Equality of Terms Containing Associative-Commutative Functions and Commutative Binding Operators is Isomorphism Complete

David A. Basin*

TR 89-1020
June 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Supported in part by an IBM Fellowship.

Equality of Terms Containing Associative-Commutative Functions and Commutative Binding Operators is Isomorphism Complete

David A. Basin*
Department of Computer Science,
Cornell University, Ithaca, NY 14853
basin@cs.cornell.edu

Abstract

We demonstrate that deciding if two terms containing uninterpreted associative-commutative function symbols and commutative variable binding operators are equal is polynomially equivalent to determining if two graphs are isomorphic. The reductions we use provide insight into this result and suggest polynomial time special cases.

1 Introduction

Decision procedures for equality play an important role in automated theorem proving. We examine the complexity of a specific problem in reasoning about equality: determining if two terms, which possibly contain AC (associative-commutative) functions and commutative variable binding operators, are equal. We prove that this is polynomially equivalent to deciding if two graphs are isomorphic.

*This work was supported, in part, by an IBM Fellowship.

If one removes variable binding operators, then polynomial AC-equality decision procedures are trivial. For example, if $\wedge$ is an (infix) AC-function then we can normalize the term

$$s = s_1 \wedge s_2 \wedge ... \wedge s_n$$

by left associating and sorting the s_i. More complex terms (e.g., the s_i themselves contain various associative and commutative functions) may be normalized by a bottom up traversal that left associates and sorts subterms as permitted by associativity and commutativity. Two terms are equal if and only if their normalized counterparts are identical.

The presence of binding operators, such as $\exists$ and $\forall$ in predicate calculus, complicates equality reasoning. Equality of terms with bound variables must be considered modulo renaming of bound variables (α-conversion). However, α-convertible terms may normalize differently. Consider the following example, where $\wedge$ is, as before, associative and commutative, $\exists$ commutes, and f_1 and f_2 are different uninterpreted function symbols that are neither associative nor commutative.

$$\begin{aligned} s &= \exists u.\exists v.(f_1(u,v) \wedge f_1(v,u)) \wedge f_2(u,v) \\ t &= \exists w.\exists x.f_1(w,x) \wedge (f_2(x,w) \wedge f_1(x,w)) \end{aligned}$$

These two terms are equal. If w and x in the second equation are renamed to v and u respectively, this renamed term is normalized by sorting quantifiers lexicographically, and left associating and sorting the conjuncts, this becomes apparent. The critical point though, is that if we simply normalized s and t (s, as is written, is already in the above described normal form) without renaming w and x, then the normal form of t,

$$\exists w.\exists x.(f_1(w,x) \wedge f_1(x,w)) \wedge f_2(x,w),$$

would not be α-convertible to s.

Of course, in the above example, simply comparing the two f_2 terms tells us what the proper bound variable renaming should be. However, in general, the problem is not so simple. The difficulty arises as both subterms and binding operators may be reordered and this obscures what, if any, bound variable renamings will allow the terms to normalize to α-variants. If the binding operators do not commute[1], then equality again

[1]For example, $\lambda x.\lambda y.x$ is not equal to $\lambda y.\lambda x.x$. On the other hand, in predicate calculus, $\exists$ does commute. That is $\exists x.\exists y.t(x,y)$ is equivalent to $\exists y.\exists x.t(x,y)$.

is decidable in polynomial time. For example, if λ is a non-commutative variable binding operator, then we can decide the equality of two terms $t_1 = \lambda x_1.\lambda x_2....,\lambda x_k.s_1$ and $t_2 = \lambda y_1.\lambda y_2....,\lambda y_k.s_2$ where the s_i contain AC-function symbols by renaming each bound y_i by x_i and normalizing t_1 and the renamed t_2. Another polynomial case is when the binding variables commute, but all the function symbols are distinct. Here we can normalize terms by sorting based on function names. The sorted bodies would then suggest a renaming of bound variables that would allow quantifiers to be properly reordered such that equal terms were α-equivalent.

