Quasigroups and Related Systems **15** (2007), 263 – 271

Short identities implying a quasigroup is a loop or group

Nick C. Fiala

Abstract

In this note, we find all identities in product only with at most six variable occurrences that imply that a quasigroup satisfying the identity is a not necessarily trivial loop (group). These investigations were aided by the automated theorem-prover Prover9 and the model-finder Mace4.

1. Introduction

A *quasigroup* consists of a non-empty set Q equipped with a binary operation, which we simply denote by juxtaposition, such that for all $a, b \in Q$, there exist unique $x, y \in Q$ such that $ax = b$ and $ya = b$. Quasigroups are of interest not only in algebra but in combinatorics as well. Alternatively, we may define quasigroups equationally as algebras $(Q; \cdot, \backslash, /)$ of type $(2, 2, 2)$ such that $x\backslash(x \cdot y) = y$, $(x \cdot y)/y = x$, $x \cdot (x\backslash y) = y$, and $(x/y) \cdot y = x$.

A quasigroup is *trivial* if it consists of a single element. A quasigroup Q is a *left* (*right*) *loop* if there exists a *left* (*right*) *neutral element* $e \in Q$ such that $ex = x$ ($xe = x$) for all $x \in Q$. A *loop* is a quasigroup that is both a left loop and a right loop.

Henceforth, e will always denote the (left, right) neutral element of a (left, right) loop and the variables x, y, and z will always be universally quantified over the elements of a quasigroup.

Definition 1.1. We say that an identity *implies that a quasigroup is a* (*left, right*) *loop* (*group*) if and only if all quasigroups satisfying the identity are (left, right) loops (groups). Furthermore, if there exists a non-trivial (left, right) loop (group) satisfying the identity, then we say that the identity *implies that a quasigroup is a not necessarily trivial* (*left, right*) *loop* (*group*).

2000 Mathematics Subject Classification: 20N05
Keywords: quasigroup, loop, group, automated reasoning

In [1], Belousov raised the problem of determining which identities imply that a quasigroup is a loop. It is well-known that an associative quasigroup is a loop, and therefore a group. In [5], it is shown that each of the four *Moufang identities*

$$(x(yz))x = (xy)(zx) \quad (xz)(yx) = x((zy)x)$$
$$((xy)z)y = x(y(zy)) \quad ((yz)y)x = y(z(yx))$$

imply that a quasigroup is a loop, but not necessarily a group. More generally, in [6], Kunen considers *weak associative laws* (identities, other than associativity, for which the left-hand side and right-hand side are different associations of the same word) that imply that a quasigroup is a loop. In particular, he completely settles the problem for the identities of *Bol-Moufang type* (weak associative laws with three distinct variables and eight variable occurrences). Similarly, one may ask which identities imply that a quasigroup is a group. This question was settled for the identities of Bol-Moufang type in [11].

In this note, we endeavor to find all identities in product only with at most six variable occurrences that imply that a quasigroup is a not necessarily trivial loop (group). Perhaps some interesting identities will arise. The author hopes that this note will be of some use as a sort of beginner's tutorial on the use of automated reasoning in equational logic in general and on the powerful software Prover9 and Mace4 in particular. As such, a great deal of detail is shown and many examples and references are given.

2. Prover9 and Mace4

In this section, we briefly describe the software Prover9 and Mace4.

Prover9 [10] is a resolution-style [2], [13] automated theorem-prover for first-order logic with equality that was developed by McCune. Prover9 is the successor to the well-known OTTER [8] theorem-prover and, like OTTER, utilizes the *set of support strategy* [2], [14].

The language of Prover9 is the language of *clauses*, a clause being a disjunction of (possible one or zero) literals in which all variables whose names begin with u, v, w, x, y, or z are implicitly universally quantified and all other symbols represent constants, functions, or predicates (relations). An axiom may also be given to Prover9 as an explicitly quantified first-order formula which is immediately transformed by Prover9 into a set of clauses by a *Skolemization* [2], [3] procedure. The conjunction of these clauses is not necessarily logically equivalent to the formula, but they will be *equisatisfiable* (one is satisfiable if and only if the other is as well) [2], [3]. Therefore, the set of clauses can be used by Prover9 in place of the formula in proofs by contradiction.

