Automatic Proofs and Counterexamples for Some Ortholattice Identities

William McCune[*]
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois, 60439, U.S.A.

January 14, 1998

Abstract

This note answers questions on whether three identities known to hold for orthomodular lattices are true also for ortholattices. One identity is shown to fail by MACE, a program that searches for counterexamples, an the other two are proved to hold by EQP, an equational theorem prover. The problems, from work in quantum logic, were given to us by Norman Megill.

Keywords: Automatic theorem proving, ortholattice, quantum logic, theory of computation.

1 Introduction

An *ortholattice* is an algebra with a binary operation $\vee$ (join) and a unary operation $'$ (complement) satisfying the following (independent) set of identities.

$x \wedge y = (x' \vee y')'$ (definition of meet)
$x \vee y = y \vee x$
$(x \vee y) \vee z = x \vee (y \vee z)$
$x \vee (x \wedge y) = x$
$x'' = x$
$x \vee (y \vee y') = y \vee y'$

[*]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.

From these identities one can easily prove commutativity and associativity of the meet operation, idempotence of meet and join, and the existence of a 1 and 0 with the usual Boolean properties. Therefore I use the following identities as well for ortholattices.

$$
\begin{array}{ll}
x \wedge y = y \wedge x & \\
(x \wedge y) \wedge z = x \wedge (y \wedge z) & \\
x \vee x = x & x \wedge x = x \\
x \vee x' = 1 & x \wedge x' = 0 \\
x \vee 1 = 1 & x \wedge 0 = 0 \\
x \vee 0 = x & x \wedge 1 = x
\end{array}
$$

An *orthomodular lattice* is an ortholattice satisfying

$$x \vee (x' \wedge (x \vee y)) = x \vee y.$$

Consider the following three equations.

$$((a \wedge b') \vee a')' \vee ((a \wedge b') \vee ((a' \wedge ((a \vee b') \wedge (a \vee b))) \vee (a' \wedge ((a \vee b') \wedge (a \vee b))')))) = 1 \quad \text{(E1)}$$

$$(a \vee ((a' \wedge ((a \vee b') \wedge (a \vee b))) \vee (a' \wedge ((a' \wedge b) \vee (a' \wedge b'))))) = 1 \quad \text{(E2)}$$

$$(((a' \wedge b) \vee (a' \wedge b')) \vee (a \wedge (a' \vee b)))' \vee (a' \vee b) = 1 \quad \text{(E3)}$$

These three equations arose in work on quantum logic by Mladen Pavičić and Norman Megill [9, 10]. Each equation was known to hold for orthomodular lattices, but it was unknown whether any of them holds for ortholattices. Megill asked whether any of Argonne's automated deduction programs could be used to solve the problems.

Equation E1 was shown to fail by the program MACE, which found an ortholattice of order 10 violating E1. Equations E2 and E3 were each shown to hold for ortholattices by the program EQP, which produced equational proofs. Equation E3 was proved for ortholattices by Megill (by hand), independently and in parallel to our work, shortly after he asked if our programs could prove it.

Quantum logic is meant to describe probability of events in quantum dynamics (cf. Boolean algebra describes probability in classical dynamics). A. R. Marlow states [3]:

> Quantum theory is simply the replacement in standard probability theory of event-as-subset-of-a-set (abelian, distributive) by event-as-subspace-of-a-Hilbert-space (non-abelian, non-distributive).

However, working with quantum logic has been difficult; for example, decidability of the word problem for orthomodular lattices is unknown. R. Goldblatt states[2]:

> ... further evidence of the intractability of quantum logic. It is perhaps the first example of a natural and significant logic that leaves the usual methods defeated.

The work of Pavičić and Megill is on using various versions of the modus ponens rule with quantum logic. Such inference rules may affect computational efficiency of systems based on quantum logic.

1.1 The Programs EQP and MACE

EQP [6] is an automated theorem-proving program for statements in first-order equational logic. It has several strategies for applying equational reasoning and searching for proofs. One of its strengths is that associativity and commutativity of binary operations are built into the inference rules. This feature makes EQP perform well on many problems involving lattice-like structures.

