Solution of the Robbins Problem

WILLIAM McCUNE *
Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439.

Submitted to *J. Automated Reasoning*, Jan. 22, 1997.

Abstract. In this article we show that the three equations known as commutativity, associativity, and the Robbins equation are a basis for the variety of Boolean algebras. The problem was posed by Herbert Robbins in the 1930s. The proof was found automatically by EQP, a theorem-proving program for equational logic. We present the proof and the search strategies that enabled the program to find the proof.

Key words: Associative-commutative unification, Boolean algebra, EQP, equational logic, paramodulation, Robbins algebra, Robbins problem.

1. Introduction

This article contains the answer to the Robbins question of whether all Robbins algebras are Boolean. The answer is *yes, all Robbins algebras are Boolean.* The proof that answers the question was found by EQP, an automated theorem-proving program for equational logic.

In 1933, E. V. Huntington presented the following three equations as a basis for Boolean algebra [6, 5]:

$$x + y = y + x, \qquad \text{(commutativity)}$$
$$(x + y) + z = x + (y + z), \qquad \text{(associativity)}$$
$$n(n(x) + y) + n(n(x) + n(y)) = x. \qquad \text{(Huntington equation)}$$

The unary operation n can be read as *complement.* (Boolean algebra is ordinarily presented in terms of addition, multiplication, complement, 0, and 1. From Huntington's basis, one can show that a 0 and a 1 with the appropriate properties exist, and if multiplication is defined in the obvious way, it also has the appropriate properties.)

Shortly thereafter, Herbert Robbins posed the question of whether the Huntington equation can be replaced with the following equation, which is shorter by one occurrence of n:

$$n(n(x + y) + n(x + n(y))) = x. \qquad \text{(Robbins equation)}$$

The Robbins equation is clearly valid in all Boolean algebras, so the question can be rephrased as *Does the Huntington equation follow from commutativity,*

* Supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

associativity, and the Robbins equation? (No algorithm exists for deciding whether a finite set of equations is a basis for Boolean algebra [12].)

Robbins and Huntington could not find a proof or counterexample, and the problem later became a favorite of Alfred Tarski, who gave it to many of his students and colleagues [2], [3, p. 245]. Algebras satisfying commutativity, associativity, and the Robbins equation became known as *Robbins algebras*, and the question was sometimes phrased as *Are all Robbins algebras Boolean*?[1]

As far as we know, automated deduction was first attempted on the problem in 1979, when Steve Winker, a student visiting Argonne, learned of the problem from Joel Berman. Larry Wos, one of Winker's advisors at Argonne, suggested attacking the problem by looking for properties, which we call *sufficient conditions*, that force Robbins algebras to be Boolean. For example, it is nearly trivial to show that Robbins algebras satisfying $n(n(x)) = x$ are Boolean.[2]

Examples of conditions that were shown to be sufficient by Argonne's theorem provers are (1) $\forall x(x + x = x)$, (2) $\exists c \forall x(c + x = x)$, and (3) $\exists c \forall x(c + x = c)$.

Winker then proved (by hand) several weaker conditions sufficient. The two such conditions that play a role in the present work are contained in the following two lemmas.

Lemma 1 (S. Winker [16, 17]). *A Robbins algebra satisfying* $\exists c \exists d(c + d = c)$ *is a Boolean algebra.*

Lemma 2 (S. Winker [16, 17]). *A Robbins algebra satisfying* $\exists c \exists d(n(c+d) = n(c))$ *is a Boolean algebra.*

Appendix B contains a computer proof of Lemma 1. Lemma 2 is a strengthening of Lemma 1 (the hypothesis is weaker), and its proof is much more difficult; it is included here mainly for historical purposes (see Sections 2 and 5).

2. The Solution

This section contains the key result—the proof of Lemma 3. The theorem that Robbins algebras and Boolean algebras coincide then follows directly from Lemma 1 (also from Lemma 2).

Lemma 3. *All Robbins algebras satisfy* $\exists c \exists d(c + d = c)$.

