Mining Propositional Simplification Proofs for Small Validating Clauses

Ian Wehrman and Aaron Stump
Computer Science and Engineering, Washington University in St. Louis
{iwehrman,stump}@cse.wustl.edu, http://cl.cse.wustl.edu/

Abstract

The problem of obtaining small conflict clauses in SMT systems has received a great deal of attention recently. We report work in progress to find small subsets of the current partial assignment that imply the goal formula when it has been propositionally simplified to a boolean value. The approach used is *algebraic proof mining*. Proofs from a propositional reasoner that the goal is equivalent to a boolean value (in the current assignment) are viewed as first-order terms. An equational theory between proofs is then defined, which is sound with respect to the quasi-order "proves a more general set theorems." The theory is completed to obtain a convergent rewrite system that puts proofs into a canonical form. While our canonical form does not use the smallest subset of the current assignment, it does drop many unnecessary parts of the proof. The paper concludes with discussion of the complexity of the problem and effectiveness of the approach.

1 Introduction

The problem of obtaining small conflict clauses, long known in the SAT community to be crucial for high performance [8], has recently been of great interest to the satisfiability modulo theories (SMT) community [5, 3]. This paper reports work in progress to find small subsets of the current partial assignment that imply the goal formula when it has been propositionally simplified to a boolean value (*true* or *false*). We refer to clauses that imply the goal formula as *validating clauses*. Validating clauses in the context of validity correspond to conflict clauses in the context of satisfiability. The techniques we propose can be applied to either problem, but we refer only to the first case in the paper.

SMT tools like CVC [6] and CVC Lite [2] work (roughly) by first choosing an atomic formula to case split on, followed by simplification of the (non-clausal) goal. Splitting and simplification proceeds until the formula simplifies to a boolean value. If that value is *true*, we would like to record

a subset of the current partial assignment as a validating clause. The current assignment itself is not useful in guiding the search, but a small proper subset can be. We hope to find such a small subset because incremental simplification is often redundant in the following sense. Consider the formula $(\chi \vee \psi) \wedge \phi$. Simplification may proceed (left to right) as follows:

$$(\chi \vee \psi) \wedge \phi \overset{\chi \Leftrightarrow F}{\dashrightarrow} \psi \wedge \phi \overset{\psi \Leftrightarrow T}{\dashrightarrow} \phi \overset{\phi \Leftrightarrow T}{\dashrightarrow} T. \tag{1}$$

Here, the assignment with domain χ,ψ,ϕ is a validating clause, implying $(\chi \vee \psi) \wedge \phi \Leftrightarrow T$. It is easy to see, however, that the first decision in this assignment is redundant. If simplification had omitted the first decision,

$$(\chi \vee \psi) \wedge \phi \overset{\psi \Leftrightarrow T}{\dashrightarrow} \phi \overset{\phi \Leftrightarrow T}{\dashrightarrow} T, \tag{2}$$

the formula would still simplify to T, resulting in a smaller and potentially more useful validating clause.

SMT tools such as CVC generate *proofs* of simplification. These proofs correspond to the step-by-step simplification of the goal to a boolean value. The main observation of this paper is that proofs can be transformed to find smaller validating clauses. Given a formula ϕ and proof that $\phi \Leftrightarrow T$ from simplification, we can reduce the proof using a term rewriting system (TRS) to one using a smaller assignment — proofs as in (1) are reduced to proofs as in (2). This is of potential value for systems that rely on simplifying a non-clausal goal formula, such as CVC Lite which uses both clausal and non-clausal forms of the goal, the former in order to obtain validating clauses [1]. The method proposed in this paper could be used to obtain small validating clauses without resorting to clausal form. In addition, studying the transformation of proofs could pave the way for more sophisticated proof mining techniques and applications.

The approach used is *algebraic proof mining* [7]. Proofs obtained from propositional simplification are viewed as first-order terms. A (finite) equational theory between proofs is defined, which is sound with respect to the quasi-order "proves a more general set of theorems." The theory is completed to obtain a convergent TRS which puts proofs into a canonical form. While our rewrite system does not result in a canonical form that uses the smallest subset of the current assignment, it does remove clearly unnecessary parts of the proof. These unnecessary parts correspond to the simplifications performed, e.g., on the left side of a disjunction whose right side simplifies to *true*. The simplifications performed on the left disjunct are unnecessary, because the whole disjunction will simplify to *true* whether or not the left disjunct is simplified.

In Sec. 2, we describe the propositional formulas and proofs under consideration and, in Sec. 3, we present a term rewriting system for proof reduction.

2 Propositional Equivalence Formulas and Proofs

We begin by defining the propositional formulas considered in our system and then describe a set of first-order terms that represent the proof rules. Finally, we give an equational theory for simplifying proof terms, along with a completed set of rewrite rules.