MACE [4] is a program that searches for finite models of first-order statements. In practice, it is limited to fairly simple statements without many variables, and it usually cannot find large models. Even with these limitations, it is a valuable complement to our theorem provers.

(Our more well-known theorem prover Otter [5, 8] does not seem as effective as EQP for lattice-like problems because Otter lacks associative-commutative unification and several valuable paramodulation strategies. See [7] for several examples of using Otter for this type of problem.)

2 Equation E1

I had no intuition about whether E1 could be proved for ortholattices. Hence, I put the programs to work in parallel, with EQP searching for a proof and MACE searching for a counterexample.

To have MACE search for a counterexample, I gave it the orthomodular axioms and asserted that there exist two elements, a and b, for which E1 fails. Equation E1 is too complex for MACE, but (because it is negated, with existentially quantified variables) we can introduce names for subterms of E1, replacing it with the following set of simpler equations.[1]

[1] The procedure of breaking up equations is straightforward; it could easily be made automatic.

$a \wedge b' = d_1$
$a \vee b' = d_2$
$a \vee b = d_3$
$a' = d_4$
$d_2 \wedge d_3 = d_5$
$d_4 \wedge d_5' = d_6$
$d_4 \wedge d_5 = d_7$
$d_7 \vee d_6 = d_8$
$(d_1 \vee d_4)' \vee (d_1 \vee d_8) \neq 1$

The ortholattice identities listed in the introduction involving 0, 1, and idempotence were included as well, because such simple equations reduce the search space that MACE must explore.

MACE was iterated, looking for models (i.e., ortholattices violating E1) of orders 1, 2, ... The search spaces were quickly exhausted up through order 6 without finding any models.[2] Order 8 was exhausted, without models, in about 10 minutes using 15 megabytes of memory.[3] A counterexample of order 10 was found in about 15 minutes using 84 megabytes. Table 1 shows the model as produced by MACE, and Figure 1 shows the corresponding ortholattice diagram.

Table 1: Ortholattice violating E1

a	2
b	8
d_1	7
d_2	1
d_3	5
d_4	3
d_5	5
d_6	4
d_7	0
d_8	4

$\wedge$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	2	0	0	2	0	7	0	7
3	0	3	0	3	4	0	3	0	0	4
4	0	4	0	4	4	0	4	0	0	4
5	0	5	2	0	0	5	8	7	8	7
6	0	6	0	3	4	8	6	0	8	4
7	0	7	7	0	0	7	0	7	0	7
8	0	8	0	0	0	8	8	0	8	0
9	0	9	7	4	4	7	4	7	0	9

$\vee$	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1	1
2	2	1	2	1	1	5	1	2	5	1
3	3	1	1	3	3	1	6	1	6	1
4	4	1	1	3	4	1	6	9	6	9
5	5	1	5	1	1	5	1	5	5	1
6	6	1	1	6	6	1	6	1	6	1
7	7	1	2	1	9	5	1	7	5	9
8	8	1	5	6	6	5	6	5	8	1
9	9	1	1	1	9	1	1	9	1	9

$'$	
0	1
1	0
2	3
3	2
4	5
5	4
6	7
7	6
8	9
9	8

[2] MACE is designed to be complete; that is, if no models are found for order n, then there shouldn't be any. Of course, proof by exhaustive search is especially questionable.

[3] All MACE and EQP searches were run on a 180 MHz i686 processor with 128 megabytes of RAM, running the Linux operating system. MACE and EQP are written in the C programming language.

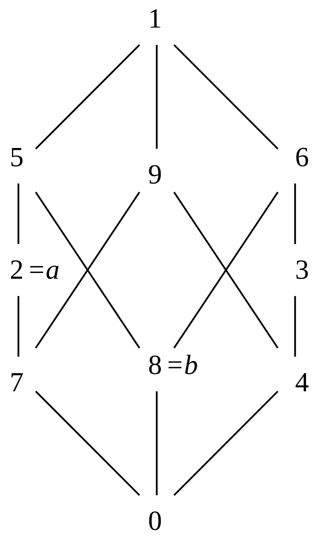

Figure 1: Ortholattice violating E1

3 Equation E2

As with E1, I had no intuition about whether E2 holds for ortholattices. I started a proof search with EQP. Before the counterexample search with MACE was even started, EQP had found a proof.