Proof. The proof (found automatically by the program EQP) starts with the Robbins equation and uses paramodulation (Section 3.3) with built-in associative-

[1] In [6], Huntington included $x + x = x$ in his basis (along with commutativity, associativity, and the Huntington equation) and incorrectly stated that the four equations are independent; the correction appeared in [5], where he showed that $x + x = x$ can be derived from the other three (which are independent). Folklore incorrectly has it that the Robbins problem arose from Huntington's mistake, in particular, that the error in Huntington's paper is that the Robbins equation appears in place of the Huntington equation [6]. According to Robbins [14], the Robbins problem is not related to Huntington's error.

[2] *Proof.* With the Robbins equation, let x be $n(x)$, complement both sides of the equation, and simplify with $n(n(x)) = x$ to obtain the Huntington equation.

commutative (AC) unification (Section 3.1) and simplification with built-in AC matching. We abbreviate $x+x$ as $2x$, $x+x+x$ as $3x$, and so on. The justification "$m \to n$" indicates paramodulation from m into n, and "simp:n" indicates simplification with n.

7	$n(n(n(x)+y)+n(x+y)) = y$	[Robbins equation[3]]
10	$n(n(n(x+y)+n(x)+y)+y) = n(x+y)$	$[7 \to 7]$
11	$n(n(n(n(x)+y)+x+y)+y) = n(n(x)+y)$	$[7 \to 7]$
29	$n(n(n(n(x)+y)+x+2y)+n(n(x)+y)) = y$	$[11 \to 7]$
54	$n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+z)+n(y+z)) = z$	$[29 \to 7]$
217	$n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+n(y+z)+z)+z) = n(y+z)$	$[54 \to 7]$
674	$n(n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+n(y+z)+z)+z+u)+n(n(y+z)+u)) = u$	$[217 \to 7]$
6736	$n(n(n(n(3x)+x)+n(3x))+n(n(n(3x)+x)+5x)) = n(n(3x)+x)$	$[10 \to 674]$
8855	$n(n(n(3x)+x)+5x) = n(3x)$	[6736 $\to$ 7,simp:54,flip]
8865	$n(n(n(n(3x)+x)+n(3x)+2x)+n(3x)) = n(n(3x)+x)+2x$	$[8855 \to 7]$
8866	$n(n(n(3x)+x)+n(3x)) = x$	[8855 $\to$ 7,simp:11]
8870	$n(n(n(n(3x)+x)+n(3x)+y)+n(x+y)) = y$	$[8866 \to 7]$
8871	$n(n(3x)+x)+2x = 2x$	[8865,simp:8870,flip]

Equation 8871 asserts the existence of an object c, namely $2x$, and an object d, namely $n(n(3x)+x)$, such that $c + d = c$. Q.E.D.

(The preceding proof is what the program presents to the user. The AC instantiations and substitutions performed by the program are difficult to reconstruct by hand, so we include a detailed version of the proof in Appendix A. See Sections 4 and 5 for statistics on the proof search.)

Note that Equation 8855 satisfies the Lemma 2 condition $\exists c$, namely $3x$, and $\exists d$, namely $n(n(3x)+x)+2x$, such that $n(c+d) = n(c)$, so we could have stopped there and used Lemma 2 instead of Lemma 1. See Section 5.

Theorem. *The three equations {commutativity, associativity, Robbins} are a basis for the variety of Boolean algebras.*

Proof. This follows directly from Lemma 1, Lemma 3, and the observation that the Robbins equation is valid in all Boolean algebras.

3. The Theorem Prover EQP

The theorem prover that found the proof of Lemma 3 is EQP, presented in [11].[4] EQP is restricted to equational logic and can perform associative-commutative

[3] EQP automatically rearranges AC subterms and renames variables.

[4] EQP is similar in many ways to our more well known theorem prover Otter [10]; the most important differences are that Otter (1) does not have associative-commutative unification or matching, (2) applies to full first-order statements, (3) has fewer paramodulation options, and (4) always uses the given clause algorithm to drive the search.

(AC) unification and matching. The search algorithm is similar to Knuth-Bendix completion [9], with the option of using the "basic" restriction [1, 13].

We describe here some of EQP's features that were used in the proof of Lemma 3. See [11] for details on these and the other features of EQP.