Let A be a countable set of propositional variables, V the set of boolean values $\{T, F\}$ and $\mathcal{S}$ the set of propositional formulas defined inductively by

$$\mathcal{S} ::= \ A \mid (\mathcal{S} \vee \mathcal{S}) \mid (\mathcal{S} \wedge \mathcal{S}) \mid \neg\mathcal{S}.$$

Let $\mathcal{D}$ be the set of *equivalence formulas* and $\mathcal{E}$ be the set of *boolean-valued equivalence formulas* defined inductively by

$$\mathcal{D} ::= \mathcal{S} \Leftrightarrow \mathcal{S} \mid \mathcal{E}$$
$$\mathcal{E} ::= \mathcal{S} \Leftrightarrow V.$$

Note that $\mathcal{E} \subset \mathcal{D}$ and $\mathcal{E}$ only contains formulas of the form $\phi \Leftrightarrow T$ and $\phi \Leftrightarrow F$. We use the notation $Var(\phi)$ to denote the set of propositional variables occurring in formula ϕ.

A *decision* $u = \langle a, v\rangle$ is associates a propositional variable $a \in A$ with a boolean value $v \in V$. An *assignment* $\mathcal{U}$ is any set of decisions that is a partial function; its *extension* $\hat{\mathcal{U}} = \{\langle u, a \Leftrightarrow v\rangle \mid u = \langle a, v\rangle \in \mathcal{U}\}$ is the relation between the decisions of an assignment and the respective theorems they prove. We inductively define the set of *equivalence proofs* $\mathcal{P}$ with the following grammar:

$$\begin{aligned}\mathcal{P} ::= {} & \mathcal{U} \mid \mathsf{Trans}(\mathcal{P}, \mathcal{P}) \mid \mathsf{Refl} \mid \mathsf{NotFalse} \mid \mathsf{NotTrue} \mid \mathsf{OrTrue1} \mid \mathsf{OrTrue2} \mid \\ & \mathsf{OrFalse1} \mid \mathsf{OrFalse2} \mid \mathsf{AndTrue1} \mid \mathsf{AndTrue2} \mid \mathsf{AndFalse1} \mid \mathsf{AndFalse2} \mid \\ & \mathsf{CongrOr1}(\mathcal{P}) \mid \mathsf{CongrOr2}(\mathcal{P}) \mid \mathsf{CongrAnd1}(\mathcal{P}) \mid \mathsf{CongrAnd2}(\mathcal{P}) \mid \\ & \mathsf{CongrNot}(\mathcal{P}).\end{aligned}$$

The proof rules (if not their notation) should be familiar to the reader: Refl denotes reflexivity of equivalence, Trans the transitivity of equivalence, etc. A complete description of the rules is given in Fig. 1.

Let $\vdash_{\mathcal{D}}$ be the smallest relation on $\mathcal{P} \times \mathcal{D}$ that extends $\hat{\mathcal{U}}$ and denotes the set of theorems proved by an equivalence proof, defined inductively by the universal closures of the formulas in Fig. 1. Instead of $\langle p, \phi\rangle \in \vdash_{\mathcal{D}}$ we write $p \vdash_{\mathcal{D}} \phi$ (read "p proves ϕ"). The relation $\vdash$ is the restriction of $\vdash_{\mathcal{D}}$ to $\mathcal{P} \times \mathcal{E}$.

We define the *proof generality quasi-order* as the smallest relation $\succcurlyeq_{\mathcal{D}}$ on $\mathcal{P} \times \mathcal{P}$ such that $p_1 \succcurlyeq_{\mathcal{D}} p_2$ iff $\forall \phi \in \mathcal{D}.\, p_1 \vdash_{\mathcal{D}} \phi \Rightarrow p_2 \vdash_{\mathcal{D}} \phi$. Similarly, let $\succcurlyeq$ be the relation on $\mathcal{P} \times \mathcal{P}$ such that $p_1 \succcurlyeq p_2$ iff $\forall \phi \in \mathcal{E}.\, p_1 \vdash \phi \Rightarrow p_2 \vdash \phi$. Note $\succcurlyeq$ is a subset of $\succcurlyeq_{\mathcal{D}}$. We say p_2 is *more general than* p_1 if $p_1 \succcurlyeq p_2$. For example, $\mathsf{Trans}(\mathsf{CongrAnd1}(p_1), \mathsf{AndFalse2}) \succcurlyeq \mathsf{AndFalse2}$ because, if $p_1 \vdash \phi \Leftrightarrow \phi'$, both

