

The Algebra of Equality Proofs

Aaron Stump and Li-Yang Tan

Dept. of Computer Science and Engineering
Washington University in St. Louis
St. Louis, Missouri, USA
Web: <http://cl.cse.wustl.edu/>

Abstract. Proofs of equalities may be built from assumptions using proof rules for reflexivity, symmetry, and transitivity. Reflexivity is an axiom proving $x=x$ for any x ; symmetry is a 1-premise rule taking a proof of $x=y$ and returning a proof of $y=x$; and transitivity is a 2-premise rule taking proofs of $x=y$ and $y=z$, and returning a proof of $x=z$. Define an equivalence relation to hold between proofs iff they prove a theorem in common. The main theoretical result of the paper is that if all assumptions are independent, this equivalence relation is axiomatized by the standard axioms of group theory: reflexivity is the unit of the group, symmetry is the inverse, and transitivity is the multiplication. Using a standard completion of the group axioms, we obtain a rewrite system which puts equality proofs into canonical form. Proofs in this canonical form use the fewest possible assumptions, and a proof can be canonized in linear time using a simple strategy. This result is applied to obtain a simple extension of the union-find algorithm for ground equational reasoning which produces minimal proofs. The time complexity of the original union-find operations is preserved, and minimal proofs are produced in worst-case time $O(n^{\log_2 3})$, where n is the number of expressions being equated. As a second application, the approach is used to achieve significant performance improvements for the CVC cooperating decision procedure. [Submitted to RTA 2005.]

1 Introduction

Ground equational reasoning plays an important role in many approaches to verification and automated reasoning [8, 13, 17, 12, 3, 14, 9]. Recently, there has been interest in producing minimal proofs from algorithms for ground equational reasoning [11, 10, 5]. Minimal proofs are of interest primarily for performance reasons: they can be exported to a fast SAT solver as conflict clauses, which greatly improve search space pruning [1].

In this paper, we approach the problem of minimal proofs by studying the algebra of equality proofs themselves (Section 2). It turns out that theorem equivalence of equality proofs with independent assumptions is completely characterized by the axioms for free groups (Sections 3, 4, and 5). This enables us to use a standard convergent rewrite system for free group terms to put equality proofs into canonical form (Section 5). If all assumptions are independent, this

form is minimal, in the sense that it uses the unique minimal set of assumptions needed to prove the equality. We analyze the number of steps required for canonization using the rewrite rules. A simple strategy yields canonization in time linear in the size of the equality proof (Section 6), although without a strategy canonization can take cubic time (Section 7). We then show how these results can be used to obtain minimal proofs of equations $x = y$ from a simple augmentation of the standard union-find algorithm for ground equational reasoning in $(O(n^{\log_2 3}))$ time, where n is the size of the equivalence class for x and y (Section 8). Finally, major improvements in search space pruning and overall performance are obtained in the context of the CVC tool [16] using the algebra of equality proofs (Section 9).

2 Equality Proofs

Notation: If f is an n -ary function symbol and A_1, \dots, A_n are sets of terms, we write $f(A_1, \dots, A_n)$ for $\{f(t_1, \dots, t_n) \mid t_1 \in A_1, \dots, t_n \in A_n\}$.

Let \mathcal{U} be a set of *assumptions*, and let \mathcal{P} be the set of *equality proofs* inductively defined as follows, where *Refl*, *Symm*, and *Trans* are function symbols of arities 0, 1, and 2, respectively:

$$\mathcal{P} ::= \mathcal{U} \mid \text{Refl} \mid \text{Symm}(\mathcal{P}) \mid \text{Trans}(\mathcal{P}, \mathcal{P})$$

Let \mathcal{A} be a set of *atoms*, and let \mathcal{E} be the set of all equations between atoms. Let $::$ be a total function from \mathcal{U} to \mathcal{E} , specifying which equation is proved by each assumption. Extend this to a non-functional relation on all of $\mathcal{P} \times \mathcal{E}$, which says which theorems are proved by which equality proofs; this is done inductively by the universal closures of the following clauses:

$$\begin{aligned} \text{Refl} &:: a = a \\ p &:: a_1 = a_2 \Rightarrow \text{Symm}(p) :: a_2 = a_1 \\ p_1 &:: a_1 = a_2 \wedge p_2 :: a_2 = a_3 \Rightarrow \text{Trans}(p_1, p_2) :: a_1 = a_3 \end{aligned}$$

We say that the assumptions in \mathcal{U} are *independent* iff no equation proved by an assumption is equationally entailed by the other assumptions. For example, assumptions of $a = b$, $b = c$, and $a = c$, respectively, are not independent.

Finally, define a relation $==$ of *theorem equivalence* on $\mathcal{P} \times \mathcal{P}$ by the universal closure of

$$p_1 == p_2 \Leftrightarrow (\exists e. p_1 :: e \wedge p_2 :: e)$$

Two equality proofs are in this relation iff they prove a theorem in common. For example, $\text{Trans}(\text{Refl}, \text{Refl}) == \text{Refl}$, because each proof proves $a = a$, for some $a \in \mathcal{A}$. It turns out that technical reasons prevent us from adopting the stronger notion where proofs are equivalent iff they prove exactly the same theorems. Not every equality proof proves a theorem. Write \equiv for meta-equality on sets like \mathcal{A} , \mathcal{E} , and \mathcal{P} . Then, for example, if $p \in \mathcal{P}$ proves $a = b$ with $a \not\equiv b$, then $\text{Trans}(p, p)$ does not prove any theorem. The following lemmas help justify the use of the notion of proving a theorem in common as our notion of theorem equivalence. We define $\hat{\mathcal{P}}$ to be $\text{Dom}>::$, so that $p \in \hat{\mathcal{P}}$ iff p proves a theorem.

