be learned using the MCSAT proof rules as follows:

First we apply the `Decide` rule for all literals $L_1, \ldots, L_n$ of C and D. Note that we start with the empty trail, thus initially all literals are undefined and can therefore be decided. Note furthermore that we assumed a finite *Basis* set that contains all literals from all clauses in $\mathcal{C}$.

$$\texttt{Decide:}\frac{\langle M, \mathcal{C}\rangle}{\langle [\![M, \neg L_i]\!], \mathcal{C}\rangle} \qquad \textbf{if} \begin{cases} L_i \in \mathit{Basis}, \\ \text{value}(L_i, M) = \mathit{undef} \end{cases}$$

Now, the trail evaluates both C and D to *false* and we use the `Propagate` rule with $(C \vee L)$ to propagate L.

$$\texttt{Propagate:}\frac{\langle M, \mathcal{C}\rangle}{\langle [\![M, (C \vee L) \to L]\!], \mathcal{C}\rangle} \qquad \textbf{if} \begin{cases} \text{value}(C, M) = \mathit{false}, \\ \text{value}(L, M) = \mathit{undef} \end{cases}$$

Having value$(L, M) = \mathit{true}$ now, $(D \vee \neg L)$ is conflicting and we apply the `Conflict` rule.

$$\texttt{Conflict:}\frac{\langle [\![M, (C \vee L) \to L]\!], \mathcal{C}\rangle}{\langle [\![M, (C \vee L) \to L]\!], \mathcal{C}\rangle \Vdash (D \vee \neg L)} \qquad \textbf{if} \begin{cases} (D \vee \neg L) \in \mathcal{C}, \\ \text{value}(D \vee \neg L) = \mathit{false} \end{cases}$$

We perform resolution using the `Resolve` rule to obtain $(C \vee D)$.

$$\texttt{Resolve:}\frac{\langle [\![M, (C{\vee}L){\to}L]\!], \mathcal{C}\rangle \Vdash (D{\vee}\neg L)}{\langle M, \mathcal{C}\rangle \Vdash (C{\vee}D)} \qquad \textbf{if} \begin{cases} L \in (C{\vee}L), \\ (C{\vee}D) = \text{resolve}(\\ \quad (C{\vee}L), (D{\vee}\neg L), L) \end{cases}$$

To add the conflict clause to the set of clauses $\mathcal{C}$ we use the `Learn` rule.

$$\texttt{Learn:}\frac{\langle M, \mathcal{C}\rangle \Vdash (C \vee D)}{\langle M, \mathcal{C} \cup \{(C \vee D)\}\rangle \Vdash (C \vee D)} \qquad \textbf{if} \{\ (C \vee D) \notin \mathcal{C}$$

We have achieved our goal of adding $(C \vee D)$ to $\mathcal{C}$ and return to the initial state with the `Restart` rule.

$$\texttt{Restart:}\frac{\langle M, \mathcal{C} \cup \{(C \vee D)\}\rangle \Vdash (C \vee D)}{\langle [\![]\!], \mathcal{C} \cup \{(C \vee D)\}\rangle} \qquad \textbf{if} \{\ \mathit{true}$$

We observe that this sequence of proof rules is polynomial in the size of the clause $(C \vee D)$ – we need $|C|+|D|+5$ rule applications – and we return to the same initial state afterwards, except for the added clause $(C \vee D)$. Note that we did not use any theory reasoning in the MCSAT rule applications and thus pay no *hidden costs* (as we later discuss in section 4).

`Strong Theory Derivation`.

We need to create some arbitrary clause C which is valid in the theory, that is $T \models C$. Similar to the previous simulation, we assume that $C \not\equiv \mathit{true}$ and $C \notin \mathcal{C}$ as there is nothing to do in these cases.