3.1. AC Unification and Matching

Associative-commutative (AC) unification [15] builds the properties of associativity and commutativity of a binary operation into the inference process so that the corresponding equations need not be present as explicit axioms. Two terms are *AC identical* if they can be made identical by reassociating and commuting subterms. An *AC unifier* of two terms is a substitution (of terms for variables) that makes the two terms AC identical. *AC unification* is the process of finding the set of most general AC unifiers (which is always finite) for a pair of terms. *AC matching* is a special case of AC unification in which only one of the two given terms is instantiated; AC matching is used to simplify derived equations and to determine whether one equation subsumes another.

EQP uses Stickel's AC unification algorithm [15]. Let $+$ be an AC operation, and suppose we are trying to unify $s \equiv s_1 + \cdots + s_n$ with $t \equiv t_1 + \cdots + t_m$. The algorithm constructs a linear homogeneous Diophantine equation that represents identity of s and t; then it computes the basis of solutions. The basis has the property that every solution is a linear combination of the members of the basis. To find all most general AC unifiers, the algorithm considers each subset of the basis. A subset produces a *potential unifier* if it passes all unification conditions except unification of subterms, say s_i and t_j.

A pair of terms can have a great number of most general AC unifiers, and we have an optional heuristic, the *super-0 strategy*, that eliminates the more complicated unifiers.[5] The effect of the super-0 strategy is that if a subset S produces a potential unifier, no supersets of S are considered. Since a different variable is associated with each member of the subset, the corresponding AC unifier, if it exists, is more complicated for larger subsets.

The super-0 strategy causes incompleteness of the proof procedure because it eliminates some of the most general unifiers,[6] but we have not seen any cases in practical work where it blocks all proofs. (We have seen cases, however, in which it blocks short proofs, increasing the time required to find a proof.) The heuristic was used to find the proof of Lemma 3, and its role seems to have been important because a similar search without the heuristic failed to find a proof.

Although AC matching is a special case of AC unification, we use a different algorithm, because (1) AC matching is less complicated (conceptually and practi-

[5] The RRL system [8] has a strategy similar to our super-0 strategy.

[6] With the benchmark problem to find the AC unifiers of the pair of terms $x + x + x$ and $y+z+u+v$, EQP ordinarily returns 1,044,569 unifiers; with the super-0 restriction, EQP returns 139 unifiers.

cally), (2) when using AC matching, we need only one matching substitution, (3) speed of AC matching is much more important than the speed of AC unification because for each AC unifier, the AC matching code can be called hundreds of thousands of times when simplifying the derived equation. For AC matching, we use a backtracking algorithm of our own design, with ideas from the RRL system [8]. See [11] for details on EQP's AC unification and matching.

3.2. Paramodulation and Demodulation

The search for a proof uses paramodulation (an inference rule for equality) and demodulation (simplification of inferred equations) [18]. A simplification ordering $\succ$ on terms is used to orient equations and to guarantee termination of demodulation. The term ordering is $t_1 \succ t_2$ if $length(t_1) > length(t_2)$ and no variable has more occurrences in t_2 than in t_1.[7] Every input and derived equation is oriented, if possible, so that the left-hand side is greater, and each oriented equation is added to the set of demodulators.

Paramodulation is not permitted from or into right-hand sides of oriented equations, and paramodulation is not permitted from or into variables. Each equation inferred by paramodulation is simplified with the set of demodulators. If the simplified equation passes the retention tests (typically subsumption and a length limit) and is orientable, it is then used to simplify all other equations in memory.

EQP provides the option of using the "basic" restriction on paramodulation. In summary, the "basic" restriction says that terms that arise by instantiation alone are ineligible as "into" terms. In more detail, consider each equation as a pair, $\langle$skeleton,substitution$\rangle$. Input equations have an empty (or identity) substitution, and derived equations are constructed as follows. The "into" term must exist as a nonvariable term in the skeleton of the "into" parent. The skeleton of the paramodulant is constructed from the skeletons of the parents by simple equality replacement, and the substitution of the paramodulant is constructed from the substitutions of the parents and the unifier for the inference.[8] Without the "basic" restriction, terms that exist only in the substitution part would be admissible "into" terms as well.