Lemma 1 *If $p \in \hat{\mathcal{P}}$ does not contain any assumptions, then $\forall a \in \mathcal{A}. p :: a = a$. Further, if $a \neq b$, then p does not prove $a = b$.*

Lemma 2 *If $p \in \hat{\mathcal{P}}$ contains an assumption, then p proves exactly one theorem.*

3 Algebraic Characterization of Theorem Equivalence

In this Section, we show that theorem equivalence on proofs which prove a theorem is axiomatized by the standard axioms of group theory, as long as the assumptions in \mathcal{U} are independent. In more detail, for all $p_1, p_2 \in \hat{\mathcal{P}}$, $p_1 == p_2$ holds iff the equation $p_1 \cong p_2$ is provable using standard congruence rules for equality and the universal closures of the formulas given in Figure 1. For comparison, the group axioms are given in more customary form in Figure 2. *Refl* plays the role of the group unit, *Symm* is the inverse operator, and *Trans* is the multiplication.

$$\begin{array}{lll} [\textit{Associativity}] & \textit{Trans}(\textit{Trans}(p_1, p_2), p_3) & \cong \textit{Trans}(p_1, \textit{Trans}(p_2, p_3)) \\ [\textit{Unit}] & \textit{Trans}(\textit{Refl}, p) & \cong p \\ [\textit{Inverse}] & \textit{Trans}(\textit{Symm}(p), p) & \cong \textit{Refl} \end{array}$$

Fig. 1. Group Axioms for Equality Proofs

$$\begin{array}{lll} [\textit{Associativity}] & (x_1 * x_2) * x_3 & \cong x_1 * (x_2 * x_3) \\ [\textit{Unit}] & 1 * x & \cong x \\ [\textit{Inverse}] & x^{-1} * x & \cong 1 \end{array}$$

Fig. 2. Group Axioms

We observe first that strictly speaking, neither \mathcal{P} nor $\hat{\mathcal{P}}$ forms a group with operators *Refl*, *Symm*, and *Trans* and equivalence relation $==$. For the case of \mathcal{P} , this is because if p is an equality proof which does not prove any theorem, then the left hand side (lhs) of the [*Inverse*] axiom of Figure 1 does not prove any theorem, but the rhs (*Refl*) does. Hence, the two proofs do not prove the same theorems, so the axiom is just false (if $==$ is taken for \cong) for domain \mathcal{P} . We prove below that all the axioms are sound with respect to $==$ for $\hat{\mathcal{P}}$, but $\hat{\mathcal{P}}$ is not closed in general under *Trans*. As noted above, if $p :: a = b$ with $a \neq b$, then $\textit{Trans}(p, p)$ is not in $\hat{\mathcal{P}}$. So, $\hat{\mathcal{P}}$ does not form a group under *Refl*, *Symm*, and *Trans*. Nevertheless, we have the following results, which are proved in Sections 4 and 5 below.

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

Theorem 2 (Completeness) *For all $p_1, p_2 \in \hat{\mathcal{P}}$, if $p_1 == p_2$, then $p_1 \cong p_2$.*

4 Proof of Soundness

Lemma 3 (Theorem Determinacy) *Suppose $p \in \hat{\mathcal{P}}$, $p :: a_1 = a_2$ and $p :: b_1 = b_2$. Then $a_1 \equiv b_1$ iff $a_2 \equiv b_2$.*

Proof. If p contains no assumption, then by Lemma 1, $a_1 \equiv a_2$ and $b_1 \equiv b_2$, and we have the result by transitivity of \equiv . If p contains an assumption, then by Lemma 2, p proves exactly one theorem, and so $a_1 \equiv b_1$ and $a_2 \equiv b_2$.

Proof (Soundness (Theorem 1)). First we observe that the congruence rules for \cong are sound. If $p_1 == p_2$, then $Symm(p_1) == Symm(p_2)$. For the congruence rule for $Trans$, we reason as follows. Suppose one of p_1 or p_2 contains assumptions. WLOG, say it is p_1 . Then by Lemma 2, both p_1 and $Trans(p_1, p)$ prove at most one theorem, for any p . If the latter proves no theorem, we contradict the hypothesis of Theorem 1 that the proofs in question prove a theorem. If $Trans(p_1, p)$ proves a theorem, then since p_2 proves a theorem in common with p_1 , p_2 must prove the same theorem as p_1 , and hence $Trans(p_2, p)$ proves a theorem in common with $Trans(p_1, p)$. If neither p_1 nor p_2 contains an assumption, then by Lemma 1, they prove all the same theorems, and hence $Trans(p_1, p)$ and $Trans(p_2, p)$ prove a theorem in common, assuming again that they prove any theorem at all. We now consider the group axioms for \cong .

Case [Associativity]: Suppose $Trans(Trans(p_1, p_2), p_3) :: a = d$. By the definition of $::$, this implies that there is a c such that $Trans(p_1, p_2) :: a = c$ and $p_3 :: c = d$. The former fact implies again by the definition of $::$ that there is a b such that $p_1 :: a = b$ and $p_2 :: b = c$. Then clearly $Trans(p_1, Trans(p_2, p_3))$ proves $a = d$.

Case [Unit]: If $Trans(Refl, p)$ proves $a = b$, then p must prove $a = b$.

Case [Inverse]: If $Trans(Symm(p), p)$ proves $a = c$, then there must be a b such that $Symm(p) :: a = b$ and $p :: b = c$. The former consequence implies that $p :: b = a$. By Lemma 3, $a \equiv c$, so $Trans(Symm(p), p) :: a = a$. We also have, of course, $Refl :: a = a$. Note that this is the point at which defining theorem equivalence as proving exactly the same theorems breaks down. For we also have, e.g., $Refl :: b = b$, but by Lemma 1, if p contains an assumption, $Trans(Symm(p), p)$ cannot prove two theorems. And hence, since it proves $a = a$, it cannot prove $b = b$. So this axiom would not be sound with the stronger version of theorem equivalence.