Our main hurdle is that MCSAT does not allow for learning arbitrary clauses but only the current conflict clause. We therefore have to construct an (artificial) conflict that yields the desired clause. We assume that our finite basis *Basis* includes all literals that ever occur in the Res*(T) proof. We can construct and learn an arbitrary (valid) clause C using the MCSAT proof rules as follows:

Starting again from the empty trail, we use `Decide` repeatedly to add $\neg L$ for every $L \in C$ to the trail. Note that `Decide` allows to decide any literal from *Basis*, independent on whether they appear in the input formula.

$$\texttt{Decide:}\frac{\langle M, \mathcal{C}\rangle}{\langle [\![M, \neg L]\!], \mathcal{C}\rangle} \qquad \textbf{if} \begin{cases} L \in \mathit{Basis} \\ \text{value}(L, M) = \mathit{undef} \end{cases}$$

We know that C is a valid clause ($T \models C$) but we have value$(C, M) = \textit{false}$ due to the previous decisions. Thus M is infeasible and we can apply the `T-Conflict` rule with $E = C$. Recall that a trail is *infeasible* if it is inconsistent in the theory – which is the case here as M implies $\neg C$ but $T \models C$.

$$\texttt{T-Conflict:}\frac{\langle M, \mathcal{C}\rangle}{\langle M, \mathcal{C}\rangle \Vdash C} \qquad \textbf{if}\begin{cases}\text{infeasible}(M),\\ C = \text{explain}(M)\end{cases}$$

Now we can learn the desired clause C using the `Learn` rule.

$$\texttt{Learn:}\frac{\langle M, \mathcal{C}\rangle \Vdash C}{\langle M, \mathcal{C} \cup \{C\}\rangle \Vdash C} \qquad \textbf{if}\left\{\ C \notin \mathcal{C}\right.$$

Finally we return to the initial state with the `Restart` rule.

$$\texttt{Restart:}\frac{\langle M, \mathcal{C} \cup \{C\}\rangle \Vdash C}{\langle [\![]\!], \mathcal{C} \cup \{C\}\rangle} \qquad \textbf{if}\left\{\ \textit{true}\right.$$

We observe that we need $|C| + 3$ proof rule applications to learn an arbitrary clause (that is neither *true* nor already present in $\mathcal{C}$) and return to the initial state.

3.1.1 Practicability

We have seen that the MCSAT framework assumes that it can decide the feasibility of the trail, though (depending on the theory) deciding (in)feasibility might be computationally hard or even undecidable. While theorists might simply assume an "oracle" we now examine how actual implementations deal with this. Practically relevant implementations of infeasible(M) for non-linear real arithmetic are incomplete. The respective papers suggest to apply the `T-Conflict` rule only if infeasibility can be detected by checking the consistency of *univariate* constraints. One might even (reasonably) argue that this restriction is not a technical one, but is a fundamental idea in MCSAT that allows the theory exploration in the first place – a complete implementation of infeasible(M) would essentially prevent the theory exploration from happening (like in our simulation). Simulating the `Strong Theory Derivation` rule with an incomplete implementation of infeasible(M) is a bit technical but doable, however, we need to consider also the length of the resulting proof.

The basic idea of the theory exploration in MCSAT is to guess a partial theory assignment (via `T-Decide`) until its infeasibility can be detected, generalize the unsatisfying assignment to an unsatisfying region around it (via `T-Conflict`) and exclude it (via `Learn`) from the further search and backtrack the theory assignment. This usually happens multiple times for a certain variable until the whole space is excluded for this variable and we have to change the assignment of the previously assigned theory variable.

In the above scenario the trail is already unsatisfiable due to the Boolean assignments, and we eventually discover this fact after exploring a certain number of regions and come up with a reason for the conflict. Note that the number of rule applications needed in this case grows with the number of regions we need to exclude and the complexity of this process highly depends on the background theory. For the question of completeness we refer to [dMJ] while we observe that for non-linear arithmetic, considering the number of cells a cylindrical algebraic decomposition may have to consider, the number of regions may easily grow exponentially (e.g. in the number of theory variables).

This unfortunately conflicts with our hope to find a *polynomial* reduction. However, this was to be expected: a major aspect of MCSAT is that certain parts of the theory reasoning are moved into the core proof system. Thus we should not be surprised if our core proof system exhibits bad asymptotic behavior if we consider a hard theory, while the proof system we compare it to completely *hides* the theory reasoning from the complexity measure. Note that theory reasoning for non-linear arithmetic based on cylindrical algebraic decomposition, whether it is infeasible(M) in MCSAT or the question whether $T \models C$ in Res*(T) may incur a doubly exponential runtime cost. We discuss how these two theory questions relate in section 4.