As far as we know, the "basic" restriction was first used in the context of narrowing [4] for unification problems in equational theories. It has been proved complete for several variants variants of paramodulation in the full first-order

[7] The length of a term is the number of function, constant, and variable symbols, and the length of an AC term is computed from the binary tree representation. Demodulation clearly terminates, because each rewrite decreases the length of the equation. This ordering is primitive, but adequate for this work.

[8] The "basic" restriction is not implemented by storing equations as $\langle$skeleton,substitution$\rangle$. Instead, "basic" positions in equations are marked, and the marks are inherited during paramodulation.

predicate calculus with equality [1, 13], and it is related to the idea of prime superpositions [7].

The "basic" restriction imposes an order on derivations, but, like most other strategies, its use can interfere with searches as well as help them. It appears to have been a key strategy in finding the proof of Lemma 3 and similar proofs.

3.3. The Search Algorithm

A pairing algorithm is used to select the next equations for application of paramodulation. Let the weight of a pair of equations be the sum of the lengths of its members, and let the age of a pair be the sum of the ages of its members. (The age of an equation is determined by its position in the sequence of retained equations.) In each iteration of the search loop, either the lightest pair not yet selected or the oldest pair not yet selected is chosen. The *pair selection ratio*, one of EQP's important search parameters, is used to specify the ratio, *lightest:oldest*. The default value is 1:0, that is, to always select the lightest pair.

4. Use of EQP

EQP is not an interactive program. The user states the conjecture, sets a few search parameters, and starts the search. As EQP searches, it sends derived equations and some statistics to the output file. If the search fails or does not look promising, the user can adjust the parameters and try again. By iterating in this way, we try to achieve a *well-behaved search* [11]. We had up to three generic UNIX workstations available when working on this project, so we ran (independent) searches in parallel as well.

The attack that led to the solution of the Robbins problem took place over the course of five weeks (September 6 through October 11, 1996). The search parameters were varied in the following ways in various combinations: (1) the limit on the length of retained equations started at 36 and was raised to 40, 50, 60, 70, and 80; (2) searches were run with and without the super-0 restriction on AC unifiers; (3) searches were run with and without the "basic" restriction on paramodulation; and (4) several searches were run with pair selection ratio 1:0, and several with 1:1.

After about fourteen multiday searches, using a total of about five CPU-weeks of computer time, a proof of the Lemma 2 sufficient condition, $\exists c \exists d (n(c+d) = n(c))$, was found. The successful search took almost 8 days and used about 30 megabytes of memory. The successful search parameters were a length limit of 70, the super-0 restriction, the "basic" restriction, and a pair selection ratio of 1:1.

During the successful search, 49,548 equations (i.e., critical pairs) were derived; during simplification of those derived equations, rewriting was attempted on

2,612,977 terms, and 5,981 terms were rewritten. Of the simplified equations, 17,663 were retained, all of which were oriented and became demodulators. Of the total search time, 0.1% was spent deriving equations (including all AC unification time), 74% was spent simplifying equations, 22% was spent finding existing equations that could be simplified with newly adjoined demodulators, and 3% was spent deciding whether simplified equations were subsumed by existing equations.

5. Proofs of the Huntington Equation

We ran several more searches to try to (1) determine whether a proof could have been found earlier or quicker if we had used different search parameters, (2) find a simpler proof, and (3) find proofs of other sufficient conditions.

Table I lists a summary of the experiments, including the first successful search (m5-70). All of the listed searches started with the Robbins equation and used

Table I. Statistics for Various Proofs

Search	Max-weight	Ratio	Proof	Days	Length
m5-70	70	1:1	Cond. 2	7.85	15
m5-60a	60	1:1	Cond. 2	5.71	15
			Cond. 1	5.72	17
			Hunt.	8.83	194
m5-60b	60	4:1	Cond. 2	3.08	15
			Cond. 1	3.09	17
			Hunt.	5.68	194
m5-50	50	1:1	Cond. 2	10.03	8
			Cond. 1	10.03	12
			Hunt.	10.75	86
m5-50b	50	4:1	Cond. 2	4.89	8
			Cond. 1	4.90	12
			Hunt.	5.56	86

AC unification with the super-0 strategy and the "basic" restriction on paramodulation. All searches except m5-70 had multiple goals, including the Lemma 1 condition (Cond. 1), the Lemma 2 condition (Cond. 2), and the Huntington equation (Hunt.). For the searches with multiple goals, EQP was told to keep searching and prove as many goals as it could.