These complications, caused by the addition of commutative binding operators to theories containing AC-functions, do not arise in theorem provers designed around equational theories or binding-variable free logics; they occur in theories that contain variable binding constructs. Our theorem proving work has been in the Nuprl proof development system[4] and Nuprl's logic, a constructive type theory, contains such constructs.

The context in which these problems arose in our work is interesting in its own right. We have written a rewrite package for Nuprl and have recently extended it to allow rewrites driven by higher-order matching as described by Huet and Lang[8]. However, instantiating variables with functions necessitates naming new binding variables. Thus, even if a user attempts to maintain the identity of binding variable names across different terms (in the hope that they will then normalize to α-variants), the rewriting itself may introduce incongruously named bound variables. It is surprising to begin with that the names of bound variables should cause so much concern. It is equally surprising that the combination of two established theorem proving techniques (rewriting and higher-order matching) should be so troublesome.

Our work is similar to that of [9,2] in that they analyze the complexity of matching problems in the presense of associative-commutative function symbols. Indeed, their result that AC-Matching is NP-complete might suggest that our problems are also. However, their problem has much more latitude then ours. For example, the substitutions that they consider need not be variable bijections which is required in our problems. Moreover, it would be very surprising if their NP-complete problems is reducible to ours which is shown to to be isomorphism complete.

Our paper is organized as follows. Section 2 contains basic definitions. Section 3 provides problem statements. Section 4 contains our proof that

our problem is isomorphism complete. The final section considers special cases and draws conclusions.

2 Preliminaries

Terms

Let $T(V,O)$[2] be a collection of terms constructed from denumerable sets of variables V and operators O. O may contain *binding operators*, functions whose arguments contains a variable bound by that operator. That is, if θ is a binding operator its argument is a *bound-id term* written as $x.t$ where x is bound in the term t.

For example, if O is a denumerable set of function symbols, predicates, and propositional connectives, then $T(O)$ contains the terms and formulas of a propositional calculus. If we add the binding operators $\exists$ and $\forall$, which each take a single bound-id term (e.g., $\exists x.t \stackrel{\text{def}}{=} \exists(x.t)$), then $T(O)$ contains the terms and formulas of a predicate calculus. In our own work, O is the set of Nuprl term-constructors and $T(O)$ is the set of terms of Nuprl's type theory.

Substitution

A *substitution* σ is a function from $V \rightarrow T$ that is the identity almost everywhere. Substitutions are extended to T in the obvious way

$$\sigma(f(x_1.t_1, ..., x_n.t_n)) = f(x'_1.\sigma(t_1), ..., x'_n.\sigma(t_n))$$

where $f \in O$ and the binding variables x'_i and their bound occurrences have been, when necessary, renamed to avoid capture. We represent the substitution that maps the variables x_i to the terms t_i by

$$[t_1, t_2, ..., t_n / x_1, x_2, ..., x_n].$$

[2]We will usually omit V or O when the specific set can be determined from context.

Associative-Commutative Equality

A binary operator f is *associative* when it satisfies

$$f(x, f(y, z)) = f(f(x, y), z) \tag{A}$$

for all x and y. f is *commutative* when

$$f(x, y) = f(y, x). \tag{C}$$

We also a call unary binding operator θ[3] commutative when

$$\theta(x.\theta(y.t)) = \theta(y.\theta(x.t)). \tag{C'}$$

The predicate calculus operators $\exists$ and $\forall$ are commutative in the above sense. We shall call operators that obey both (A) and (C) *AC-functions* and operators that obey (C') *C-binding operators.*

We assume that equality is a reflexive, transitive, symmetric, and congruent relation. That is:

$$s = s \tag{1}$$

$$s = t \Rightarrow t = s \tag{2}$$

$$s = t \wedge t = w \Rightarrow s = w \tag{3}$$

$$s = t \Rightarrow f(s_1, ..., s_{i-1}, s, s_{i+1}, ..., s_n) = f(s_1, ..., s_{i-1}, t, s_{i+1}, ..., s_n) \tag{4}$$

$$s = t \Rightarrow x.s = x.t \tag{5}$$

The fourth property, functional congruence, holds for all f of arity n and for every $i \leq n$. The final property, bound-id term congruence, permits congruence reasoning about bound-id operators. We shall make use of these properties without mention.