We thus conclude that the presented reduction is polynomial for the MCSAT proof system, but may not be for an actual MCSAT implementation. The additional complexity is however not *new* but only *becomes visible* as the MCSAT proof system makes the theory reasoning explicit while Res*(T) does not.

3.2 Res*(T) simulates MCSAT

We observe that MCSAT has three separate places where clauses can "exist", namely the set of clauses $\mathcal{C}$, the trail M and the current conflict clause C. When simulating MCSAT with Res*(T), we make sure that the set of clauses that Res*(T) operates on always includes $\mathcal{C}$, clauses from M and C.

Note that Res*(T) retains all clauses that it constructs as is not designed to *forget* clauses while MCSAT may drop clauses occasionally. We note that removing clauses can bring advantages in practice, but the number of additional clauses is linear in the number of rule applications – MCSAT needs at least one rule application to construct a clause in the first place – and the practical overhead of additional clauses – for example due to larger lookup tables – is polynomial.

To prove that Res*(T) simulates MCSAT, it suffices to show that all clauses that ever occur in the MCSAT derivation can also be constructed using the Res*(T) proof rules. In case of unsatisfiability Res*(T) deduces the empty clause immediately before the application of the `Unsat` rule in MCSAT while any other termination of the Res*(T) proof system indicates satisfiability. We assume that initially both proof systems start with the same set of input clauses. We identify the rules where the MCSAT rule system constructs new clauses: `Resolve`, `T-Propagate` and `T-Conflict`.

All other rules either do not manipulate any clauses (`Decide`, `Sat`, `Consume`$_1$, `Consume`$_2$, `Unsat`, `T-Decide`, `T-Consume`, `Restart`), move clauses between any of the three places (`Propagate`, `Conflict`, `Backjump`, `Learn`) or remove existing clauses from the current state (`Forget`, `T-Backjump-Decide`).

`Resolve`.

The `Resolve` rule takes the two clauses C (the current conflict clause) and D (from the trail M) and produces the resolvent with respect to some literal L. By construction we know that Res*(T) has both C and D available and can thus apply the `Resolution` rule to produce the same resolvent.

`T-Propagate` and `T-Conflict`.

Both the `T-Propagate` rule and the `T-Conflict` rule construct a new clause E by calling the MCSAT explain method. This method produces "a valid theory lemma" (as specified in [dMJ]), thus we have $T \models E$ and can obtain these clauses using the `Strong Theory Derivation` rule.

As we have simulated all MCSAT rules that can be used to construct new clauses, we can now take any MCSAT proof and convert it into a Res*(T) proof by using the presented simulations for the `Resolve`, `T-Propagate` and `T-Conflict` rule and removing all other proof steps.

We have shown that MCSAT and Res*(T) simulate each other and are thus equivalent (in terms of proof complexity), concluding our proof for theorem 1.

4 Complexity of theory computations

One can very well argue that the length of a proof (in terms of proof steps) is not a satisfactory measure to assess the computational complexity of performing this proof. We already see in the analysis above that we can make proof rules arbitrarily powerful so that we can essentially prove anything with a single step. We could as well devise a prove system that executes individual processor instructions and thus requires an insane amount of proof steps. To compare proof systems it is thus crucial that they operate on a similar level of abstraction as we assume every proof rule to take some fixed amount of time.

One such abstraction is that we assume that theory queries (whatever a *theory query* may be in detail) only require constant effort here: one proof step for `Strong Theory Derivation` in Res*(T) and one proof step for either `T-Propagate` or `T-Conflict` in MCSAT. One may very well be worried here, as we know that theory queries for some theories can be extremely expensive (even doubly exponential for non-linear arithmetic using cylindrical algebraic decomposition) while all the other rules are easy to compute. This leads to two issues that influence how meaningful the above analysis is: how often are these particular rules used and do these theory queries have comparable computational effort?

If we use theory queries (almost) equally often in both proof systems, we can relax our assumption that all proof rules take constant time. Instead we consider *theory rules* (`Strong Theory Derivation`, `T-Propagate` and `T-Conflict`) and *non-theory rules*. We claim that all non-theory rules can be applied *quickly* (low polynomial runtime) and it suffices to consider their overall number as we did above. If we can additionally show that the theory rules have comparable computational effort in the different proof systems, this implies that the overall effort is (complexity-wise) the same.