5 Canonical Proofs and Completeness

The proof of our completeness theorem (Theorem 2) relies on the canonical forms for equality proofs. Recall that the rewrite rules of Figure 3 are a convergent completion of the group axioms of Figure 2, oriented from left to right [7]. These rules are given again in Figure 4, formulated for equality proofs.

$$\begin{aligned}
(x_1 * x_2) * x_3 &\rightarrow x_1 * (x_2 * x_3) \\
1 * x &\rightarrow x \\
x * 1 &\rightarrow x \\
x^{-1} * x &\rightarrow 1 \\
x * x^{-1} &\rightarrow 1 \\
1^{-1} &\rightarrow 1 \\
(x^{-1})^{-1} &\rightarrow x \\
(x * y)^{-1} &\rightarrow y^{-1} * x^{-1} \\
x^{-1} * (x * y) &\rightarrow y \\
x * (x^{-1} * y) &\rightarrow y
\end{aligned}$$

Fig. 3. Convergent System for Simplifying Group Terms

$$\begin{aligned}
\text{Trans}(\text{Trans}(p_1, p_2), p_3) &\rightarrow \text{Trans}(p_1, \text{Trans}(p_2, p_3)) \\
\text{Trans}(\text{Refl}, p) &\rightarrow p \\
\text{Trans}(p, \text{Refl}) &\rightarrow p \\
\text{Trans}(\text{Symm}(p), p) &\rightarrow \text{Refl} \\
\text{Trans}(p, \text{Symm}(p)) &\rightarrow \text{Refl} \\
\text{Symm}(\text{Refl}) &\rightarrow \text{Refl} \\
\text{Symm}(\text{Symm}(p)) &\rightarrow p \\
\text{Symm}(\text{Trans}(p_1, p_2)) &\rightarrow \text{Trans}(\text{Symm}(p_2), \text{Symm}(p_1)) \\
\text{Trans}(\text{Symm}(p_1), \text{Trans}(p_1, p_2)) &\rightarrow p_2 \\
\text{Trans}(p_1, \text{Trans}(\text{Symm}(p_1), p_2)) &\rightarrow p_2
\end{aligned}$$

Fig. 4. Convergent System for Canonizing Proofs

Theorem 3 (Canonical Proofs) *Suppose an equality proof p is in canonical form with respect to the rewrite system of Figure 4. Then p is either Refl or in the set \mathcal{C} inductively defined by:*

$$\mathcal{C} ::= \mathcal{U} \mid \text{Symm}(\mathcal{U}) \mid \text{Trans}(\mathcal{U}, \mathcal{C}) \mid \text{Trans}(\text{Symm}(\mathcal{U}), \mathcal{C})$$

Furthermore, no assumption is used twice in p , and if $p \in \mathcal{C}$, then there is no $a \in \mathcal{A}$ such that $p :: a = a$.

Proof. The proof is by induction on the form of p . If p is Refl or in \mathcal{U} , it is clearly in \mathcal{C} and satisfies the condition on assumptions. It does not prove any equation of the form $a = a$, by independence of assumptions. If $p \equiv \text{Symm}(p_1)$ for some p_1 , then by IH, $p_1 \in \mathcal{C}$. We cannot have $p_1 \equiv \text{Symm}(p_2)$, since then p is not canonical. For the same reason, we cannot have $p_1 \equiv \text{Trans}(p_2, p_3)$ or $p_1 \equiv \text{Refl}$. The only possibility is that $p_1 \in \mathcal{U}$, which shows that $p \in \mathcal{C}$. This also implies that p does not prove any equations of the form $a = a$, since p_1 does not by independence of assumptions. The condition on assumptions is clearly satisfied, since p contains a single assumption.

Finally, if $p \equiv \text{Trans}(p_1, p_2)$ for some p_1, p_2 , then by IH we may assume $p_1, p_2 \in \mathcal{C}$; p is not canonical if one of p_1 or p_2 is Refl . We also cannot have

$p_1 \equiv \text{Trans}(p_3, p_4)$, since p would not be canonical. Similar considerations show that the only possibilities are $p_1 \equiv u \in \mathcal{U}$ or $p_1 \equiv \text{Symm}(u) \in \text{Symm}(\mathcal{U})$. This shows $p \in \mathcal{C}$. To show that the condition on assumptions is satisfied by p , we have by IH that it is satisfied by p_2 . It now suffices to show that u cannot occur in p_2 . Suppose $p_1 :: a = b$ for some $a, b \in \mathcal{A}$. Since p_1 contains an assumption, a and b are, in fact, the unique atoms such that $p_1 :: a = b$. Suppose u occurs in p_2 . Since $p_2 \in \mathcal{C}$, this means that there is a sequence L_1, \dots, L_n of proofs in $\mathcal{U} \cup \text{Symm}(\mathcal{U})$ such that $p_2 \equiv \text{Trans}(L_1, \text{Trans}(\dots, \text{Trans}(L_n, q)))$ for some $q \in \mathcal{C}$; where $L_n \equiv u$ or $L_n \equiv \text{Symm}(u)$. Suppose $L_n \equiv p_1$. If $n = 1$, then the only way p ($\equiv \text{Trans}(p_1, \text{Trans}(p_1, q))$) can prove a theorem is if $a \equiv b$, which contradicts independence of assumptions. If $n > 1$, then $\text{Trans}(L_1, \text{Trans}(\dots, \text{Trans}(L_{n-2}, L_{n-1}))) :: b = a$. This contradicts independence, since this latter term and L_n do not contain the same assumption by IH. Suppose now that $L_n \not\equiv p_1$, and hence, $L_n :: b = a$. Suppose $n > 1$. We have $\text{Trans}(L_1, \text{Trans}(\dots, \text{Trans}(L_{n-2}, L_{n-1}))) :: b = b$. If $n = 2$ then $L_{n-1} :: b = b$, contradicting independence of assumptions. Otherwise, $\text{Trans}(L_1, \text{Trans}(\dots, L_{n-2})) :: b = x$ for some x , and $L_{n-1} :: x = b$. This again contradicts independence of assumptions. A similar argument shows that p does not prove any equation of the form $b = b$. If $n = 1$, then $p \equiv \text{Trans}(p_1, \text{Trans}(L_1, q))$, where either $p_1 = \text{Symm}(L_1)$ or $L_1 = \text{Symm}(p_1)$. In either case, p is not in canonical form.