u	$\vdash_{\mathcal{D}}$	$a \Leftrightarrow v$	if $a \in A, v \in V, u = \langle a, v \rangle \in \mathcal{U}$
$\mathsf{Trans}(p_1, p_2)$	$\vdash_{\mathcal{D}}$	$\phi \Leftrightarrow \chi$	if $p_1 \vdash_{\mathcal{D}} \phi \Leftrightarrow \psi, p_2 \vdash_{\mathcal{D}} \psi \Leftrightarrow \chi$
Refl	$\vdash_{\mathcal{D}}$	$\phi \Leftrightarrow \phi$	
OrTrue1	$\vdash_{\mathcal{D}}$	$T \vee \phi \Leftrightarrow T$	
OrTrue2	$\vdash_{\mathcal{D}}$	$\phi \vee T \Leftrightarrow T$	
OrFalse1	$\vdash_{\mathcal{D}}$	$F \vee \phi \Leftrightarrow \phi$	
OrFalse2	$\vdash_{\mathcal{D}}$	$\phi \vee F \Leftrightarrow \phi$	
AndTrue1	$\vdash_{\mathcal{D}}$	$T \wedge \phi \Leftrightarrow \phi$	
AndTrue2	$\vdash_{\mathcal{D}}$	$\phi \vee T \Leftrightarrow \phi$	
AndFalse1	$\vdash_{\mathcal{D}}$	$F \wedge \phi \Leftrightarrow F$	
AndFalse2	$\vdash_{\mathcal{D}}$	$\phi \wedge F \Leftrightarrow F$	
NotFalse	$\vdash_{\mathcal{D}}$	$\neg F \Leftrightarrow T$	
NotTrue	$\vdash_{\mathcal{D}}$	$\neg T \Leftrightarrow F$	
$\mathsf{CongrOr1}(p_1)$	$\vdash_{\mathcal{D}}$	$\phi \vee \psi \Leftrightarrow \phi' \vee \psi$	if $p_1 \vdash_{\mathcal{D}} \phi \Leftrightarrow \phi'$
$\mathsf{CongrOr2}(p_1)$	$\vdash_{\mathcal{D}}$	$\phi \vee \psi \Leftrightarrow \phi \vee \psi'$	if $p_1 \vdash_{\mathcal{D}} \psi \Leftrightarrow \psi'$
$\mathsf{CongrAnd1}(p_1)$	$\vdash_{\mathcal{D}}$	$\phi \wedge \psi \Leftrightarrow \phi' \wedge \psi$	if $p_1 \vdash_{\mathcal{D}} \phi \Leftrightarrow \phi'$
$\mathsf{CongrAnd2}(p_1)$	$\vdash_{\mathcal{D}}$	$\phi \wedge \psi \Leftrightarrow \phi \wedge \psi'$	if $p_1 \vdash_{\mathcal{D}} \psi \Leftrightarrow \psi'$
$\mathsf{CongrNot}(p_1)$	$\vdash_{\mathcal{D}}$	$\neg\phi \Leftrightarrow \neg\phi'$	if $p_1 \vdash_{\mathcal{D}} \phi \Leftrightarrow \phi'$.

Figure 1: Proof Rules

prove $\phi \wedge F \Leftrightarrow F$. However, the latter proves theorems of the form $\phi \wedge F$ *for any formula* ϕ, while the former only holds for the particular formula ϕ referenced in the disjunct.

Note that it is not the case that all syntactically well-formed proofs prove a theorem. For example, $\mathsf{Trans}(\mathsf{CongrOr1}(p), \mathsf{AndFalse1})$ — clearly, $\mathsf{CongrOr1}(p) \vdash \phi \vee \psi \Leftrightarrow \phi' \vee \psi$ is not compatible with the definition of AndFalse1. We define $\hat{\mathcal{P}}$ as $Dom(\vdash)$, the set of all proofs that prove a theorem of $\mathcal{E}$.

3 Propositional Proof Reduction

Now that we have defined our proof rules as a set of first-order terms, we turn to transformations of these terms. Consider $p \in \hat{\mathcal{P}}$ and $\phi \in \mathcal{S}$ such that $p \vdash \phi \Leftrightarrow T$. Associated with p is a truth assignment whose domain is the set of propositional variables that occur in ϕ. This truth assignment corresponds to a validating clause for ϕ (a model under which the formula simplifies to *true*). It is possible, however, that some of the assignments may be redundant — i.e., a subset of the truth assignment validates the formula. For example, the validating clause of the first proof in Fig. 2 comprises the truth assignment for all propositional variables in $Var(\chi) \cup Var(\psi)$. However, it is clear that the decisions occurring in p_1 are inconsequential because p_2