Two terms s and t are said to be *associative-commutative equivalent* (or AC-equivalent, denoted by $s =_{AC} t$) when the syntactic identity of s and t, modulo bound variable renaming, can be proven using (A), (C), (C') and Equations 1 through 5 as axiom schemas and rules of inference (along with modus ponens). Equivalently, $s =_{AC} t$ if the above equations can be used to rewrite s to s', and s' and t differ only in the names of bound variables.

[3]We shall, as a matter of convention, name operators by Greek letters when they are known to bind variables, and use lower case Roman letters otherwise.

3 Problem Statements

Our proof that deciding AC term equality is polynomially equivalent (written as $\equiv^p$) to Graph Isomorphism uses the following problem statements.

1. **(Directed) Graph Isomorphism** (referred to as DGI in the directed case and GI in the undirected case)

   Instance: Two (directed) graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.

   Question: Is $G_1 \sim G_2$? That is, is there a bijection $\phi: V_1 \rightarrow V_2$ such that $(v_i, v_j) \in E_1$ iff $(\phi(v_i), \phi(v_j)) \in E_2$?

2. **Labeled Directed Graph Isomorphism** (referred to as LDGI)

   Instance: Two directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ that may have labels associated with their vertices and edges.

   Question: Is $G_1 \sim G_2$? That is, is there a bijection $\phi : V_1 \rightarrow V_2$ such that $(v_i, v_j) \in E_1$ iff $(\phi(v_i), \phi(v_j)) \in E_2$ and ϕ preserves edge and vertex labelings?

3. **AC-Equality of Terms Containing Commutative Binding Operators.** (referred to as ACE)

   Instance: Two terms $s,t \in T(O)$ where O possibly contains associative and commutative functions and commutative binding operators.

   Question: Is $s =_{AC} t$?

4 Equivalence Proof

Our proof proceeds in two parts. First we demonstrate that GI polynomial reduces to ACE.

Lemma 1 $GI \leq^p ACE$

Proof: Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be a given instance of GI. Let θ, f, g be three distinct operator symbols, where θ is a C-binding operator, f an AC-binary function, and g a commutative function symbol. We now show how to construct terms

$$s, t \in T(V_1 \cup V_2, \{\theta, f, g\}),$$

such that $s =_{AC} t$ if and only if G_1 is isomorphic to G_2. Let $m = |V_1|$ and $n = |E_1|$. Construct s by outermost binding the variables in V_1 with m nested applications of θ. These binding applications are followed by nested right-associated applications of f to the n occurrences of g, which are in turn applied to the endpoint vertices of the edges in E_1. That is, if the kth edge in E_1 is (v_{i_k}, v_{j_k}) (ordered arbitrarily[4]) then the kth application of g in s will be $g(v_{i_k}, v_{j_k})$ Hence s is

$$\theta(v_1. ... \theta(v_m. f(g(v_{i_1}, v_{j_1}), f(g(v_{i_2}, v_{j_2}), ..., f(g(v_{i_{n-1}}, v_{j_{n-1}}), g(v_{i_n}, v_{j_n}))...). \tag{6}$$

We construct t analogously using G_2. Both constructions may be accomplished in polynomial time (in fact in log space).