Completeness now follows easily:

Proof (Completeness (Theorem 2)). Assuming $p_1, p_2 \in \hat{\mathcal{P}}$ prove a theorem in common, we must show $p_1 \cong p_2$. By Soundness (Theorem 1), we may assume p_1 and p_2 are in the canonical form of Theorem 3. Suppose $p_1 \not\equiv p_2$. Then neither one can be *Refl*, since by Theorem 3, *Refl* is the only proof in that canonical form which proves an equation of the form $a = a$. So $p_1, p_2 \in \mathcal{C}$. Since p_1 and p_2 both prove a theorem and both contain an assumption, by Lemma 2, we may suppose they both prove just $a = b$ (where $a \not\equiv b$). Suppose $p_2 \in \mathcal{U} \cup \text{Symm}(\mathcal{U})$. Then independence of assumptions is violated, since $p_1 :: a = b$ but $p_1 \not\equiv p_2$. Suppose $p_2 \equiv \text{Trans}(L, \text{Trans}(L_1, \dots, L_n))$. Then $L :: a = x$ and $\text{Trans}(L_1, \dots, L_n) :: x = b$ for some x . Hence,

$$\text{Symm}(\text{Trans}(\text{Trans}(L_1, \dots, L_n), \text{Symm}(p_1))) :: a = x$$

which again contradicts independence of assumptions (since $L \in \mathcal{U} \cup \text{Symm}(\mathcal{U})$ also proves $a = x$).

6 A Linear-Time Strategy for Canonizing Proofs

In this Section, we show that if the rules of Figure 4 are applied with a leftmost outermost strategy, a given equality proof can be canonized in a number of steps linear in its size. Let **Collapse** be the set of all rewrite rules from the Figure which have either a variable or a constant on the rhs. We call the remaining two rules *RAssoc* and *InverseIn*, respectively:

$$\begin{aligned} \text{Trans}(\text{Trans}(p_1, p_2), p_3) &\rightarrow \text{Trans}(p_1, \text{Trans}(p_2, p_3)) \\ \text{Symm}(\text{Trans}(p_1, p_2)) &\rightarrow \text{Trans}(\text{Symm}(p_2), \text{Symm}(p_1)) \end{aligned}$$

Define the *right-linear spine* of a proof p to be the longest position π of the form 1^* such that $p|_{\pi'}$ is a *Trans* expression, for every prefix π' of π . The internal nodes of a proof are those of its subexpressions that are *Trans* or *Symm* expressions. A node p' is on the right-linear spine π of a proof p if there is a prefix π' of π such $p|_{\pi'} \equiv p'$. Now the leftmost outermost strategy decreases the measure $(m_1(p), m_2(p), m_3(p))$ in the lexicographic combination of the usual arithmetic ordering, where the components are:

- $m_1(p)$ The number of subterms occurring in some subterm of p of the form $\text{Symm}(p')$.
- $m_2(p)$ The size (denoted $|p|$) of p .
- $m_3(p)$ The number of internal nodes not on the right-linear spine of p .

Note that the whole measure is bounded above by $(|p|, |p|, |p|)$. We now show that every rule decreases this measure. *InverseIn* decreases m_1 . This is because we are using an outermost strategy, and so if *InverseIn* applies to a subterm, that subterm is not itself contained in a *Symm* node at a shorter position in p . No other rule increases m_1 , no matter what strategy is used. Note that when *InverseIn* decreases m_1 , it causes m_2 to increase by 1. No other rule can increase m_2 , and the **Collapse** rules all decrease it, without increasing m_3 . This is again true no matter what strategy is used. The rule *RAssoc* decreases m_3 . This is so again because we are using an outermost strategy. If *RAssoc* applies to a subterm of p , that subterm must actually occur on the right-linear spine of p . If not, it is either immediately beneath some *Symm* node, in which case *InverseIn* applies; or else it is the left child of some other *Trans* node. That other *Trans* node cannot be on the right-linear spine, since if it were, *RAssoc* would have been applied to it using our leftmost outermost strategy. An inductive argument based on similar reasoning shows that other *Trans* node cannot be off the right-linear spine. Hence, the original subterm we considered must have indeed been on the right-linear spine. And then the length of that spine is increased by one by applying *RAssoc*. We can then bound the number of steps to canonize p above by $4|p|$. This is so since each component is decreased linearly from an amount bounded by $|p|$, with only the linear increase in m_2 caused by applying *InverseIn* to account for additionally.