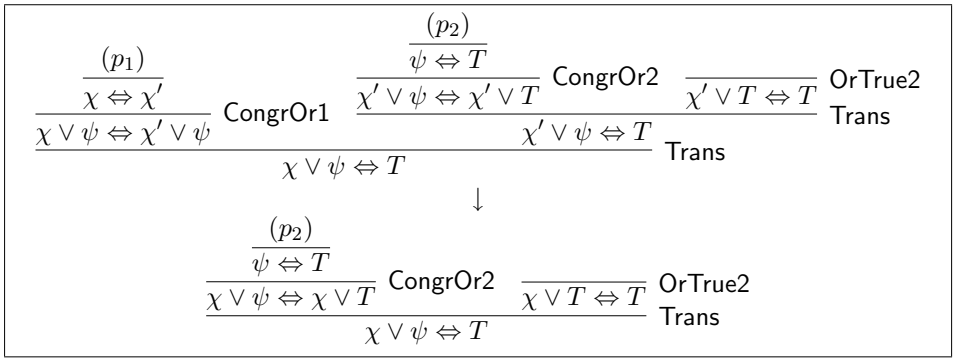

Figure 2: Proof Reduction Example

$\vdash \psi \Leftrightarrow T$ and so only the decisions in p_2 are needed to prove $\chi \vee \psi \Leftrightarrow T$.

We now present an equational theory for removing inconsequential decisions contained within proofs. The basic reduction steps, presented as oriented rewrite rules on the first-order terms of $\hat{\mathcal{P}}$, are given in Fig. 3. These rules are used to transform the proofs from propositional simplification into a canonical form (given in Sec. 3.1) with fewer unnecessary subproofs. E.g., if $p \vdash_{\mathcal{D}} \phi \vee T \Leftrightarrow T$ then any derivation that proves equivalences of ϕ proves a subset of theorems of one that ignores ϕ (an instance of the first **Deriv-Cut** rule). The following lemma expresses the fact that the universal closures of the basic rules in Fig. 3 are sound w.r.t. $\succcurlyeq_{\mathcal{D}}$.

Lemma 1 (Basic Rule Soundness)**.** *For all $p_1, p_2 \in \hat{\mathcal{P}}$, if $p_1 \rightarrow p_2$, then $p_1 \succcurlyeq_{\mathcal{D}} p_2$.*

Using the equational theorem proving tool Waldmeister [4], we can augment our basic rules using Knuth-Bendix completion, resulting in a confluent and terminating (i.e., convergent) rewrite system for reducing proofs. The completed set of 58 rules is given in Sec. A in the appendix. It is straightforward to verify, given the previous lemma and the correctness of the completion procedure, that proof generality holds for reduction w.r.t. the completed set of rewrite rules: for any p_1, p_2 such that p_1 rewrites to p_2 it is the case that p_2 is more general than the p_1. This yields the following theorem, where we write $\xrightarrow{*}$ to denote the reflexive-transitive closure of $\rightarrow$ over the completed TRS.

Theorem 1 (Soundness)**.** *For all $p_1, p_2 \in \hat{\mathcal{P}}$, if $p_1 \xrightarrow{*} p_2$ then $p_1 \succcurlyeq_{\mathcal{D}} p_2$.*

3.1 Canonical Form

We can describe the canonical form of proofs rewritten in this rewrite system as follows. Consider proofs of formulas of the form $\phi \Leftrightarrow T$. If $\phi := \chi \vee \psi$, then the canonical form will prove exactly one disjunct is equivalent to *true*

Right-Assoc
$\mathsf{Trans}(\mathsf{Trans}(x_1, x_2), x_3) \rightarrow \mathsf{Trans}(x_1, \mathsf{Trans}(x_2, x_3))$

Trans-Refl
$\mathsf{Trans}(\mathsf{Refl}, x_1) \rightarrow x_1$
$\mathsf{Trans}(x_1, \mathsf{Refl}) \rightarrow x_1$

Congr-Refl
$\mathsf{CongrOr1}(\mathsf{Refl}) \rightarrow \mathsf{Refl}$
$\mathsf{CongrOr2}(\mathsf{Refl}) \rightarrow \mathsf{Refl}$
$\mathsf{CongrAnd1}(\mathsf{Refl}) \rightarrow \mathsf{Refl}$
$\mathsf{CongrAnd2}(\mathsf{Refl}) \rightarrow \mathsf{Refl}$
$\mathsf{CongrNot}(\mathsf{Refl}) \rightarrow \mathsf{Refl}$

Deriv-Cut
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{OrTrue2}) \rightarrow \mathsf{OrTrue2}$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{OrTrue1}) \rightarrow \mathsf{OrTrue1}$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{AndFalse2}) \rightarrow \mathsf{AndFalse2}$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{AndFalse1}) \rightarrow \mathsf{AndFalse1}$

