The Semantics of Answer Literals

Kenneth Kunen [1]

University of Wisconsin, Madison, WI 53706, U.S.A.

kunen@cs.wisc.edu

August 14, 1995

ABSTRACT

We describe a semantics for answer literals which is not tied to the specific details of the resolution proof procedure. We also describe a number of applications of answer literals to mathematical theorem proving.

§**1. Introduction.** The use of answer literals is well-known in resolution. It was introduced in the 1960's by Green [5], and is now discussed in many texts on automated reasoning [2, 13] and on AI. This technique is usually presented as trick for keeping track of the variable bindings, thus simulating in a general resolution framework what Prolog would do with Horn clauses. For example, consider:

D1. $p(a,b)$
D2. $p(b,c)$
D3. $gp(x,z) \vee \neg p(x,y) \vee \neg p(y,z)$
Q1. $\neg gp(x,z) \vee ANS(x,z)$

We may think of D1,D2,D3 as the database; D1,D2 axiomatize the parenthood relation and D3 is an axiom about grandparenthood. The clause Q1 can be viewed as the negation of a theorem (or query), $\exists xz\, gp(x,z)$. If this negation is expressed simply as the clause $\neg gp(x,z)$, then resolution will derive the empty clause, proving the theorem, but not actually producing an explicit x, z satisfying $gp(x,z)$. Instead, we feed clause Q1 to the resolution prover, deriving $ANS(a,c)$, meaning that in the course of the deduction, x got bound to a and z got bound to c, so $gp(a,c)$ holds.

Here, since all the clauses are Horn clauses, this particular example has an obvious translation into Prolog syntax. Non-Horn examples may produce disjunctive answers. For example, if we replaced D1 by $p(a_1,b) \vee p(a_2,b)$, then we could derive $ANS(a_1,c) \vee ANS(a_2,c)$, but we could not derive either $ANS(a_1,c)$ or $ANS(a_2,c)$.

A completely different application with exactly the same semantics comes from reasoning about exceptions, as in the following famous example:

R0. $\neg exception(x,y) \vee ANS(x,y)$
R1. $\neg bird(x) \ \vee\ has\_wings(x) \ \vee\ exception(x,r1)$
R2. $\neg bat(x) \ \vee\ has\_wings(x) \ \vee\ exception(x,r2)$
R3. $\neg has\_wings(x) \ \vee\ can\_fly(x) \ \vee\ exception(x,r3)$
F1. $bird(tweety)$
F2. $\neg can\_fly(tweety)$

[1] Author supported by NSF Grant CCR-9503445.

Deleting the ANS literal: R0,R1,R2,R3,F1,F2 are inconsistent. R0 says that no rules have exceptions. We may view R0,R1,R2,R3,F1 as the database, and F2 as the negation of the query. Or, exchanging the rolls of query and database, we may also say that $\exists xy\, exception(x, y)$ is a logical consequence of R1,R2,R3,F1,F2. Then, *re-inserting* the ANS literal, we compute which rule got broken – again, we get a disjunctive answer: $ANS(tweety, r1) \vee ANS(tweety, r3)$.

Now, the answer literal technique can be used with other resolution-based methods, such as paramodulation. Some less trivial mathematical examples will be given in §§3,4, but a simple illustration is:

D1. $2 \cdot h(y) = y \quad \vee \quad 2 \cdot h(y) + 1 = y$
D2. $(u + 1) - 1 = u$
Q1. $2 \cdot x \neq c \ \vee \ ANS(x)$
Q2. $2 \cdot x \neq c + 1 \ \vee \ ANS(x)$