In all cases, the first proof found and the shortest proofs found are of the Lemma 2 condition. The Lemma 1 condition was always found shortly thereafter in just two or four more steps. In Section 2, we chose to present the proof of the Lemma 1 condition (from the search m5-50) because it leads to a simpler overall proof—Winker's proof (and the computer proof in Appendix B) that Condition 1 is sufficient is much less complicated than his proof (or our computer proof [11]) that Condition 2 is sufficient.

The most important result of these additional experiments is that the Huntington equation was proved directly, giving us automatic solutions that do not rely on either of the Winker lemmas. In particular, following theorem was proved automatically.

$$\left\{\begin{array}{l} x+y = y+x \\ (x+y)+z = x+(y+z) \\ n(n(x+y)+n(x+n(y))) = x \text{ (Rob.)} \end{array}\right\} \Rightarrow n(n(x)+y)+n(n(x)+n(y)) = x \text{ (Hunt.)}$$

We have not presented any of the Huntington equation proofs because of their lengths. [9]

World Wide Web Reference

The program EQP (including the source code), the input files, and the EQP proofs referred to in this article are available on the World Wide Web through the following page.

`http://www.mcs.anl.gov/home/mccune/ar/robbins/`

Also available at that location are the results of several attempts to translate our proofs into more human-readable forms.

Appendix A: Detailed Proof of Lemma 3

This appendix contains a detailed proof of Lemma 3 (i.e., Condition 1). We start with the Robbins equation (7) and derive equation (8871) with AC unification and matching. The numbering of the steps is the same as in the less-detailed proof in Section 2.

$$n(n(n(x)+y)+n(x+y)) = y \tag{7}$$
$$n(n(3x)+x)+2x = 2x \tag{8871}$$

STEP 10

With (7), let x be $n(x)+y$ and y be $n(x+y)$:

[9] The Huntington axiom proof lengths (194 and 86), when compared with the Condition 1 proof lengths (17 and 12), do not indicate relative complexity of the proofs. After Condition 1 is proved, other Boolean properties such as $x + x = x$ are proved, and the nature of the search changes dramatically. In particular, most of the steps toward the ends of the proofs of the Huntington equation are trivial.

$$n(\underline{n(n(n(x)+y)+n(x+y))}+n(n(x)+y+n(x+y))) = n(x+y).$$

Use (7) to replace the underlined term; then rearrange:

$$n(n(n(x+y)+n(x)+y)+y) = n(x+y). \qquad [7\to 7]\ (10)$$

STEP 11

With (7), let y be $n(n(x)+y)$ and x be $x+y$:

$$n(\underline{n(n(x+y)+n(n(x)+y))}+n(x+y+n(n(x)+y))) = n(n(x)+y).$$

Use (7) to replace the underlined term; then rearrange:

$$n(n(n(n(x)+y)+x+y)+y) = n(n(x)+y). \qquad [7\to 7]\ (11)$$

STEP 29

With (7), let x be $n(n(x)+y)+x+y$ and y be y:

$$n(\underline{n(n(n(n(x)+y)+x+y)+y)}+n(n(n(x)+y)+x+2y)) = y.$$

Use (11) to replace the underlined term; then rearrange:

$$n(n(n(n(x)+y)+x+2y)+n(n(x)+y)) = y. \qquad [11\to 7]\ (29)$$

STEP 54

With (7), let x be $n(n(n(x)+y)+x+2y)+n(n(x)+y)$ and y be z:

$$n(n(\underline{n(n(n(n(x)+y)+x+2y)+n(n(x)+y))}+z)+n(n(n(n(x)+y)+x+2y)+ n(n(x)+y)+z)) = z.$$

Use (29) to replace the underlined term; then rearrange:

$$n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+z)+n(y+z)) = z. \qquad [29\to 7]\ (54)$$

STEP 217

With (7), let x be $n(n(n(x)+y)+x+2y)+n(n(x)+y)+z$ and y be $n(y+z)$:

$$n(\underline{n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+z)+n(y+z))}+n(n(n(n(x)+y)+x+2y)+ n(n(x)+y)+z+n(y+z))) = n(y+z).$$