Congr-Drop
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{OrFalse1}) \rightarrow \mathsf{Trans}(\mathsf{OrFalse1}, x_1)$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{OrFalse2}) \rightarrow \mathsf{Trans}(\mathsf{OrFalse2}, x_1)$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{AndTrue1}) \rightarrow \mathsf{Trans}(\mathsf{AndTrue1}, x_1)$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{AndTrue2}) \rightarrow \mathsf{Trans}(\mathsf{AndTrue2}, x_1)$

Congr-Pull
$\mathsf{Trans}(\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{CongrOr2}(x_2)), \mathsf{Trans}(\mathsf{CongrOr1}(x_3), \mathsf{CongrOr2}(x_4)))$
$\quad \rightarrow \mathsf{Trans}(\mathsf{CongrOr1}(\mathsf{Trans}(x_1, x_3)), \mathsf{CongrOr2}(\mathsf{Trans}(x_2, x_4)))$
$\mathsf{Trans}(\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{CongrAnd2}(x_2)), \mathsf{Trans}(\mathsf{CongrAnd1}(x_3), \mathsf{CongrAnd2}(x_4)))$
$\quad \rightarrow \mathsf{Trans}(\mathsf{CongrAnd1}(\mathsf{Trans}(x_1, x_3)), \mathsf{CongrAnd2}(\mathsf{Trans}(x_2, x_4)))$
$\mathsf{Trans}(\mathsf{CongrNot}(x_1), \mathsf{CongrNot}(x_2)) \rightarrow \mathsf{CongrNot}(\mathsf{Trans}(x_1, x_2))$

Figure 3: Basic Reduction Rules

and use the appropriate cut-off rule to show ϕ is *true* (either OrTrue1 for χ or OrTrue2 for ψ). If $\phi := \chi \wedge \psi$, then the canonical form first has a proof that one side is *true* and then that the other side is *true*, using either AndTrue1 or AndTrue2 (as appropriate) as the only intermediate step. If $\phi := \neg\psi$, then the canonical form has a proof that ψ is *false* and then uses the NotFalse rule, proving $\neg F \Leftrightarrow T$. The form for *false* disjunctions, conjunctions and negations is similar. The canonical form is completely characterized as a context-free grammar in Fig. 4 and Lem. 2 (in the appendix) states that it is a consequence of our rewrite system.

Unfortunately, although the rewrite system is convergent, not all proofs that prove a theorem in common have the same canonical form. Consider the formula $\phi \vee \psi \Leftrightarrow T$, where non-canonical proof p reduces *both* disjuncts

$$
\begin{array}{rcl}
\mathcal{C}_T & ::= & \mathcal{U} \mid \mathcal{C}_{T\vee 1} \mid \mathcal{C}_{T\vee 2} \mid \mathcal{C}_{T\wedge 1} \mid \mathcal{C}_{T\wedge 2} \mid \mathcal{C}_{T\neg} \\
\mathcal{C}_F & ::= & \mathcal{U} \mid \mathcal{C}_{F\vee 1} \mid \mathcal{C}_{F\vee 2} \mid \mathcal{C}_{F\wedge 1} \mid \mathcal{C}_{F\wedge 2} \mid \mathcal{C}_{F\neg} \\
\mathcal{C}_{T\vee 1} & ::= & \mathsf{Trans}(\mathsf{CongrOr1}(\mathcal{C}_T), \mathsf{OrTrue1}) \\
\mathcal{C}_{T\vee 2} & ::= & \mathsf{Trans}(\mathsf{CongrOr2}(\mathcal{C}_T), \mathsf{OrTrue2}) \\
\mathcal{C}_{F\vee 1} & ::= & \mathsf{Trans}(\mathsf{CongrOr1}(\mathcal{C}_F), \mathsf{Trans}(\mathsf{OrFalse1}, \mathcal{C}_F)) \\
\mathcal{C}_{F\vee 2} & ::= & \mathsf{Trans}(\mathsf{CongrOr2}(\mathcal{C}_F), \mathsf{Trans}(\mathsf{OrFalse2}, \mathcal{C}_F)) \\
\mathcal{C}_{F\wedge 1} & ::= & \mathsf{Trans}(\mathsf{CongrAnd1}(\mathcal{C}_F), \mathsf{AndFalse1}) \\
\mathcal{C}_{F\wedge 2} & ::= & \mathsf{Trans}(\mathsf{CongrAnd2}(\mathcal{C}_F), \mathsf{AndFalse2}) \\
\mathcal{C}_{T\wedge 1} & ::= & \mathsf{Trans}(\mathsf{CongrAnd1}(\mathcal{C}_T), \mathsf{Trans}(\mathsf{AndTrue1}, \mathcal{C}_T)) \\
\mathcal{C}_{T\wedge 2} & ::= & \mathsf{Trans}(\mathsf{CongrAnd2}(\mathcal{C}_T), \mathsf{Trans}(\mathsf{AndTrue2}, \mathcal{C}_T)) \\
\mathcal{C}_{T\neg} & ::= & \mathsf{Trans}(\mathsf{CongrNot}(\mathcal{C}_F), \mathsf{NotFalse}) \\
\mathcal{C}_{F\neg} & ::= & \mathsf{Trans}(\mathsf{CongrNot}(\mathcal{C}_T), \mathsf{NotTrue})
\end{array}
$$