Prover9 can be asked to prove a potential theorem by giving it clauses or formulas expressing the hypotheses and a clause or formula expressing the negation of the conclusion. Prover9 finds a proof when it derives the *empty clause*, a contradiction. Prover9 can also be used for *question answering* through the use of *answer literals* [2], [4], [7].

Prover9 has an *autonomous mode* [10] in which all inference rules, settings, and parameters are automatically set based upon a syntactic analysis of the input clauses. The mechanisms of inference for purely equational problems are *paramodulation* and *demodulation*, a restricted from of paramodulation [2], [12]. Paramodulation *from* an equation i *into* an equation j is accomplished as follows: *unify* the left-hand side l of i with a subterm s of j by finding a substitution into the variables of l and s that make them identical (a *most general unifier*), instantiate j with the corresponding substitution, and infer the equation obtained by replacing s in j with the corresponding instance of the right-hand side of i.

One very important parameter used by Prover9 is the *maximum weight* [10] of a clause. By default, the weight of a literal is the number of occurrences of constants, variables, functions, and predicates in the literal and the weight of a clause is the sum of the weights of its literals. Prover9 discards derived clauses whose weight exceeds the maximum weight specified. By specifying a maximum weight, we sacrifice *refutation-completeness* (the guarantee of the existence of a derivation of the empty clause from a non-satisfiable set of clauses) [2], [13], although in practice it is frequently necessary in order to control the size of the clause space while searching for a proof. We will use the autonomous mode throughout this paper, sometimes overriding Prover9's assignment to the maximum weight parameter.

A useful companion to Prover9 is Mace4 [9], also developed by McCune. Mace4 is a finite first-order model-finder. With possibly some minor modifications, the same input can be given to Mace4 as to Prover9, Prover9 searching for a proof by contradiction and Mace4 searching for counterexamples of specified sizes (a structure of size n with a single binary operation found by Mace4 would be returned as an $n \times n$ Cayley table with the elements of the structure assumed to be $0, 1, \ldots, n-1$ and the element in the ith row and jth column being ij).

Remark 2.1. The reader should note that Mace4 interprets non-negative integers as *distinct* constants and other constants as not necessarily distinct unless otherwise stated. This is in contrast to Prover9 which interprets all constants as not necessarily distinct unless otherwise stated. The use of non-negative integers for constants in Mace4 can have the advantage of speeding up the search for a model.

The scripting language Perl was also used to further automate the process.

3. The Search

In this section, we describe our search for identities in product only with at most six variable occurrences that imply that a quasigroup is a not necessarily trivial loop. Clearly, we need not consider identities with more than three distinct variables.

First, all identities in product only with at most three distinct variables and at most six variable occurrences with different left-hand side and right-hand side were generated up to renaming, canceling, mirroring, and symmetry. This resulted in 1353 identities.

Next, we sent each identity (stored in the Perl variable `$identity`) to Prover9 and ran

```
set(auto).                    % autonomous mode
assign(max_seconds, 1).       % one second time limit
                              % per identity
op(500, infix, [/, *, \]).    % quasigroup operations
clauses(sos).                 % set of support clauses
x \ (x * y) = y.
(x * y) / y = x.
x * (x \ y) = y.
(x / y) * y = x.              % quasigroup
e * x = x.
x * e = x.                    % loop
$identity.                    % candidate identity
a != e.                       % non-trivial
end_of_list.                  % end of set of support clauses
```

to search for a proof that a loop satisfying the identity must be trivial. Any identity for which a proof was found was eliminated. This resulted in 332 identities.

Remark 3.1. We determine whether or not Prover9 has found a proof by observing its exit status. Prover9 outputs an exit code of 0 if and only if it finds a proof.