Use (54) to replace the underlined term; then rearrange:

$$n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+n(y+z)+z)+z) = n(y+z). \qquad [54\to 7]\ (217)$$

STEP 674

With (7), let y be u and x be $n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+n(y+z)+z)+z$:

$$n(n(\underline{n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+n(y+z)+z)+z)}+u)+n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+n(y+z)+z)+z+u)) = u.$$

Use (217) to replace the underlined term; then rearrange:

$$n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+n(y+z)+z)+z+u)+n(n(y+z)+u)) = u. \ [217\to 7]\ (674)$$

STEP 6736

With (10), let x be $3v$ and y be $n(n(3v)+v)+2v$:

$$n(n(n(3v+n(n(3v)+v)+2v)+n(3v)+n(n(3v)+v)+2v)+n(n(3v)+v)+2v) = n(3v+n(n(3v)+v)+2v). \qquad (10')$$

With (674), let x be $3v$, y be v, z be $2v$, and u be $n(n(3v)+v)$:

$$n(\underline{n(n(n(n(3v)+v)+5v)+n(n(3v)+v)+n(3v)+2v)+2v+n(n(3v)+v))}+n(n(3v)+n(n(3v)+v))) = n(n(3v)+v). \ (674')$$

Replace the underlined term of (674′), which is AC-identical to the left-hand side of (10′), with the right-hand side of (10′):

$$n(n(3v+n(n(3v)+v)+2v)+n(n(3v)+n(n(3v)+v))) = n(n(3v)+v).$$

Rename the variable and rearrange:

$$n(n(n(n(3x)+x)+n(3x))+n(n(n(3x)+x)+5x)) = n(n(3x)+x). \qquad [10\to 674]\ (6736)$$

STEP 8855

With (7), let x be $n(n(3x)+x)+n(3x)$ and y be $n(n(n(3x)+x)+5x)$:

$$n(\underline{n(n(n(n(3x)+x)+n(3x))+n(n(n(3x)+x)+5x))}+n(n(n(3x)+x)+n(3x)+n(n(n(3x)+x)+5x))) = n(n(n(3x)+x)+5x).$$

Use (6736) to replace the underlined term:

$$n(n(n(3x)+x)+n(n(n(3x)+x)+n(3x)+n(n(n(3x)+x)+5x))) = n(n(n(3x)+x)+5x).$$

With (54), let x be $3x$, z be $n(3x)$, and y be x:

$$n(n(n(n(n(3x)+x)+5x)+n(n(3x)+x)+n(3x))+n(x+n(3x))) = n(3x).$$

The left-hand sides of the preceding two equations are AC-identical; hence

$$n(n(n(3x)+x)+5x) = n(3x). \qquad [6736\to7{:}54]\ (8855)$$

Step 8865

With (7), let y be $n(n(3x)+x)+2x$ and x be $3x$:

$$n(n(n(3x)+n(n(3x)+x)+2x)+\underline{n(3x+n(n(3x)+x)+2x)}) = n(n(3x)+x)+2x.$$

Uset (8855) to replace the underlined term; then rearrange:

$$n(n(n(n(3x)+x)+n(3x)+2x)+n(3x)) = n(n(3x)+x)+2x. \qquad [8855\to7]\ (8865)$$

Step 8866

With (7), let x be $n(n(3x)+x)+4x$ and y be x:

$$n(n(n(n(n(3x)+x)+4x)+x)+\underline{n(n(n(3x)+x)+5x)}) = x.$$

Replace the underlined term with the right-hand side of (8855):

$$n(\underline{n(n(n(n(3x)+x)+4x)+x)}+n(3x)) = x. \qquad (A2)$$

With (11), let x be $3x$ and y be x:

$$n(n(n(n(3x)+x)+4x)+x) = n(n(3x)+x). \qquad (A3)$$

Use (A3) to replace the underlined term of (A2):

$$n(n(n(3x)+x)+n(3x)) = x. \qquad [8855\to7{:}11]\ (8866)$$

Step 8870

With (7), let x be $n(n(3x)+x)+n(3x)$:

$$n(n(\underline{n(n(n(3x)+x)+n(3x))}+y)+n(n(n(3x)+x)+n(3x)+y)) = y.$$

Use (8866) to replace the underlined term; then rearrange:

$$n(n(n(n(3x)+x)+n(3x)+y)+n(x+y)) = y. \qquad [8866\to7]\ (8870)$$

STEP 8871

With (8870), let y be $2x$:

$$n(n(n(n(3x)+x)+n(3x)+2x)+n(3x)) = 2x.$$

Use the preceding equation to simplify (8865):

$$n(n(3x)+x)+2x = 2x. \qquad [8865{:}8870]\ (8871)$$

Q.E.D.