7 Canonizing Proofs without a Strategy

In this Section, we analyze the complexity of canonization if no strategy is used. Canonization of term p can take time at least cubic in $|p|$. To see this, first observe that right associating a left-associated term of size n takes $O(n^2)$ time if a leftmost innermost strategy is used. This is because with an innermost strategy, each assumption starting with the third one from the left must be pushed past

all the assumptions to its left. This takes $\sum_{i=3}^n i = O(n^2)$ time. Now consider the following example

$$\text{Symm}_1(\dots(\text{Symm}_{\frac{n}{2}}(\text{Trans}(\dots(\text{Trans}(a_1, a_2), a_3), \dots), a_{\frac{n}{2}}))\dots)$$

where $a_1, \dots, a_{\frac{n}{2}}$ are assumptions. For each instance of the inverse operator Symm , it obviously takes (at least) linear time to distribute it inwards to the innermost positions. Since InverseIn swaps the positions of its arguments, each distribution of Symm into a proof in completely right associated form results in a completely left-associated form. The proof then has to be put into right associated form again. If we alternate pushing inverses in and right associating fully left associated terms, the overall time complexity is $O(n^3)$. The following theorem shows that this is, in fact, the worst case.

Theorem 4 (Analysis with no strategy) *It takes $O(|p|^3)$ steps to normalize a proof p using the rules of Figure 4 (without a fixed strategy).*

Proof. Let $\text{TransPos}(p)$ be the set of positions in p at which there is a Trans operator. Let $\text{lefts}(\pi)$ be the number of 0s in position π , and let $\text{invs_above}(\pi, p)$ be the number of prefixes of π at which p has a Symm operator. Now define the following three measures:

$$\begin{aligned} m_1(p) &= \sum_{\pi \in \text{TransPos}(p)} \text{invs_above}(\pi, p), \\ m_2(p) &= \sum_{\pi \in \text{TransPos}(p)} \text{lefts}(\pi), \\ m_3(p) &= |p| \end{aligned}$$

The claim is that each rule reduces the measure

$$m(p) = (m_1(p), m_2(p), m_3(p))$$

in the lexicographic combination of the usual less-than relation on natural numbers. **Collapse** rules clearly reduce m_3 , and can readily be seen to preserve m_1 and m_2 . InverseIn reduces m_1 , and RAssoc maintains m_1 while reducing m_2 .

We obtain a bound of $2|p|^3$ for the number of rewrite steps to normalize p . This is done using a more refined analysis of the changes to the measure m , presented by the table of Figure 5. Each row bounds the effect of a rule or rules on $m(p)$, for an arbitrary proof p , by showing the worst-case (slowest decrease) change on the measure when p is rewritten to some p' . In the worst case, RAssoc rewrites $\text{Trans}(\text{Trans}(p_1, p_2), p_3)$ to $\text{Trans}(p_1, \text{Trans}(p_2, p_3))$ where p_1 is an assumption. This is the worst case because p_1 then contributes nothing to $m_2(p)$, since its position is not in $\text{TransPos}(p)$. If p_1 contained a Trans node, then m_2 would decrease by more than 1. As it is, the only decrease to $m_2(p)$ is due to the fact that the Trans node at position 1 in p' is to the left of one fewer Trans nodes than the node at position 0 in p . The worst case shown by Figure 5 for InverseIn occurs when the rule is applied at the top position of $\text{Symm}(\text{Trans}(p_1, p_2))$, where p_1 is an assumption and p_2 is hence a term of size $|p| - 3$. In this case, m_2 increases by $|p| - 3$, because p_2 occurs to the left of one more Trans node in the resulting term than in the original term p .

Rule(s)	Starting measure	Ending measure
<i>Collapse</i>	(m_1, m_2, m_3)	$(m_1, m_2, m_3 - 1)$
<i>RAssoc</i>	(m_1, m_2, m_3)	$(m_1, m_2 - 1, m_3)$
<i>InverseIn</i>	(m_1, m_2, m_3)	$(m_1 - 1, m_2 + p - 3, m_3 + 1)$

Fig. 5. How the rules change the measure $m(p)$ of a proof p

To obtain the $2|p|^3$ bound, we next observe that $m_1(p)$ and $m_2(p)$ are both bounded by $|p|^2$ for any p . Each use of *InverseIn* results in a decrease by 1 in m_1 and an increase by $|p| - 3$ in m_2 . To offset the latter increase, *RAssoc* will have to be used some number of times bounded by $|p|$. Hence, the overall number of steps is bounded by the sum of $|p|^2$ for the initial value of m_2 ; $|p|^3$ to reduce all the additions to m_2 caused by reducing m_1 ($|p|$ for each reduction to m_1 , which is bounded by $|p|^2$); $|p|^2$ to offset the additions to m_3 incurred by *InverseIn*; and $|p|$ for the initial value of m_3 . For any p with $|p| > 2$, this sum is bounded above by $2|p|^3$. (The only reducible term of size less than or equal to 2 is *Symm(Refl)*, which reduces to *Refl* in just one step.)

8 Minimal Proofs from Union-Find

This Section presents an approach to producing minimal proofs from the well-known union-find algorithm [4, Chapter 22]. Proofs are minimal in the sense that they use the unique minimal subset of independent assumptions from which a given equation can be derived. Recall that union-find maintains equivalence classes of atoms in balanced, lazily path-compressed trees. Each atom has an associated *find pointer* which points towards the root of its tree. Roots of trees have null find pointers.

We obtain minimal proofs from union-find by first instrumenting the code for union and find to maintain proofs (cf. [15, Chapter 5]). Unioning the equivalence classes for atoms x and y requires a proof that $x = y$. Here, such proofs are just assumptions from \mathcal{U} . Finding the representative y of x 's equivalence class produces a proof that $x = y$. Each non-null find pointer has an associated proof. We maintain the invariant that if x 's find pointer points to y , then the associated proof is a proof of $x = y$. This invariant is maintained as illustrated in Figures 6 and 7. Find pointers are denoted with solid arrows, and the associated proofs are written (using the compact group notation) next to them. In Figure 7, the dotted arrow is for an assumption given to union. The proof produced for a call to find for atom x is just the proof associated with x 's find pointer after path compression has modified it to point directly to the root of x 's tree. If x is the root of its tree, the proof is just *Refl*.