We then sent each remaining identity to Mace4 and ran

```
assign(max_seconds, 60).                    % one minute time limit
                                            % per identity
op(500, infix, [/, *, \]).                  % quasigroup operations
clauses(theory).                            % theory clauses
x \ (x * y) = y.
(x * y) / y = x.
x * (x \ y) = y.
(x / y) * y = x.                            % quasigroup
$identity.                                  % candidate identity
end_of_list.                                % end of theory clauses
formulas(theory).                           % theory formulas
-(exists e all x (e * x = x & x * e = x)).  % not a loop
end_of_list.                                % end of
                                            % theory formulas
```

to search for a non-loop quasigroup of size at most 200 (this is simply Mace4's upper limit and is specified on the command line with `-n2 -N200` or just `-N200`) that satisfies the identity. Any identity for which an example was found was eliminated. This resulted in 35 identities. For example, the identity

$$x(((yx)z)y) = z$$

was eliminated since it is valid in the non-loop quasigroup below.

```
 * :
     | 0 1 2 3 4 5 6 7
   --+----------------
   0 | 1 6 4 3 5 0 7 2
   1 | 3 2 5 1 4 7 0 6
   2 | 0 4 6 7 2 1 3 5
   3 | 5 7 3 4 1 2 6 0
   4 | 2 3 7 6 0 5 4 1
   5 | 7 5 2 0 6 3 1 4
   6 | 6 1 0 2 7 4 5 3
   7 | 4 0 1 5 3 6 2 7
```

Remark 3.2. We determine whether or not Mace4 has found a model by observing its exit status. Mace4 outputs an exit code of 0 if and only if it finds a model.

Next, we sent each remaining identity to Prover9 and ran

```
set(auto).                         % autonomous mode
assign(max_seconds, 60).           % one minute time limit
                                   % per weight per identity
assign(max_weight, $max_weight).   % maximum clause weight
op(500, infix, [/, *, \]).         % quasigroup operations
clauses(sos).                      % set of support clauses
x \ (x * y) = y.
(x * y) / y = x.
x * (x \ y) = y.
(x / y) * y = x.                   % quasigroup
$identity.                         % candidate identity
x * f(x) != f(x) # answer(x).      % not a left loop
end_of_list.                       % end of
                                   % set of support clauses
```

to search for a proof that the identity implies that a quasigroup is a left loop. We have *Skolemized* [2], [3] the negation of $(\exists e)(\forall x)(ex = x)$ to obtain the clause `x * f(x) != f(x)`, where `f` is a *Skolem function* [2], [3]. We use the answer literal `answer(x)` to obtain an expression for the left neutral element for the identities for which we find a proof (this information could also be extracted from the proof itself, although this is not always so easy to do and does not lend itself to automating). We always make a run for every value of the Perl variable `$max_weight` from 20 to 100 in steps of 10. A proof was found for all 35 identities. For example, Prover9 found the following proof that the identity