Appendix B: Proof of Lemma 1

This appendix contains a proof of Lemma 1, conjectured and first proved by Winker [16, 17], then later proved automatically by EQP [11].

To simplify the presentation, we first prove a stronger condition sufficient. Both computer proofs were found by EQP, the first in about 5 seconds, and the second in about 2319 seconds.

Lemma 0. *A Robbins algebra satisfying* $\exists c(c+c=c)$ *is a Boolean algebra.*

Proof. We assert that the Huntington equation fails to hold, and we derive a contradiction. The terms A, B, and C are constants. (In the justification, n' indicates the extension of equation n; that is, if n is $t_1 = t_2$, n' is $t_1 + x = t_2 + x$.)

1	$C + C = C$	[hypothesis]
2	$n(n(n(x) + y) + n(x + y)) = y$	[Robbins equation]
3	$n(B + n(A)) + n(n(B) + n(A)) \neq A$	[denial of Huntington equation]
4	$n(n(C) + n(C + n(C))) = C$	[$1 \to 2$]
5	$n(n(C + n(C) + x) + n(C + x)) = C + x$	[$1' \to 2$]
8	$n(n(C + x) + n(n(C) + n(C + n(C)) + x)) = x$	[$4 \to 2$]
9	$n(C + n(C + n(C) + n(C))) = n(C)$	[$4 \to 2$]
13	$n(n(C) + n(C + n(C) + n(C + n(C)))) = C$	[$1 \to 8$]
20	$n(n(C) + n(C + n(C) + n(C))) = C$	[$9 \to 2$,simp:1]
22	$n(C + n(C) + n(C)) = n(C + n(C))$	[$9 \to 2$,simp:20,flip]
24	$n(C + n(C + n(C))) = n(C)$	[9,simp:22]
32	$C + n(C + n(C)) = C$	[$24 \to 5$,simp:13,flip]
35	$n(C + n(C)) + x = x$	[$32' \to 2$,simp:8,flip]
42	$n(n(n(x)) + n(x)) = n(C + n(C))$	[$35 \to 2$,simp:35]
50	$n(n(n(n(x)) + x)) = n(n(x))$	[$42 \to 2$,simp:35]
52	$n(n(n(x))) = n(x)$	[$42 \to 2$,simp:35,50]
58	$n(n(x)) = x$	[$2 \to 52$,simp:2]
87	$n(n(x) + y) + n(x + y) = n(y)$	[$2 \to 58$,flip]
88	$A \neq A$	[3,simp:87,58]

Lemma 1 (S. Winker [16, 17]). *A Robbins algebra satisfying* $\exists c \exists d(c + d = c)$ *is a Boolean algebra.*

Proof. The terms C and D are constants.