Here, the database, D1,D2, axiomatizes some simple facts about the integers, where $h(x)$ denotes the largest integer $\leq x/2$. The *two* query clauses come from the *one* theorem, $\forall y \exists x (2 \cdot x = y \ \vee \ 2 \cdot x = y + 1)$, but we want not only to prove the theorem, but also to retrieve x as an explicit function of y. Negating the theorem in the standard way, we replace y by a Skolem constant c, and the negated $\exists x$ becomes a $\forall x$, but then we insert an ANS disjunct, obtaining $\forall x(\neg(2 \cdot x = c \vee 2 \cdot x = c + 1) \vee ANS(x))$, which becomes the clauses Q1,Q2. After two binary resolution steps and two paramodulation steps, we obtain $ANS(h(c+1))$, which presumably proves $\forall y(2 \cdot h(y+1) = y \ \vee \ 2 \cdot h(y+1) = y+1)$. As we see, when the query is converted to clauses, the answer literal may appear several times in the resolution input. Because of this, and because of the introduction of Skolem constants and functions, one may begin to wonder exactly what the results of such derivations really say about solutions to the original query.

Discussions in the literature [5, 8, 2] of the theory of answer literals are very much tied to the procedural details of the resolution format. They are also based on Herbrand models, so it is not clear how these discussions apply in the presence of equality. They also assume that the ANS appears in only one clause.

In §2, we give a simple semantic explanation of the answer literal method which is not tied to resolution at all. Thus, answer literals could just as well be used with any other proof procedure, such as tableau-based systems, which have been advocated [4] as an alternative to resolution.

The specifics of the deductive system only come in when we consider the issues of soundness and completeness. Completeness sometimes leads to tricky theoretical problems, since many of the known "completeness" results about resolution-based systems really only establish *refutation* completeness – that is, the empty cause will be derived if the input clauses are inconsistent; this does not necessarily imply that every answer clause $ANS(\tau)$ which follows from the axioms will be derived. As usual, soundness questions are easier, but even here we must be careful to distinguish between soundness and refutation soundness. These issues are also covered in §2.

In §§3,4, we illustrate the abstract discussion in §2 with two classes of examples, obtained using McCune's system, OTTER [9].

The first class, covered in §3, is, perhaps, obvious – one can solve an equation. That is, we prove from some algebraic theory that a solution exists, and we use the answer literal to retrieve the solution. Simple examples of this kind occur in the text of Chang and Lee [2]. More sophisticated examples from combinatory logic occur in [13] (see §10.6). We give two examples. In the first, we show how to use OTTER to produce *all* answers to a given problem. The second illustrates the interpretation of Skolem functions occurring in the solution.

In the second class, covered in §4, we solve an *inequation*. This is of interest when we are trying to pin down which instances of a rule are needed to prove a given fact. For example, it is well-known that a semigroup (associative structure) with an identity e satisfying $\forall x(x^2 = e)$ must be a group (obviously), and must also be commutative. But, to prove $ab = ba$, exactly which squares do we need to be the identity? Or, turning this around, if we have $ab \neq ba$, then there is some element whose square is not the identity. What is it? Formally, from the axiom $ab \neq ba$, we are proving the solvability of an inequation, $\exists x(x^2 \neq e)$, and we are using the answer literal to find a solution. Or, using the terminology of the all-birds-can-fly example, assuming a failure of commutativity, we are pinning down an exception to the "rule" $\forall x(x^2 = e)$.

We discuss this example in more detail in §4, and then give a more complicated example related to the Robbins algebra problem.

§**2. Semantics.** In this section, our basic results involve purely semantic notions of logical consequence – not any particular proof theory. We presume some standard presentation of first-order logic. In particular, we shall use $\mathcal{L}$ for our *vocabulary* – that is the set of all function, predicate, and constant symbols. A *sentence* is a logical formula with no free variables. If ϕ is any formula, let $\forall\phi$ be its universal closure.

An $\mathcal{L}$-*structure*, $\mathfrak{M}$, consists of a non-empty domain of discourse, M, together with an assignment of a suitable semantic entity, $s^{\mathfrak{M}}$, for each symbol $s \in \mathcal{L}$. In the case that $\mathcal{L}$ includes the symbol '=', we shall always assume that $\mathfrak{M}$ interprets '=' as true identity on M. If ϕ is a sentence of $\mathcal{L}$, then $\mathfrak{M} \models \phi$ means that ϕ is true in $\mathfrak{M}$. If Σ is a set of sentences of $\mathcal{L}$, then $\mathfrak{M} \models \Sigma$ means that $\mathfrak{M} \models \phi$ for each $\phi \in \Sigma$. If ψ is another sentence of $\mathcal{L}$, then $\Sigma \models \psi$ means that for all $\mathcal{L}$-structures $\mathfrak{M}$, if $\mathfrak{M} \models \Sigma$ then $\mathfrak{M} \models \psi$. Σ is *inconsistent* iff there is no $\mathfrak{M}$ at all such that $\mathfrak{M} \models \Sigma$. So, $\Sigma \models \psi$ if and only if $\Sigma \cup \{\neg\psi\}$ is inconsistent.