Recalling the above reductions, we see that both proof systems need the exact same amount of theory rules. MCSAT needs no theory rules to simulate `Resolution` and a single application of `T-Conflict` to simulate `Strong Theory Derivation`. Res*(T) on the other hand only applies `Strong Theory Derivation` once to simulate `T-Conflict` or `T-Propagate`.

What remains to analyze is the computational effort for each of these theory queries. When Res*(T) simulates MCSAT (as described above) it generates the same clauses that MCSAT generated: if we would have an implementation for `Strong Theory Derivation` that would do this (significantly) more efficiently than whatever MCSAT does in the explain and infeasible methods, we could take this method and use it in MCSAT as well. We have also seen that MCSAT can simulate the `Strong Theory Derivation` rule (with a single theory call to the infeasible method while explain is trivial as no theory assignments are present [dMJ]) and thus a more efficient infeasible method would directly allow us to improve upon the implementation of the `Strong Theory Derivation` rule. We can just assume that both proof systems may use the exact same techniques for the theory queries and thus their computational effort is the same.

5 Impact on CDCL(T)

From the perspective of the SMT community the comparison to Res*(T) is of course interesting, but the actual question is how MCSAT relates to CDCL(T). Though we did not answer this specifically, we can both infer and speculate now more substantively than before. We know from [RKG] that CDCL(T) (with strong theory derivation) is equivalent to Res*(T) in terms of proof complexity as well, hence transitively MCSAT and CDCL(T) are equivalent (in terms of proof complexity).

CDCL(T) is presented as an algorithm in [RKG], but [NOT06] also (equivalently) defines CDCL(T) as a proof system. Though we cannot claim *equivalency* (beyond the aspect of proof complexity) between CDCL(T) and Res*(T), we can very well do so between MCSAT and Res*(T) as argued above. We conjecture that most proof rules from CDCL(T) and MCSAT nicely align and the ideas from the reductions above should be a good start for the remaining proof rules. Thus we believe that this work might be a first step towards a proof that establishes equivalency between CDCL(T) and MCSAT.

Like for MCSAT the theory queries within CDCL(T) are much more specific than those from Res*(T) and thus we expect the issues discussed in section 4 to essentially vanish here as well. Though the theory queries pose different questions, we know from practical implementations that the algorithms to answer them are very similar.

6 Conclusion

We have recalled the definitions of MCSAT (from [dMJ]) and Res*(T) (from [RKG]) and shown that they are equivalent with respect to their proof complexity, and even equivalent in a stronger sense that we described as *algorithmically equivalent*. We have discussed that though the complexity of theory queries is unknown, the fact that we simulate every theory query individually allows us to assume that the required effort is the same in both proof systems: if one would be (significantly) faster, we assume that the presented simulation can be used in the other proof system as well with only polynomial overhead. Still a more thorough analysis would be important to conduct in the future.

Finally we have discussed the importance of the presented work for the actual goal, the comparison of MCSAT and CDCL(T). We have noted that we established equivalency (in terms of proof complexity) of MCSAT and CDCL(T) which gives rise to the hope that they are in fact *equivalent* in the sense that they can (almost) directly simulate each other using the ideas presented here. A corresponding proof is also left for future work.

References

[BHvMW09] A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[dMJ] Leonardo de Moura and Dejan Jovanović. A model-constructing satisfiability calculus. In *Proceedings of VMCAI 2013*, volume 7737 of *LNCS*, pages 1–12.

[JdM] Dejan Jovanović and Leonardo de Moura. Solving non-linear arithmetic. In *Proceedings of IJCAR 2012*, pages 339–354.

[KS08] Daniel Kroening and Ofer Strichman. *Decision Procedures: An Algorithmic Point of View.* Springer, 2008.

[MS] João P. Marques Silva and Karem A. Sakallah. Grasp: a new search algorithm for satisfiability. In *Proceedings of ICCAD 1996*, pages 220–227.

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53:937–977, 2006.

[RKG] Robert Robere, Antonina Kolokolova, and Vijay Ganesh. The proof complexity of SMT solvers. In *Proceedings of CAV 2018*, volume 10982 of *LNCS*, pages 275–293.