2	$D + C = C$	[hypothesis]
3	$n(n(n(x) + y) + n(x + y)) = y$	[Robbins equation]
4	$n(n(C) + n(D + n(C))) = D$	$[2 \to 3]$
8	$n(n(D + n(C + x) + y) + n(C + x + y)) = D + y$	$[2' \to 3]$
20	$n(D + n(C + n(D + n(C)))) = n(D + n(C))$	$[4 \to 3]$
34	$n(n(n(n(x) + y) + n(x + y) + z) + n(y + z)) = z$	$[3 \to 3]$
35	$n(n(n(n(x) + y) + x + y) + y) = n(n(x) + y)$	$[3 \to 3]$
56	$n(n(C) + n(D + n(C + n(x)) + n(C + x))) = D$	$[2 \to 34]$
151	$n(n(D + n(C + n(D + n(C))) + x) + n(n(D + n(C)) + x)) = x$	$[20 \to 3]$
152	$n(n(D + n(C)) + n(C + n(D + n(C)))) = D$	$[20 \to 3,\text{simp}:2]$
173	$n(D + n(D + n(C) + n(C + n(D + n(C))))) = n(C + n(D + n(C)))$	$[152 \to 3]$
197	$n(n(C + n(D + n(C))) + n(C + n(C + n(D + n(C))))) = C$	$[2' \to 151]$
280	$n(n(n(n(n(x) + y) + x + y) + y + z) + n(n(n(x) + y) + z)) = z$	$[35 \to 3]$
837	$n(C + n(D + n(C))) = n(C)$	$[4 \to 151,\text{simp}:173]$
839	$n(n(C) + n(C + n(C))) = C$	$[197,\text{simp}:837,837]$
842	$n(n(C + x) + n(n(C) + n(C + n(C)) + x)) = x$	$[839 \to 3]$
844	$n(C + n(C + n(C + n(C)))) = n(C + n(C))$	$[839 \to 3]$
883	$n(n(C + n(C)) + n(C + C + n(C + n(C)))) = C$	$[844 \to 3]$
946	$n(C+n(C+n(C)+n(C+C+n(C+n(C))))) = n(C+C+n(C+n(C)))$	$[883 \to 3]$
1706	$n(C + C + n(C + n(C))) = n(C)$	$[839 \to 280,\text{simp}:946]$
1734	$D + n(C + n(C)) = D$	$[1706 \to 8,\text{simp}:56,\text{flip}]$
1745	$C + n(C + n(C)) = C$	$[2' \to 1734',\text{simp}:2]$
1802	$n(C + n(C)) + x = x$	$[1745' \to 3,\text{simp}:842,\text{flip}]$

From Equation 1802, we have a term e, namely $n(C+n(C))$, such that $e+e = e$; the result follows by Lemma 0. Q.E.D.

References

1. L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 607*, pages 462–476. Springer-Verlag, 1992.
2. S. Burris. Correspondence, November 1996.
3. L. Henkin, J. D. Monk, and A. Tarski. *Cylindric Algebras, Part I.* North-Holland, 1971.
4. J.-M. Hullot. Canonical forms and unification. In R. Kowalski and W. Bibel, editors, *Proceedings of CADE-5, LNCS Vol. 87*, pages 318–334, Berlin, 1980. Springer-Verlag.
5. E. V. Huntington. Boolean algebra. A correction. *Trans. AMS*, 35:557–558, 1933.
6. E. V. Huntington. New sets of independent postulates for the algebra of logic, with special reference to Whitehead and Russell's *Principia Mathematica*. *Trans. AMS*, 35:274–304, 1933.
7. D. Kapur, D. Musser, and P Narendran. Only prime superpositions need be considered in the Knuth-Bendix completion procedure. *J. Symbolic Computation*, 6:19–36, 1988.

8. D. Kapur and H. Zhang. RRL: Rewrite Rule Laboratory user's manual. Technical Report 89-03, Department of Computer Science, University of Iowa, 1989.
9. D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebras*, pages 263–297. Pergamon Press, Oxford, 1970.
10. W. McCune. OTTER 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, Ill., 1994.
11. W. McCune. 33 basic test problems: A practical evaluation of some paramodulation strategies. In Robert Veroff, editor, *Automated Reasoning: Essays in Honor of Larry Wos*, chapter 5. MIT Press, 1997. To appear.
12. G. F. McNulty. Undecidable properties of finite sets of equations. *J. Symbolic Logic*, 41:589–604, 1976.
13. R. Niewenhuis and A. Rubio. Theorem proving with ordering and equality constrained clauses. *J. Symbolic Computation*, 19(4):321–351, 1995.
14. H. Robbins. Phone conversation, October 1996.
15. M. Stickel. A unification algorithm for associative-commutative functions. *J. ACM*, 28(3):423–434, 1981.
16. S. Winker. Robbins algebra: Conditions that make a near-Boolean algebra Boolean. *J. Automated Reasoning*, 6(4):465–489, 1990.
17. S. Winker. Absorption and idempotency criteria for a problem in near-Boolean algebras. *J. Algebra*, 153(2):414–423, 1992.
18. L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications,* 2nd edition. McGraw-Hill, New York, 1992.