On top of proof-producing union and find, we define a check function, that checks whether or not atoms x and y are equal under the assumptions given to union. If they are equal, the function produces a minimal proof of $x = y$. The implementation is as follows. We call find on x and y . If they have the

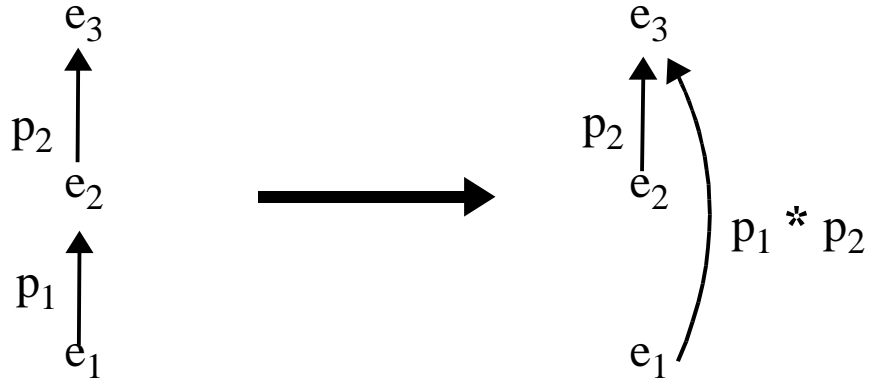


Fig. 6. Maintaining proofs during path compression.

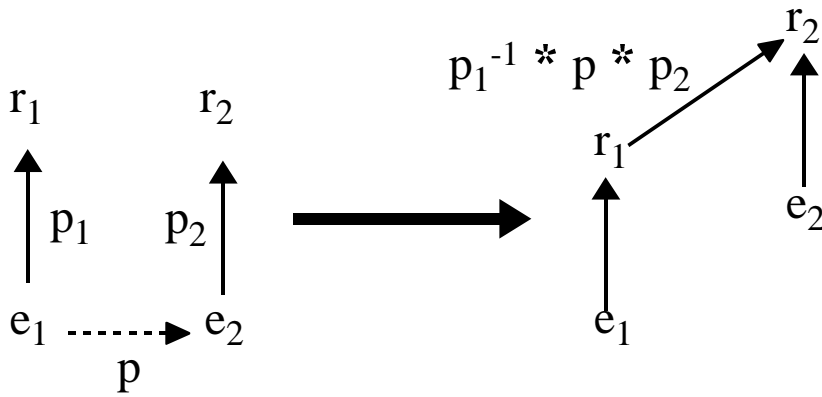


Fig. 7. Maintaining proofs during unions.

same representative z , then `find` produces proofs p_1 and p_2 of $x = z$ and $y = z$, respectively. To compute the minimal proof that $x = y$, we simply canonize the proof $\text{Trans}(p_1, \text{Symm}(p_2))$ using the rules of Figure 4. Since we can canonize a proof using the strategy of Section 6 in time linear in the proof's size, it suffices to bound the size of $\text{Trans}(p_1, \text{Symm}(p_2))$ to bound the additional time needed for producing the minimal proof.

We bound the size of proofs returned by `find` as follows. Define the *proof to the root* for a non-root atom x in a union-find tree to be $\text{Trans}(p_1, \text{Trans}(p_2, \dots, p_n))$, where p_1, \dots, p_n are the proofs associated with the find pointers on the path from x to the root of the tree. Clearly the size of the proof to the root for x is the size of the proof returned by `find` for x . Suppose that $T(n)$ is a bound on the size of the biggest proof to the root in a tree of size n maintained by union-find. Clearly the size of the biggest proof to the root is not affected by path compression. So consider the effect of doing a union. The worst case is when two trees of equal

size n are merged. The size of the new tree is $2 * n + 1$. The proof associated with the new find pointer (see Figure 7) is clearly bounded by $2 * T(n) + 4$, since it consists of one *Symm* node, two *Trans* nodes, an assumption, and two proofs to the roots of the merged trees. The size of the biggest proof to the root in the resulting tree is hence $3 * T(n) + 4$. This is because all the paths in one of the merged trees have been augmented by a find pointer whose associated proof is of size $2 * T(n) + 4$. So $T(n)$ must satisfy $T(2 * n + 1) = 3 * T(n) + 4$. Textbook techniques yield a solution to this recurrence of $T(n) = O(n^{\log_2 3})$. In the worst case, when the size of the minimal proof is $O(n)$, this result is quite close to the result of $O(n \log n)$ obtained in [10]. It must be noted, however, that in that work, minimal proofs of size k are obtained in $O(k \log n)$ time, which is clearly better than the bound obtained here, if $k \ll n$ or n is very large. The advantage of the approach presented here is its simplicity: the cited work requires rather subtle additions to union-find to compute minimal proofs.

9 Application to Cooperating Decision Procedures

In this Section, the above ideas are applied in the context of the CVC (“Cooperating Validity Checker”) system to obtain major performance improvements on benchmark formulas from hardware verification [16]. In review, CVC and similar tools like CVC Lite (CVC’s successor) and ICS separate boolean reasoning from theory-specific reasoning [2, 6]. A fast propositional SAT solver tries to find an assignment to the propositional skeleton of the input formula, or possibly to an equisatisfiable CNF version. Either at each step as the assignment is generated or once it is found, cooperating decision procedures (DPs) are consulted about the consistency of the assignment. For example, a large formula might contain many equalities or other interpreted atomic formulas between ground terms. The SAT solver chooses an assignment to some of those formulas which makes the goal formula satisfiable, if the meanings of the interpreted predicates are ignored. The cooperating DPs then determine if that assignment is consistent with the meanings of the interpreted predicates. If not, a subset of the assignment is identified as inconsistent and returned to the SAT solver as a *conflict clause*. Conflict clauses are maintained during subsequent search for a satisfying assignment, and can greatly prune the search space [1]. Smaller conflict clauses are always more effective than their supersets at pruning the search space.