As usual, I directed EQP to search for a proof by contradiction, asserting the existence of elements a and b for which E2 fails. EQP found the proof below in about 4 seconds.

The justification "$m \rightarrow n$" indicates associative-commutative paramodulation from m into n, that is, equality substitution, using (an instance of) m, into a subterm of (an instance of) n; "simp:i,j,..." indicates associative-commutative simplification with $i, j, \ldots$; and "flip" indicates that the equation is reversed so that the more complex side is on the left.

1	$x'' = x$	
3	$(x \wedge y) \vee x = x$	
4	$(x' \vee y')' = x \wedge y$	
7	$x' \vee x = 1$	
13	$a \vee ((a' \wedge ((b \wedge a') \vee (b' \wedge a'))) \vee (a' \wedge ((b \vee a) \wedge (a \vee b')))) \neq 1$	
15	$(x' \vee y)' = y' \wedge x$	[1 → 4]
19	$(x' \wedge y)' = y' \vee x$	[15 → 1]
20	$x' \wedge y' = (x \vee y)'$	[1 → 15,flip]
21	$a \vee ((a' \wedge ((b \vee a)' \vee (b \wedge a'))) \vee (a' \wedge ((b \vee a) \wedge (a \vee b')))) \neq 1$	[13,simp:20]
24	$(x' \vee y) \wedge y = y$	[3 → 15,simp:1,19,flip]

26	$(x \vee y) \wedge y = y$	$[1 \to 24]$
27	$x' \vee ((y' \wedge x) \vee y) = 1$	$[15 \to 7]$
30	$((x' \wedge y) \vee z)' = z' \wedge (y' \vee x)$	$[15 \to 15]$
32	$((x \vee y) \wedge z) \vee (y \wedge z) = (x \vee y) \wedge z$	$[26' \to 3]$
120	$x' \wedge (y' \vee (z' \wedge u)) = (((u' \vee z) \wedge y) \vee x)'$	$[19 \to 30$,flip]
121	$a \vee ((a \vee ((b \vee a) \wedge (a \vee b')))' \vee (a' \wedge ((b \vee a) \wedge (a \vee b')))) \neq 1$	[21,simp:120]
161	$((x \vee y) \wedge (y \vee z)) \vee y = (x \vee y) \wedge (y \vee z)$	$[26 \to 32]$
162	$1 \neq 1$	[121,simp:161,27]

The search strategy is summarized in the following points. See [6] further for details on these EQP features.

- The pair algorithm. At each iteration of the search loop, a pair of equations was selected for application of the inference rule.
- A selection ratio of four. Through four iterations of the search loop, the shortest pair of equations was selected, then the oldest pair of equations was selected, and so on. In other words, the strategy was four parts shortest-first to one part breadth-first.
- Basic paramodulation [1]. This restriction on application of the inference rule prevented substitution of terms that arise from instantiation alone. Such inferences are redundant.
- Prime paramodulation. This restriction prevented application of the inference rule if any term in the substitution was reducible by the current set of equations. Such inferences are redundant.
- The super-0 limit on associative-commutative (AC) unifiers. When associativity and commutativity of a binary operation are built into the inference process, a great (doubly exponential) number of inferences can be made from a given pair of equations. The super-0 method prevented inferences involving complicated AC substitutions. Our implementation of the method is incomplete, because it simply discards the complicated AC substitutions; a complete implementation that orders substitutions and backtracks could be made to perform in a similar way the to current implementation.
- Simplification. An equation was used as a rewrite rule if the left-hand side had more symbols than the right and if no variable had more occurrences on the right-hand side.
- Weight limit. No limit was placed on the size of retained equations.

4 Equation E3

Again, since I had no intuition about whether E3 holds for ortholattices, I started proof and counterexample searches in parallel. In about 15 minutes, MACE exhausted the ortholattices up through order 8 without finding one that violates E3; then it was set to work on order 10. A proof search with EQP was started using the same strategy that succeeded for E2. Two days later EQP reported the following proof. (The MACE and EQP jobs ran concurrently on the same processor; the EQP process time was about 22.4 hours.)