$$(xy)(x(zy)) = z$$

implies that a quasigroup is a left loop.

```
-------- PROOF --------
Length of proof is 31.
Level of proof is 13.
Maximum clause weight is 11.
7 x \ (x * y) = y.  [input]
8 (x * y) / y = x.  [input]
9 x * (x \ y) = y.  [input]
10 (x / y) * y = x.  [input]
11 (x * y) * (x * (z * y)) = z.  [input]
12 x * f(x) != f(x) # answer(x).  [input]
13 x / (y \ x) = y.  [para (9 (a 1) 8 (a 1 1))]
14 (x / y) \ x = y.  [para (10 (a 1) 7 (a 1 2))]
15 (x * y) \ z = x * (z * y).  [para (11 (a 1) 7 (a 1 2))]
16 x / (y * (x * z)) = y * z.  [para (11 (a 1) 8 (a 1 1))]
17 x * (y * (z * (y \ x))) = z.  [para (9 (a 1) 11 (a 1 1))]
19 x * ((x / y) * (z * y)) = z.  [para (10 (a 1) 11 (a 1 1))]
```

```
26 x \ y = z * (y * (z \ x)).  [para (9 (a 1) 15 (a 1 1))]
29 x / (y * z) = y * (x \ z).  [para (9 (a 1) 16 (a 1 2 2))]
34 x / (x * y) = y.  [back_demod 9 demod (29 (R))]
35 x / (y \ z) = z * (y * x).  [para (10 (a 1) 17 (a 1 2 2)) flip a]
41 x * (y * x) = y.  [back_demod 13 demod (35)]
43 (x * y) / z = x * (z * y).  [para (11 (a 1) 34 (a 1 2))]
44 x \ y = y * x.  [para (34 (a 1) 14 (a 1 1))]
45 x / y = z * (y * (x * z)).  [para (17 (a 1) 34 (a 1 2)) demod (44)]
47 x * (y * y) = x.  [back_demod 8 demod (43)]
52 x / (y * z) = y * (z * x).  [back_demod 35 demod (44)]
55 x * (y * (z * x)) = y * z.  [back_demod 26 demod (44 44) flip a]
62 x / y = y * x.  [back_demod 45 demod (55)]
68 (x * y) * z = x * (y * z).  [back_demod 52 demod (62)]
71 x * (x * y) = y.  [back_demod 19 demod (62 68 55)]
75 x * x = y * y.  [para (47 (a 1) 71 (a 1 2))]
76 x * x = c0.  [new_symbol 75]
77 x * c0 = x.  [back_demod 47 demod (76)]
80 c0 * x = x.  [para (77 (a 1) 41 (a 1 2))]
81 $F # answer(c0).  [resolve (80 a 12 a)]
-------- end of proof  -------
```

Furthermore, lines 76 and 81 show that $xx = e$. The interested reader should consult [10] for information on how to read such computer-generated proofs.

We then sent each of these 35 identities, along with the corresponding expression for the left neutral element (stored in the Perl variable `$left_neutral`), to Prover9 and ran

```
set(auto).                        % autonomous mode
assign(max_seconds, 60).          % one minute time limit
                                  % per weight per identity
assign(max_weight, $max_weight).  % maximum clause weight
op(500, infix, [/, *, \]).        % quasigroup operations
clauses(sos).                     % set of support clauses
x \ (x * y) = y.
(x * y) / y = x.
x * (x \ y) = y.
(x / y) * y = x.                  % quasigroup
e * x = x.                        % left loop
$left_neutral = e.                % might help Prover9
$identity.                        % candidate identity
a * e != a.                       % not a right loop
end_of_list.                      % end of set of support clauses
```

to search for a proof that the identity implies that a quasigroup is a right loop (if `$left_neutral` contains the Skolem function `f`, then we omit the line `$left_neutral = e.`). A proof was found for all 35 identities.

Finally, we sent each of these 35 identities to Mace4 and ran

```
assign(max_seconds, 60).    % one minute time limit per identity
op(500, infix, [/, *, \]).  % quasigroup operations
clauses(theory).            % theory clauses
x \ (x * y) = y.
(x * y) / y = x.
x * (x \ y) = y.
(x / y) * y = x.            % quasigroup
0 * x = x.
x * 0 = x.                  % loop
$identity.                  % candidate identity
end_of_list.                % end of theory clauses
```

to search for a non-trivial loop that satisfies the identity. An example was found for all 35 identities.

4. Conclusion

In this final section, we state our main results.