CVC leverages its infrastructure for generating proofs to track assumptions. The basic idea is that when the cooperating DPs discover that an assignment proposed by the SAT solver is inconsistent, a subset of the assumptions used in that assignment can be determined by inspecting an explicit proof of the contradiction. Such proofs are generated by CVC’s DPs. In its fastest mode before the present work, the DPs generate not a full proof, but an *abstract proof* consisting just of the assumptions that would have appeared in the full proof [1]. This greatly reduces the time required to manipulate proofs and extract conflict clauses.

For the first experiments reported in this Section, CVC was modified to canonize equality proofs according to the linear-time strategy of Section 6. Note that this requires full proofs instead of abstract proofs. Each time CVC's DPs try to build an equality proof, that proof is put in canonical form. It turns out that an additional transformation on proofs is required to get significant benefits for CVC's equational reasoning. CVC's congruence closure algorithm rewrites asserted disequalities each time one of the sides is asserted equal to something else. The modified disequality is then asserted. The resulting proofs of contradictions turn out often to involve subproofs of the following form:

$$\frac{a = b \quad \frac{a = c \quad b = d}{(a = b) \Leftrightarrow (c = d)} \text{SubstEquiv}}{c = d} \text{EquivMP}$$

Such subproofs are rewritten to ones of the following form in order to take advantage of the canonization algorithm:

$$\frac{\frac{a = c}{c = a} \text{Symm} \quad \frac{a = b \quad b = d}{a = d} \text{Trans}}{c = d} \text{Trans}$$

Algebraically, this corresponds to adding the following rewrite rule to the rules of Figure 4:

$$\text{EquivMP}(p_1, \text{SubstEq}(p_2, p_3)) \rightarrow \text{Trans}(\text{Symm}(p_2), \text{Trans}(p_1, p_3))$$

Adding this rule to those of Figure 4 leads to no new critical pairs, and the resulting system is obviously still terminating. Hence, it remains convergent. Leftmost outermost application is readily seen to remain linear time.

Figure 8 compares the number of *decisions* (the number of times a value was chosen for an atomic formula in a propositional assignment) and wallclock time on 6 benchmark formulas from hardware verification, using the approach with abstract proofs and the approach with canonized full proofs. The sizes of the benchmark formulas themselves in ASCII text are also listed. We see that canonization reduces the number of decisions from anywhere from 15% to 65% on these benchmarks, but in all but one case (pp-dmem) requires more time overall. Profiling the largest benchmark (pp-regfile) reveals that 60% of the overall runtime is going to proof canonization. This is an unacceptably high price to pay for the search space pruning we are achieving. We address this problem, but first consider data on the canonization itself.

Figure 9 breaks out the number of uses of the different rewrite rules during canonization. Note that the numbers for the **Collapse** rules do not count uses of the following rules, where it is never necessary to build the left hand side at all when canonizing equality proofs as they are being built:

$$\begin{aligned} 1 * x &\rightarrow x \\ x * 1 &\rightarrow x \\ 1^{-1} &\rightarrow 1 \end{aligned}$$

Benchmark	Size (KB)	dec. orig	time orig (s)	dec. canon	time canon (s)
dlx-regfile	70.9	2807	2.1	2430	5.6
dlx-dmem	71.0	1336	1.0	1025	1.8
pp-regfile	2480.0	115197	295.7	43610	336.9
pp-dmem	1842.2	25928	68.1	11991	58.2
pp-bloaddata	314.2	4060	1.7	3502	3.3
pp-TakenBranch	1842.3	15364	26.1	9616	33.2

Fig. 8. Comparison of original CVC and CVC with canonization of equality proofs on hardware verification benchmarks (“dec.” stands for decisions).

Benchmark	Collapse	RAssoc	InverseIn	Total
dlx-regfile	87515	156463	85961	329939
dlx-dmem	25576	46584	22721	94881
pp-regfile	4620398	10391230	3933697	18945325
pp-dmem	964806	1885856	737122	3587784
pp-bloaddata	41329	61975	31831	135135
pp-TakenBranch	486355	834499	273832	1594686

Fig. 9. Number of rewrites by category for canonization of equality proofs when running modified CVC on the given benchmarks.

We address the problem of spending too much time canonizing full equality proofs as follows. Instead of canonizing equality proofs and then extracting the assumptions from proofs of contradictions (with canonical equality subproofs), we extract assumptions from uncanonized proofs of contradictions in a way that incorporates the algebra of equality proofs. In more detail, for each equality subproof occurring in a proof of a contradiction, we compute the difference between the number of positive and the number of negative occurrences of each assumption in that subproof. An occurrence is positive if it is beneath an even number of uses of *Symm*, and negative otherwise. During this computation, we queue up non-equality subproofs of the equality proof for later consideration. It is an easy lemma that the difference between the number of positive and the number of negative occurrences of an assumption is 0 iff that assumption does not occur in the canonized version of the subproof (cf. Theorem 3).

With this “smart” abstraction of otherwise uncanonized proofs of contradictions, we obtain the favorable results in Figure 10. Wallclock times range from slightly slower for some of the smaller benchmarks to 60% faster in the case of the two toughest benchmarks. Profiling the pp-regfile benchmark reveals that smart abstraction now takes a much more acceptable 10% of the overall runtime.