1	$x'' = x$	
2	$x' \vee (x \vee y) = x' \vee x$	
3	$(x \wedge y) \vee x = x$	
4	$(x' \vee y')' = x \wedge y$	
7	$x' \vee x = 1$	
8	$1 \vee x = 1$	[2,simp:7,7]
13	$b \vee (a' \vee ((b \wedge a') \vee ((a \wedge (b \vee a')) \vee (b' \wedge a')))') \neq 1$	
14	$x' \vee y' = (x \wedge y)'$	[4 $\rightarrow$ 1,flip]
15	$b \vee (a \wedge ((b \wedge a') \vee ((a \wedge (b \vee a')) \vee (b' \wedge a'))))' \neq 1$	[13,simp:14]
16	$(x' \vee y)' = y' \wedge x$	[1 $\rightarrow$ 4]
22	$(x' \wedge y)' = y' \vee x$	[16 $\rightarrow$ 1]
23	$x' \wedge y' = (x \vee y)'$	[1 $\rightarrow$ 16,flip]
24	$b \vee (a \wedge ((b \vee a)' \vee ((b \wedge a') \vee (a \wedge (b \vee a')))))' \neq 1$	[15,simp:23]
25	$(x' \vee y) \wedge y = y$	[3 $\rightarrow$ 16,simp:1,22,flip]
27	$(x \vee y) \wedge y = y$	[1 $\rightarrow$ 25]
28	$x' \vee ((y' \wedge x) \vee y) = 1$	[16 $\rightarrow$ 7]
31	$((x \vee y) \wedge z) \vee (y \wedge z) = (x \vee y) \wedge z$	[27' $\rightarrow$ 3]
32	$((x \wedge y) \vee z) \wedge (y \vee z) = (x \wedge y) \vee z$	[3' $\rightarrow$ 27]
33	$((x' \wedge y) \vee z)' = z' \wedge (y' \vee x)$	[16 $\rightarrow$ 16]
38	$((x' \vee y) \wedge z)' \vee x = z' \vee x$	[27' $\rightarrow$ 22,simp:22,flip]
146	$(x \wedge y)' \wedge (y' \vee (x' \wedge z)) = y' \vee (x' \wedge z)$	[14 $\rightarrow$ 32]
174	$x' \wedge (y' \vee (z' \wedge u)) = (((u' \vee z) \wedge y) \vee x)'$	[22 $\rightarrow$ 33,flip]
175	$((x' \vee y) \wedge z)' = z' \vee (y' \wedge x)$	[146,simp:174,31]
281	$x' \vee ((y' \wedge ((z' \vee u) \wedge x)) \vee (y \vee z)) = 1$	[38' $\rightarrow$ 28',simp:8]
2662	$x' \vee (((y \wedge z)' \wedge ((z' \vee u) \wedge x)) \vee z) = 1$	[3' $\rightarrow$ 281]
2663	$(((x \wedge y) \vee z)' \wedge (y' \vee u)) \vee (y \vee z) = 1$	[1 $\rightarrow$ 2662,simp:23]
2664	$x' \vee ((((x' \wedge y) \vee z)' \wedge (x \vee u)) \vee z) = 1$	[1 $\rightarrow$ 2663]
3141	$x' \vee ((((x' \wedge y) \vee ((x' \vee z) \wedge u))' \wedge (x \vee v)) \vee z) = 1$	[2664' $\rightarrow$ 3',simp:8,flip]
22528	$1 \neq 1$	[175 $\rightarrow$ 24,simp:3141]

5 Conclusion

These results are not deep, but they illustrate one kind of automatic deduction assistance that is becoming available to mathematicians and logicians.

The programs EQP and MACE, and the input files that produce these results are available on the Web at the following location.