Correctness is easy to establish. If ϕ is an isomorphism from G_1 to G_2 then s contains a subterm $g(v_{i_k}, v_{j_k})$ if and only if t contains either the subterm $g(\phi(v_{i_k}), \phi(v_{j_k}))$ or $g(\phi(v_{j_k}), \phi(v_{i_k}))$ in t. It follows that $s =_{AC} t$. Conversely, if $s =_{AC} t$ then there must be a renaming of binding variables in s, such that the resulting term may be rewritten to t using (A), (C), (C') and Equations 1 through 5. This renaming is an isomorphism from G_1 to G_2. □

To prove the converse, we first define a function $frontier_f(t)$ which returns a list of subterms of t.

$$frontier_f(t) = \begin{cases} frontier_f(a_1) \ @ \ ... \ @ \ frontier_f(a_n) & \text{if } t = f(a_1, ..., a_n) \\ [t] & \text{otherwise} \end{cases}$$

The operator @ appends its operand lists. The following are simple facts about *frontier* that we make use of in our proof.

Fact 1 If f is an associative function, then for all terms t, $frontier_f(t) = [C_1, ..., C_k]$ implies that $t =_{AC} f(C_1, f(C_2, ..., f(C_{k-1}, C_k)...))$.

Proof: The proof proceeds by induction on k. If $k = 1$ then $t =_{AC} t$ by reflexivity. Suppose Fact 1 is true for all $1 \leq k \leq n$ and the length of the $frontier_f(t) = n + 1$. We know that t must be of the form $f(t_1, t_2)$ and that by the induction hypothesis there exists an $j, k \leq n$ where

$$\begin{aligned} t_1 &=_{AC} f(C_1, ..., (f(C_{j-1}, C_j))...) \\ t_2 &=_{AC} f(D_1, ..., (f(D_{k-1}, D_k))...) \end{aligned}$$

[4]Edges in G_1 and G_2 are unordered and this corresponds to g being commutative.

and the C_i and D_i constitute the f-frontiers of t_1 and t_2. Then

$$\begin{aligned} frontier_f(t) &= [C_1, ..., C_j] \mathbin{@} [D_1, ..., D_k] \\ &= [C_1, ..., C_j, D_1, ..., D_k], \end{aligned}$$

and by congruence and associativity reasoning

$$\begin{aligned} t &=_{AC} f(f(C_1, ..., (f(C_{j-1}, C_j))...), f(D_1, ..., (f(D_{k-1}, D_k))...)) \\ &=_{AC} f(C_1, ..., f(C_j, f(D_1, ..., f(D_{k-1}, D_k)...). \end{aligned}$$

□

Fact 2 If f is an associative function, $frontier_f(s) = [C_1, ..., C_k]$, $frontier_f(t) = [D_1, ..., D_k]$, and for $1 \leq i \leq k$, $C_i =_{AC} D_i$, then $s =_{AC} t$.

Proof: Immediate from Fact 1. □

Fact 3 If f is an AC-function, and s and t are terms such that $frontier_f(t)$ is (pointwise) AC-equivalent to a permutation of $frontier_f(s)$, then $s =_{AC} t$.

Proof: Using Fact 2 and the associativity and commutativity of f it is easy to establish that any two terms s and s' whose f-frontiers are permutations of each other are AC-equivalent. It follows by Fact 3, that $s =_{AC} t$. □

With these facts, we may now prove:

Lemma 2 $ACE \leq^p GI$

Proof: We shall actually reduce ACE to LDGI. Proofs that labeled directed graph isomorphism is polynomially reducible to DGI may be found in [3,6]. A reduction of DGI to GI may be found in [3].

We provide a procedure ρ that for any $t \in T(O)$ returns a rooted labeled directed acyclic graph (LDAG) G_t which roughly corresponds to its parse tree. The reduction itself consists of applying this procedure to the two terms that comprise the instance of ACE.

Our procedure is specified by recursion on the structure of terms $t \in T(O)$.

Case 1: $t \equiv c$, where c is a constant

G_t is a vertex labeled by c.

Case 2: $t \equiv x$, where x is a variable
G_t is a vertex labeled by x.

Case 3: $t \equiv f(t_1, ..., t_n)$

Case 3a: f is neither associative nor commutative
Let G_t be rooted by a new vertex labeled by f with edges pointing to the roots of the G_{t_i}. Label the edge to G_{t_i} with the integer i.