Theorem 4.1. *There are exactly* 35 *identities in product only with at most six variable occurrences that imply that a quasigroup is a not necessarily trivial loop* (*up to renaming, canceling, mirroring, and symmetry*). *These* 35 *identities are shown below.*

$$\begin{array}{cccc}
(xx)y = x(yx) & x(x((yy)z)) = z & (xx)(y(yz)) = z & (x(x(yy)))z = z \\
((x(xy))y)z = z & (xx)y = z(yz) & x(((xy)z)y) = z & (xx)(y(zy)) = z \\
((xx)(yz))y = z & x(xy) = (zz)y & ((xx)y)z = zy & x(y(xy)) = zz \\
x((yx)y) = zz & x(y((xy)z)) = z & x((yx)(yz)) = z & (xy)(x(yz)) = z \\
(x(y(xy)))z = z & ((x(yx))y)z = z & x((yx)z) = yz & (xy)(x(zy)) = z \\
((xy)(xz))y = z & x(y(xz)) = zy & x((yy)(xz)) = z & (x(y(yx)))z = z \\
((x(yy))x)z = z & (xy)(yz) = xz & x(((yy)z)x) = z & x((yy)z) = zx \\
x(yz) = (xy)z & x(y(zx)) = yz & (xy)(zx) = yz & x(yz) = (xz)y \\
(xy)(zx) = zy & x(yz) = y(zx) & (xy)z = y(zx) &
\end{array}$$

Similarly, one can prove the following.

Theorem 4.2. *There are exactly* 16 *identities in product only with at most six variable occurrences that imply that a quasigroup is a not necessarily trivial group* (*up to renaming, canceling, mirroring, and symmetry*). *These* 16 *identities are shown below.*

$$\begin{array}{cccc}
x(((xy)z)y) = z & x(y((xy)z)) = z & x((yx)(yz)) = z & (xy)(x(yz)) = z \\
x((yx)z) = yz & (xy)(x(zy)) = z & ((xy)(xz))y = z & x(y(xz)) = zy \\
(xy)(yz) = xz & x(yz) = (xy)z & x(y(zx)) = yz & (xy)(zx) = yz \\
x(yz) = (xz)y & (xy)(zx) = zy & x(yz) = y(zx) & (xy)z = y(zx)
\end{array}$$

References

[1] **V. D. Belousov**: *Foundations of the Theory of Quasigroups and Loops*, Izdat. Nauka, Moscow, 1967.

[2] **C. Chang and R. Lee**: *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, San Diego, CA, 1973.

[3] **M. Davis and H. Putnam**: *A computing procedure for quantification theory*, J. Assoc. Comput. Mach. **7** (1960), 201–215.

[4] **C. Green**: *Proving by resolution as a basis for question-answering systems*, in: Machine Intelligence Vol. 4, Elsevier, New York, 1969, 183–205.

[5] **K. Kunen**: *Moufang quasigroups*, J. Algebra **183** (1996), 231–234.

[6] **K. Kunen**: *Quasigroups, loops, and associative laws*, J. Algebra **185** (1996), 194–204.

[7] **K. Kunen**: *The semantics of answer literals*, J. Automat. Reason. **17** (1996), 83–95.

[8] **W. McCune**: *OTTER* (`http://www.cs.unm.edu/~mccune/otter/`).

[9] **W. McCune**: *Mace4* (`http://www.cs.unm.edu/~mccune/prover9/`).

[10] **W. McCune**: *Prover9* (`http://www.cs.unm.edu/~mccune/prover9/`).

[11] **J. D. Phillips and P. Vojtěchovský**: *The varieties of quasigroups of Bol-Moufang type: an equational reasoning approach*, J. Algebra **293** (2005), 17–33.

[12] **G. A. Robinson and L. Wos**: *Paramodulation and theorem-proving in first-order theories with equality*, in: Machine Intelligence Vol. 4, Elsevier, New York, 1969, 135–150.

[13] **J. A. Robinson**: *A machine-oriented logic based on the resolution principle*, J. Assoc. Comput. Mach. **12** (1965), 23–41.

[14] **L. Wos, G. A. Robinson and D. F. Carson**: *Efficiency and completeness of the set of support strategy in theorem proving*, J. Assoc. Comput. Mach. **12** (1965), 536–541.

Department of Mathematics
St. Cloud State University
St. Cloud, MN 56301,
U.S.A.
E-mail: ncfiala@stcloudstate.edu

Received November 30, 2006
Revised February 7, 2007