The numbers of decisions used for the smart version are slightly different from the numbers of decisions for the canonizing version (Figure 8). Careful inspection of trace data from canonization shows that in some cases, a **Collapse** rule applies to eliminate an entire non-equality subproof. This sort of elimination will not be possible in general in the case of smart abstraction. And once different conflict

Benchmark	dec. orig	time orig (s)	dec. smart	time smart (s)
dlx-regfile	2807	2.1	2430	2.5
dlx-dmem	1336	1.0	1048	0.9
pp-regfile	115197	295.7	44547	121.9
pp-dmem	25928	68.1	11899	23.7
pp-bloaddata	4060	1.7	3461	2.1
pp-TakenBranch	15364	26.1	11928	24.6

Fig. 10. Comparison of original CVC and CVC with smart abstraction of assumptions from uncanonized proofs of contradictions.

clauses begin to be added, the behaviors of the two versions of CVC are highly likely to diverge.

10 Conclusion and Future Work

Theorem equivalence of equality proofs using independent assumptions is completely characterized by the standard axioms for free groups. Using a standard completion of the group axioms taken as rewrite rules, equality proofs can be put into canonical form. This form is minimal in the sense that the fewest possible assumptions are used. Canonization can be performed using a simple strategy in time linear in the size of the equality proof. Without a strategy, canonization can take cubic time in the proof’s size. Using these results, the standard union-find algorithm for ground equational reasoning can be instrumented to produce minimal proofs of equations $x = y$ in additional time $O(n^{\log_2 3})$, where n is the size of the equivalence class of x and y . Using the algebra of equality proofs, major improvements were achieved in the performance of the CVC tool on hardware verification benchmark formulas. The approach is attractive, because rather than carefully modifying specific algorithms to produce minimal proofs (as in [10, 5]), we apply a simple general-purpose technique. This can result, for example, in improvements for other decision procedures, like arithmetic, that do equational reasoning.

The most exciting avenue for future work is to extend the algebra of equality proofs to an algebra of congruence proofs. In the case of unary function symbols, the congruence proof rule functions like a homomorphism: $\text{Congr}(\text{Trans}(p_1, p_2)) = \text{Trans}(\text{Congr}(p_1), \text{Congr}(p_2))$. If this observation can be generalized appropriately to higher arities, we may be able to canonize congruence proofs. This promises further speedups for tools like CVC, which rely heavily on congruence closure. Careful inspection of some of the conflict clauses generated reveal cases where out of large clauses (e.g., 21 literals) derived from proofs using congruence rules, only a very small number (e.g., 3) are needed for inconsistency.

The authors wish to thank the anonymous reviewers for their helpful comments, as well as Grigori Mints and David Dill for earlier feedback on the ideas.

References

1. C. Barrett, D. Dill, and A. Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT. In *14th International Conference on Computer-Aided Verification*, 2002.
2. Clark Barrett and Sergey Berezin. CVC Lite: A new implementation of the cooperating validity checker. In *Proceedings of the 16th International Conference on Computer Aided Verification*, 2004.
3. Jerry R. Burch and David L. Dill. Automatic verification of pipelined microprocessor control. In David L. Dill, editor, *Conference on Computer-Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 68–80. Springer-Verlag, 1994.
4. T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1992.
5. L. de Moura, H. Rueß, and N. Shankar. Justifying Equality. In S. Ranise and C. Tinelli, editors, *2nd International Workshop on Pragmatics of Decision Procedures in Automated Reasoning*, 2004.
6. J. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: integrated canonizer and solver. In G. Berry, H. Comon, and A. Finkel, editors, *13th International Conference on Computer-Aided Verification*, 2001.
7. D. Knuth and P. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.
8. S. Lahiri, R. Bryant, A. Goel, and M. Talupur. Revisiting Positive Equality. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *LNCS*, pages 1–15. Springer-Verlag, 2004.
9. G. Nelson and D. Oppen. Fast decision procedures based on congruence closure. *Journal of the Association for Computing Machinery*, 27(2):356–64, 1980.
10. R. Nieuwenhuis and A. Oliveras. Union-Find and Congruence Closure Algorithms that Produce Proofs. In S. Ranise and C. Tinelli, editors, *2nd International Workshop on Pragmatics of Decision Procedures in Automated Reasoning*, 2004. (short paper).
11. R. Nieuwenhuis and A. Oliveras. Proof-producing Congruence Closure. In J. Giesl, editor, *16th International Conference on Rewriting Techniques and Applications*, 2005. (under review).
12. A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding Equality Formulas by Small Domains Instantiations. In *Proceedings of the 11th International Computer-Aided Verification Conference*, volume 1633 of *Lecture Notes in Computer Science*, pages 455–469. Springer-Verlag, 1999.
13. H. Ruess and N. Shankar. Deconstructing Shostak. In *16th IEEE Symposium on Logic in Computer Science*, 2001.
14. R. Shostak. Deciding combinations of theories. *Journal of the Association for Computing Machinery*, 31(1):1–12, 1984.
15. A. Stump. *Checking Validities and Proofs with CVC and flea*. PhD thesis, Stanford University, 2002. available from <http://www.cs.wustl.edu/~stump/>.
16. A. Stump, C. Barrett, and D. Dill. CVC: a Cooperating Validity Checker. In *14th International Conference on Computer-Aided Verification*, 2002.
17. M. Velev and R. Bryant. Superscalar Processor Verification Using Efficient Reductions of the Logic of Equality with Uninterpreted Functions. In L. Pierre and T. Kropf, editors, *Correct Hardware Design and Verification Methods*, volume 1703 of *Lecture Notes in Computer Science*, pages 37–53. Springer-Verlag, 1999.