Figure 4: Canonical Form of Simplified Proofs

to *true*. In canonical form, one disjunct will be reduced and the other discarded, but the choice is dependent on the exact form of the input proof. A consequence of this is that we cannot prove that the rewrite system returns a proof that uses the minimum number of decisions (i.e., the assignment with the fewest unique literals) — to do so, it would have to choose the disjunct that results in the globally smaller set of decisions.

However, while the rewrite system does not return the minimum number of decisions in instances like the above formula (or one in which both sides of a conjunction reduce to *false*), it will correctly prune portions of the proof for which no "non-deterministic" choice is necessary. If one side of a conjunction reduces to *true* and the other to *false* the rewrite system will always remove the latter derivation. Furthermore, the problem of choosing the disjuncts that result in the minimum-size set of decisions is NP-complete. This is seen via a reduction from VERTEX-COVER in which nodes of an input graph are seen as atomic formulas and the graph itself as a conjunction of disjuncts corresponding to pairs of vertices connected by an edge. The decisions contained in a proof that an assignment implies the resulting formula (and has the smallest number of unique decisions) corresponds to an optimal cover. Our proof reduction system removes the obviously inconsequential portions of the proof and makes an arbitrary choice in the other cases. The authors believe that, in practice, this will result in a useful reduction in the size of the assignment.

4 Conclusion and Future Work

We have presented a sound rewrite system for simplifying propositional equivalence proofs to find small validating clauses. We have described the

canonical form of the proofs and explored the effectiveness of the algebraic approach.

This is a work-in-progress and there are many avenues left to explore. The authors intend to modify the rewrite system in such a way that disjunctions whose disjuncts both simplify to *true* are preserved in the canonical form (i.e., in this case neither side is eliminated). Then, a post-processing choice could be made to find the proof with a globally smaller number of decisions. Although this problem is NP-complete, if a suitable approximation algorithm can be applied then a bound can be given on how much larger the given validating clause is from an optimal one.[1]

References

[1] C. Barrett and J. Donham. Combining SAT Methods with Non-Clausal Decision Heuristics. In S. Ranise and C. Tinelli, editors, *Pragmatics of Decision Procedures in Automated Reasoning*, 2004.

[2] Clark Barrett and Sergey Berezin. CVC Lite: A new implementation of the cooperating validity checker. In R. Alur, editor, *Proceedings of the 16th International Conference on Computer Aided Verification*, 2004.

[3] L. de Moura, H. Rueß, and N. Shankar. Justifying Equality. *Electronic Notes in Theoretical Computer Science*, 125(3):69–85, 2005. Appeared at the 2nd International Workshop on Pragmatics of Decision Procedures in Automated Reasoning (2004).

[4] B. Löchner and Th. Hillenbrand. A phytography of WALDMEISTER. *AI Communications*, 15(2–3):127–133, 2002.

[5] R. Nieuwenhuis and A. Oliveras. Proof-producing Congruence Closure. In J. Giesl, editor, *16th International Conference on Rewriting Techniques and Applications*, pages 453–468. Springer, 2005.

[6] A. Stump, C. Barrett, and D. Dill. CVC: a Cooperating Validity Checker. In *14th International Conference on Computer-Aided Verification*, pages 500–504, 2002.

[7] A. Stump and L.-Y. Tan. The Algebra of Equality Proofs. In Jürgen Giesl, editor, *16th International Conference on Rewriting Techniques and Applications*, pages 469–483. Springer, 2005.

[8] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient Conflict Driven Learning in Boolean Satisfiability Solver. In *International Conference on Computer Aided Design*, pages 279–285, 2001.

[1] Thanks to Joel Brandt and Edwin Westbrook for their help on this project and to the anonymous reviewers for their comments on an early draft.