Case 3b: f is commutative and is not associative ($n = 2$)
G_t consists of a new vertex labeled by f, with unlabeled edges pointing to the roots of G_{t_1} and G_{t_2}.

Case 3c: f is associative and is not commutative ($n = 2$)
Let G_t be rooted by a new vertex v labeled by f. If $[C_1, .., C_k]$ = $frontier_f(t)$, then for each i from 1 to k, attach a directed edge from v to the root of G_{C_i} and label this edge by i.

Case 3d: f is both associative and commutative ($n = 2$)
Same as Case 3c: except do not label the edges leaving v.

Case 4: $t \equiv \theta(x_1.\theta(x_2. ... \theta(x_n.t')...))$ where the outermost term constructor of t' is not θ

Case 4a: θ is not commutative
Let G_t be rooted by a new vertex v labeled by θ. Add an edge from v to $G_{t'}$. In addition, add edges from v to n new vertices $w_1, w_2, ..., w_n$ and label the ith edge by the integer i. For each w_i, add edges from w_i to the vertices in $G_{t'}$ labeled by x, and erase the "x"-labels on these vertices.

Case 4b: θ is commutative
Same as Case 4a, except do not label the edges from v to the w_i.

If $s =_{AC} t$ then some finite application of rewrites given by (A),(C),(C'), and Equations 1 through 5 to s results in a term α-convertible to t. It is easy to check that each such rewrite does not, up to isomorphism, alter the image of a term under ρ. Conclude that $G_s \sim G_t$.

The converse is more complicated. Formally, we prove by induction on k that for all terms s and t where $height(G_s) \leq k$ and $G_s \sim G_t$, then $s =_{AC} t$. The function *height* computes the height of its LDAG argument (i.e., the length of the longest path). We proceed by cases on the outermost operator

of s (which must be identical to the outermost operator of t as their images under ρ are isomorphic).

Case 1: $s \equiv c$, where c is a constant

As $G_s \sim G_t$, t must also be c.

Case 2: $s \equiv x$, where x is a variable

As $G_s \sim G_t$, t must also be x.

Case 3: $s \equiv f(s_1, ..., s_n)$ and $t \equiv f(t_1, ..., t_n)$

Case 3a: f is neither associative nor commutative

Then $G_s \sim G_t$ establishes an isomorphism between G_{s_i} and G_{t_i}. By the induction hypothesis we have that $s_i =_{AC} t_i$ and it follows (by congruence) that $s =_{AC} t$.

Case 3b: f is commutative and is not associative

Then $G_s \sim G_t$ implies either $G_{s_1} \sim G_{t_1}$ and $G_{s_2} \sim G_{t_2}$, or alternatively, $G_{s_1} \sim G_{t_2}$ and $G_{s_2} \sim G_{t_1}$. Either case is possible as, unlike Case 3a, the edges leaving the roots f are unlabeled. In the first case we have by the induction hypothesis that $s_1 =_{AC} t_1$ and $s_2 =_{AC} t_2$ from which $s =_{AC} t$ follows it follows by congruence. In the second case $s_1 =_{AC} t_2$ and $s_2 =_{AC} t_1$ and our result follows by congruence and the commutativity of f.

Case 3c: f is associative and is not commutative

G_s is a graph rooted by f with children $G_{C_1}, ..., G_{C_k}$ where $[C_1, ..., C_k] = \mathit{frontier}_f(s)$ Similarly G_t is a graph rooted by f with children $G_{D_1}, ..., G_{D_k}$ where $[D_1, ..., D_k] = \mathit{frontier}_f(t)$. Hence $G_s \sim G_t$ implies that $G_{C_i} \sim G_{D_i}$ and by the induction hypothesis it follows that $C_i =_{AC} D_i$. Conclude, by fact 2, that $s =_{AC} t$.

Case 3d: f is both associative and commutative

Same as Case 3c except that the isomorphism from G_s to G_t only establishes that some permutation of the $G_{C_1}, ..., G_{C_k}$ is isomorphic to the $G_{D_1}, ..., G_{D_k}$. (Unlike case 3c, the edges leaving the roots f are unlabeled.) It follows by the induction hypothesis and Fact 3 that $s =_{AC} t$.