Now, suppose that ψ is a sentence of the form $\exists\vec{x}\phi(\vec{x})$. Here, $\vec{x}$ denotes some n-tuple of variables, so that ψ is really of the form $\exists x_1, \ldots, x_n\phi(x_1, \ldots, x_n)$, where ϕ is a formula with at most $x_1, \ldots, x_n$ free. Then, in an attempt to establish $\Sigma \models \psi$, we would normally add $\forall\vec{x}(\neg\phi(\vec{x}))$ to Σ, but instead we add $\forall\vec{x}(\neg\phi(\vec{x}) \vee ANS(\vec{x}))$. Part (1) if the next theorem says that we will never derive a contradiction unless Σ itself is inconsistent, and part (2) gives the semantic meaning to an ANS assertion being derived.

2.1. Theorem. Suppose Σ is a set of sentences of $\mathcal{L}$ and $\exists\vec{x}\phi(\vec{x})$ is a sentence of $\mathcal{L}$, where $\vec{x}$ denotes an n – tuple of variables. Let ANS be an n-place predicate symbol not in $\mathcal{L}$, let $\mathcal{L}'$ be $\mathcal{L} \cup \{ANS\}$, and let Σ' be $\Sigma \cup \{\forall\vec{x}(\neg\phi(\vec{x}) \vee ANS(\vec{x}))\}$. Then:

1. Σ' is inconsistent iff Σ is inconsistent.

2. If, for $i = 1, \dots k$, each $\vec{\tau}_i$ is an n-tuple of terms of $\mathcal{L}$, then

$$\Sigma' \models \forall(ANS(\vec{\tau}_1) \vee \cdots \vee ANS(\vec{\tau}_k)) \qquad (a)$$

if and only if

$$\Sigma \models \forall(\phi(\vec{\tau}_1) \vee \cdots \vee \phi(\vec{\tau}_k)) \qquad (b)$$

Proof. For (1), one way is trivial, since $\Sigma \subset \Sigma'$. Conversely, if Σ is consistent, let $\mathfrak{M} \models \Sigma$, where $\mathfrak{M}$ is an $\mathcal{L}$-structure. So, $\mathfrak{M}$ does not assign a meaning to ANS. We may expand $\mathfrak{M}$ to an $\mathcal{L}'$ structure by declaring $ANS^{\mathfrak{M}'}$ to be universally true; $\mathfrak{M}$ and $\mathfrak{M}'$ have the same domains of discourse and agree on their interpretation of all symbols other than ANS. Then, $\mathfrak{M}' \models \Sigma'$, so Σ' is consistent.

For (2), also, (a) is immediate from (b), since Σ' is equivalent to Σ together with $\forall\vec{x}(\phi(\vec{x}) \Rightarrow ANS(\vec{x}))$. Conversely, if (b) fails, let $\mathfrak{M}$ be a model of Σ in which the sentence $\forall(\phi(\vec{\tau}_1) \vee \cdots \vee \phi(\vec{\tau}_k))$ is false. Expand $\mathfrak{M}$ to an $\mathcal{L}'$ structure by declaring $ANS^{\mathfrak{M}'}$ to be true exactly where ϕ is, so that $\mathfrak{M} \models \forall\vec{x}(\phi(\vec{x}) \iff ANS(\vec{x}))$. Then $\mathfrak{M}$ is a model for Σ' in which $\forall(ANS(\vec{\tau}_1) \vee \cdots \vee ANS(\vec{\tau}_k))$ is false. ∎

Note that we are assuming nothing about the logical complexity of Σ or ϕ. Thus, it is quite possible that $\Sigma \models \exists\vec{x}\phi(\vec{x})$ without there being any sequence of terms $\vec{\tau}_i$ as in (b). For a trivial counter-example, let $\phi(x)$ be $\forall y(p(x) \vee \neg p(y))$; then $\exists x\phi(x)$ is logically valid, and hence follows from $\emptyset$, but there are no τ_i such that $\emptyset \models \forall(\phi(\tau_1) \vee \cdots \vee \phi(\tau_k))$.