A Completed Set of Reduction Rules

The rules on the next page are convergent for ordered rewriting using a Knuth-Bendix ordering determined by the following weights and precedences, and the subsequent equations:

```
Trans = 0, OrTrue1 = 1, AndFalse1 = 1, OrTrue2 = 1,
AndFalse2 = 1, OrFalse1 = 1, AndTrue1 = 1, OrFalse2 = 1,
AndTrue2 = 1, CongrNot = 1, CongrOr2 = 1, CongrAnd2 = 1,
CongrOr1 = 1, CongrAnd1 = 1, Refl = 1

Trans > OrTrue1 > AndFalse1 > OrTrue2 > AndFalse2 > OrFalse1 >
  AndTrue1 > OrFalse2 > AndTrue2

AndTrue2 > CongrNot > CongrOr2 > CongrAnd2 > CongrOr1 > CongrAnd1 > Refl
```

$$
\begin{array}{lcl}
\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{CongrOr2}(x2)) & = & \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{CongrOr11}(x_1)) \\
\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{CongrAnd2}(x_2)) & = & \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{CongrAnd1}(x_1)) \\
\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{Trans}(\mathsf{CongrOr2}(x_2), x_3)) & = & \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{Trans}(\mathsf{CongrOr1}(x_1), x_3)) \\
\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), x_3)) & = & \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{Trans}(\mathsf{CongrAnd1}(x_1), x_3))
\end{array}
$$

$\mathsf{Trans}(\mathsf{Refl}, x_1) \to x_1$
$\mathsf{Trans}(x_1, \mathsf{Refl}) \to x_1$
$\mathsf{Trans}(\mathsf{Trans}(x_1, x_2), x_3) \to \mathsf{Trans}(x_1, \mathsf{Trans}(x_2, x_3))$

$\mathsf{CongrAnd1}(\mathsf{Refl}) \to \mathsf{Refl}$
$\mathsf{CongrAnd2}(\mathsf{Refl}) \to \mathsf{Refl}$
$\mathsf{CongrNot}(\mathsf{Refl}) \to \mathsf{Refl}$
$\mathsf{CongrOr1}(\mathsf{Refl}) \to \mathsf{Refl}$
$\mathsf{CongrOr2}(\mathsf{Refl}) \to \mathsf{Refl}$

$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{AndFalse2}) \to \mathsf{AndFalse2}$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{AndTrue2}) \to \mathsf{Trans}(\mathsf{AndTrue2}, x_1)$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{CongrAnd1}(x_2)) \to \mathsf{CongrAnd1}(\mathsf{Trans}(x_1, x_2))$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{Trans}(\mathsf{AndFalse2}, x_2)) \to \mathsf{Trans}(\mathsf{AndFalse2}, x_2)$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{Trans}(\mathsf{AndTrue2}, x_2)) \to \mathsf{Trans}(\mathsf{AndTrue2}, \mathsf{Trans}(x_1, x_2))$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), x_3)) \to \mathsf{Trans}(\mathsf{CongrAnd1}(\mathsf{Trans}(x_1, x_2)), x_3)$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{AndFalse2})) \to \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{AndFalse2})$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{AndTrue2})) \to \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{Trans}(\mathsf{AndTrue2}, x_1))$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{CongrAnd1}(x_3))) \to \mathsf{Trans}(\mathsf{CongrAnd1}(\mathsf{Trans}(x_1, x_3)), \mathsf{CongrAnd2}(x_2))$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{Trans}(\mathsf{AndFalse2}, x_3))) \to \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{Trans}(\mathsf{AndFalse2}, x_3))$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{Trans}(\mathsf{AndTrue2}, x_3))) \to \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{Trans}(\mathsf{AndTrue2}, \mathsf{Trans}(x_1, x_3)))$
$\mathsf{Trans}(\mathsf{CongrAnd1}(x_1), \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), \mathsf{Trans}(\mathsf{CongrAnd1}(x_3), x_4))) \to \mathsf{Trans}(\mathsf{CongrAnd1}(\mathsf{Trans}(x_1, x_3)), \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), x_4))$

$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{AndFalse1}) \to \mathsf{AndFalse1}$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{AndTrue1}) \to \mathsf{Trans}(\mathsf{AndTrue1}, x_1)$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{CongrAnd2}(x_2)) \to \mathsf{CongrAnd2}(\mathsf{Trans}(x_1, x_2))$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{Trans}(\mathsf{AndFalse1}, x_2)) \to \mathsf{Trans}(\mathsf{AndFalse1}, x_2)$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{Trans}(\mathsf{AndTrue1}, x_2)) \to \mathsf{Trans}(\mathsf{AndTrue1}, \mathsf{Trans}(x_1, x_2))$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{AndFalse1})) \to \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{AndFalse1})$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{AndTrue1})) \to \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{Trans}(\mathsf{AndTrue1}, x_1))$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{CongrAnd2}(x_3))) \to \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{CongrAnd2}(\mathsf{Trans}(x_1, x_3)))$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{Trans}(\mathsf{AndFalse1}, x_3))) \to \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{Trans}(\mathsf{AndFalse1}, x_3))$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{Trans}(\mathsf{AndTrue1}, x_3))) \to \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{Trans}(\mathsf{AndTrue1}, \mathsf{Trans}(x_1, x_3)))$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{Trans}(\mathsf{CongrAnd2}(x_3), x_4))) \to \mathsf{Trans}(\mathsf{CongrAnd1}(x_2), \mathsf{Trans}(\mathsf{CongrAnd2}(\mathsf{Trans}(x_1, x_3)), x_4))$
$\mathsf{Trans}(\mathsf{CongrAnd2}(x_1), \mathsf{Trans}(\mathsf{CongrAnd2}(x_2), x_3)) \to \mathsf{Trans}(\mathsf{CongrAnd2}(\mathsf{Trans}(x_1, x_2)), x_3)$