Case 4: $s \equiv \theta(x_1.\theta(x_2. ... \theta(x_n.s')...))$ and $t \equiv \theta(y_1.\theta(y_2. ... \theta(y_n.t')...))$

Case 4a: θ is not commutative

Then $G_s \sim G_t$ implies that $G_{\sigma(s')} \sim G_{t'}$ where σ is the substi-

tution $[y_1, ..., y_n/x_1, ..., x_n]$. Thus, by the induction hypothesis $\sigma(s') =_{AC} t'$ and therefore $\theta(y_1. ... \theta(y_n.\sigma(s'))...)$ is AC-equivalent to $\theta(y_1. ... \theta(y_n.t')...)$. By renaming bound variables we conclude $s =_{AC} t$.

Case 4b: θ is commutative

Same as Case 4a except that σ is a bijection from $\{x_1, ..., x_n\}$ to $\{y_1, ..., y_n\}$. Permuting the names of binding variables does not effect AC-equality as θ is commutative.

□

Putting the previous two lemmas together, we conclude:

Theorem 1 $ACE \equiv^p GI$

5 Discussion

We have given fairly direct reductions between graph isomorphism and AC-equality. In both directions the reductions are efficient enough that a previously coded solution for either of these problems could be used as a decision procedure for the other.

The reduction from ACE to GI also gives some insight into the complexity of AC-term equality and the possibility of polynomial decision procedures for special cases. Observe that if we eliminate binding operators (and hence bound variables) then the labeled DAGs associated with terms would be labeled trees and we could decide equality by extending polynomial-time tree isomorphism algorithms[1] to handle vertex and edge labelings. This confirms our observation in the first section: AC-term equality is tractable in binding variable free theories.

Even when binding variables are present, there are cases where deciding AC-equality is still tractable. For example, as planar graph isomorphism is decidable in polynomial time[7], so is AC-equality when the corresponding term graphs contain no subgraphs contractable to K_5 or $K_{3,3}$. This non-planarity arises when terms share multiple bound variables across multiple subterms. For example, three independent subterms containing the same three bound variables will correspond to a graph containing a subgraph contractable to $K_{3,3}$. We have recently been working in hardware verification. Here, the terms we deal with are relations that represent devices and the

only binding variables are existential quantifiers which represent internal wires, and are shared by relations that are wired together. In this domain, many gate-level devices have small fan-in and fan-out and, as a result, share few bound variables; their corresponding graphs are often planar.

Acknowledgements

This work benefited from significant discussions with Doug Howe, Dexter Kozen, and Michael Slifker.

References

[1] Alfred Aho, John Hopcroft, and Jeffrey Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, 1974.

[2] Dan Benanav, Deepak Kapur, and Paliath Narendran. Complexity of matching problems. In *First International Conference on Rewriting Techniques and Applications*, pages 417–429, Dijon, France, 1985.

[3] Kellogg S. Booth and Charles J. Colbourn. Problems polynomially equivalent to graph isomorphism. Technical Report CS-77-04, University of Waterloo, 1979.

[4] R.L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, 1986.

[5] Nachum Dershowitz et al. Associative-commutative rewriting. In *Eighty International Joint Conference on Artificial Intelligence*, pages 940–944, Karlsruhe, West Germany, 1983.

[6] M. Fontet. Automorphismes de graphes et planarite. *Asterisque*, pages 73–90, 1976.

[7] John Hopcroft and J.K. Wong. A linear time algorithm for isomorphism of planar graphs. In *Proc. 6th Annual ACM Symposium on Theory of Computing*, pages 172–184, 1974.

[8] Gérard Huet and Bernard Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, pages 31–55, 1978.

[9] Deepak Kapur and Paliath Narendran. NP-completeness of the set unification and matching problems. In *8th International Conference On Automated Deduction*, pages 489–495, Oxford, UK, 1986.