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

Appendix

I sent the results of the programs to Megill, who replied:

> For your interest, [your lattice] is actually used to show that the following equation does not have an ortholattice proof. The equation I sent you [E1] was the "missing piece" of a proof that I was attempting to construct. ... Specifically, this is the lattice algebra mapping of axiom A14 of a system of quantum logic of Kalmbach, *Orthomodular Lattices*, p. 240.
>
> $$\begin{aligned}&(((a' \wedge b)' \vee (((a' \vee (((a' \vee b) \wedge (a' \vee b')) \wedge (a \vee (a' \wedge b)))) \wedge (a' \\ &\vee(((a' \vee b) \wedge (a' \vee b')) \wedge (a \vee (a' \wedge b)))')) \wedge (a \vee (a' \wedge (((a' \vee b) \wedge (a' \vee \\ &b')) \wedge (a \vee (a' \wedge b))))))) \wedge ((a' \wedge b)' \vee (((a' \vee (((a' \vee b) \wedge (a' \vee b')) \wedge \\ &(a \vee (a' \wedge b)))) \wedge (a' \vee (((a' \vee b) \wedge (a' \vee b')) \wedge (a \vee (a' \wedge b)))')) \wedge (a \vee \\ &(a' \wedge (((a' \vee b) \wedge (a' \vee b')) \wedge (a \vee (a' \wedge b))))))')) \wedge ((a' \wedge b) \vee ((a' \wedge b)' \\ &\wedge(((a' \vee (((a' \vee b) \wedge (a' \vee b')) \wedge (a \vee (a' \wedge b)))) \wedge (a' \vee (((a' \vee b) \wedge (a' \\ &\vee b')) \wedge (a \vee (a' \wedge b)))')) \wedge (a \vee (a' \wedge (((a' \vee b) \wedge (a' \vee b')) \wedge (a \vee (a' \wedge b \\ &)))))))) = 1.\end{aligned} \tag{E4}$$

Of course, I wondered (1) whether the same ortholattice violating E1 also violates this equation, (2) if so, whether MACE could find it directly, and (3) if not, whether MACE could find another one.

As with E1, the denial of E4 was broken up by naming subterms so that MACE could cope with it. No counterexamples were found among the ortholattices up through order 8. The order 10 search produced an ortholattice violating E4, isomorphic to the one violating E1 (Table 1 and Figure 1).[4]

[4]The first MACE search with E4 required more memory (> 100 megabytes) than the successful E1 MACE search had used (84 megabytes), and I had to find a way reduce the space requirement. Fortunately, two of the larger equations (associativity of meet and $x \vee (y \vee y') = y \vee y'$) could be omitted, because they depend on the other axioms and lemmas; this allowed the search to run in less than 70 megabytes.

As a double check, a MACE run was set up to verify that the ortholattice (as written in Table 1) violates E4 by including constraints corresponding to the table. This type of MACE run simply verifies that an interpretation is satisfies a statement. The verification succeeded.

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] R. Goldblatt. Orthomodularity is not elementary. *J. Symbolic Logic*, 49(2):401–404, 1984.

[3] A. R. Marlow. Orthomodular structures and physical theory. In *Mathematical Foundations of Quantum Theory*, pages 59–70. Academic Press, 1977.

[4] W. McCune. A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems. Tech. Report ANL/MCS-TM-194, Argonne National Laboratory, Argonne, IL, May 1994.

[5] W. McCune. Otter 3.0 Reference Manual and Guide. Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, IL, 1994.

[6] W. McCune. 33 basic test problems: A practical evaluation of some paramodulation strategies. In Robert Veroff, editor, *Automated Reasoning and its Applications: Essays in Honor of Larry Wos*, chapter 5, pages 71–114. MIT Press, 1997.

[7] W. McCune and R. Padmanabhan. *Automated Deduction in Equational Logic and Cubic Curves*, volume 1095 of *Lecture Notes in Computer Science (AI subseries)*. Springer-Verlag, Berlin, 1996.

[8] W. McCune and L. Wos. Otter: The CADE-13 Competition incarnations. *J. Automated Reasoning*, 18(2):211–220, 1997.

[9] N. Megill, Sept. 1997. Correspondence by electronic mail.

[10] Mladen Pavičić and Norman Megill. Binary orthologic with modus ponens is either orthomodular or distributive. Submitted, November 1997.