$\mathsf{Trans}(\mathsf{CongrNot}(x_1), \mathsf{CongrNot}(x_2)) \to \mathsf{CongrNot}(\mathsf{Trans}(x_1, x_2))$
$\mathsf{Trans}(\mathsf{CongrNot}(x_1), \mathsf{Trans}(\mathsf{CongrNot}(x_2), x_3)) \to \mathsf{Trans}(\mathsf{CongrNot}(\mathsf{Trans}(x_1, x_2)), x_3)$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{CongrOr1}(x_2)) \to \mathsf{CongrOr1}(\mathsf{Trans}(x_1, x_2))$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{OrFalse2}) \to \mathsf{Trans}(\mathsf{OrFalse2}, x_1)$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{OrTrue2}) \to \mathsf{OrTrue2}$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{Trans}(\mathsf{CongrOr1}(x_2), x_3)) \to \mathsf{Trans}(\mathsf{CongrOr1}(\mathsf{Trans}(x_1, x_2)), x_3)$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{CongrOr1}(x_3))) \to \mathsf{Trans}(\mathsf{CongrOr1}(\mathsf{Trans}(x_1, x_3)), \mathsf{CongrOr2}(x_2))$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{OrFalse2})) \to \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{Trans}(\mathsf{OrFalse2}, x_1))$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{OrTrue2})) \to \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{OrTrue2})$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{Trans}(\mathsf{CongrOr1}(x_3), x_4))) \to \mathsf{Trans}(\mathsf{CongrOr1}(\mathsf{Trans}(x_1, x_3)), \mathsf{Trans}(\mathsf{CongrOr2}(x_2), x_4))$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{Trans}(\mathsf{OrFalse2}, x_3))) \to \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{Trans}(\mathsf{OrFalse2}, \mathsf{Trans}(x_1, x_3)))$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{Trans}(\mathsf{OrTrue2}, x_3))) \to \mathsf{Trans}(\mathsf{CongrOr2}(x_2), \mathsf{Trans}(\mathsf{OrTrue2}, x_3))$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{Trans}(\mathsf{OrFalse2}, x_2)) \to \mathsf{Trans}(\mathsf{OrFalse2}, \mathsf{Trans}(x_1, x_2))$
$\mathsf{Trans}(\mathsf{CongrOr1}(x_1), \mathsf{Trans}(\mathsf{OrTrue2}, x_2)) \to \mathsf{Trans}(\mathsf{OrTrue2}, x_2)$

$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{CongrOr2}(x_2)) \to \mathsf{CongrOr2}(\mathsf{Trans}(x_1, x_2))$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{OrFalse1}) \to \mathsf{Trans}(\mathsf{OrFalse1}, x_1)$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{OrTrue1}) \to \mathsf{OrTrue1}$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{CongrOr2}(x_3))) \to \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{CongrOr2}(\mathsf{Trans}(x_1, x_3)))$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{OrFalse1})) \to \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{Trans}(\mathsf{OrFalse1}, x_1))$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{OrTrue1})) \to \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{OrTrue1})$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{Trans}(\mathsf{CongrOr2}(x_3), x_4))) \to \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{Trans}(\mathsf{CongrOr2}(\mathsf{Trans}(x_1, x_3)), x_4))$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{Trans}(\mathsf{OrFalse1}, x_3))) \to \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{Trans}(\mathsf{OrFalse1}, \mathsf{Trans}(x_1, x_3)))$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{Trans}(\mathsf{OrTrue1}, x_3))) \to \mathsf{Trans}(\mathsf{CongrOr1}(x_2), \mathsf{Trans}(\mathsf{OrTrue1}, x_3))$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{Trans}(\mathsf{CongrOr2}(x_2), x_3)) \to \mathsf{Trans}(\mathsf{CongrOr2}(\mathsf{Trans}(x_1, x_2)), x_3)$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{Trans}(\mathsf{OrFalse1}, x_2)) \to \mathsf{Trans}(\mathsf{OrFalse1}, \mathsf{Trans}(x_1, x_2))$
$\mathsf{Trans}(\mathsf{CongrOr2}(x_1), \mathsf{Trans}(\mathsf{OrTrue1}, x_2)) \to \mathsf{Trans}(\mathsf{OrTrue1}, x_2)$