In resolution applications, ϕ is quantifier free and the sentences in Σ are all universal, so that the input to the resolution prover, Σ', is universal. In that case Herbrand's Theorem implies immediately that:

2.2. Theorem (Herbrand). Suppose Σ is a set of universal sentences of $\mathcal{L}$ and $\phi(\vec{x})$ is a quantifier-free formula of $\mathcal{L}$. If $\Sigma \models \exists\vec{x}\phi(\vec{x})$, then for some k, there are tuples of terms of $\mathcal{L}$, $\vec{\tau}_1, \dots, \vec{\tau}_k$, such that (b) of Theorem 2.1 holds.

However, often when resolution is applied, the original axioms are not universal, but get converted to universal form by adding Skolem functions (or constants). If the answer obtained does not mention these Skolem functions, then it is still useful by part (2) of the following Lemma. Part (1), which is really a special case, is what justifies the Skolemization step when answer literals are not present (see Theorem 4.1 of [2]).

2.3. Lemma. Suppose Φ is a set of sentences in vocabulary $\mathcal{J}$ and $\hat{\Phi}$ is a set of universal sentences in $\hat{\mathcal{J}} \supseteq \mathcal{J}$ obtained by Skolemizing Φ. Then:
1. Φ is inconsistent iff $\hat{\Phi}$ is.
2. If ψ is a sentence of $\mathcal{J}$, then $\Phi \models \psi$ iff $\hat{\Phi} \models \psi$.

In particular, in the situation of Theorem 2.1, $\mathcal{J} = \mathcal{L}' = \mathcal{L} \cup \{ANS\}$ and $\Phi = \Sigma'$. If we find that $\hat{\Phi} \models \forall(ANS(\vec{\tau}_1) \vee \cdots \vee ANS(\vec{\tau}_k))$, *and* none the $\vec{\tau}_i$ use any Skolem functions, then Lemma 2.3 applies, and we can conclude (a), and hence (b) of Theorem 2.1.

Now, if some of the $\vec{\tau}_i$ involve functions arising from Skolemizing $\exists\vec{x}\phi(\vec{x})$, then the answer might be completely worthless. To continue the above example, suppose $\phi(x)$ is $\forall y(p(x) \vee \neg p(y))$. If we are trying to show that $\emptyset \models \exists x\phi(x)$, *and* find such an x, we would start with $\forall x(\neg\phi(x) \vee ANS(x))$. This Skolemizes to the two clauses, $\neg p(x) \vee ANS(x)$ and

$p(f(z)) \vee ANS(z)$, which derives $ANS(z) \vee ANS(f(z))$, which gives no information at all beyond $\exists x \phi(x)$.

However, if the $\vec{\tau}_i$ only contain Skolem functions arising from Σ, this may impart some interesting information, since we then know that the answer is valid regardless of the choice of the Skolem functions. A concrete example where this is useful given in §3.

Finally, Skolem constants will come in if we wish to prove $\forall \vec{w} \exists \vec{x} \phi(\vec{w}, \vec{x})$, *and* see explicitly how $\vec{x}$ depends on $\vec{w}$. We introduce a tuple of Skolem constants, $\vec{c}$, for the $\vec{w}$. Rather than just deriving a contradiction from $\forall \vec{x} \neg \phi(\vec{c}, \vec{x})$, we instead reason from $\forall \vec{x}(\neg \phi(\vec{c}, \vec{x}) \vee ANS(\vec{x}))$. The τ_i appearing in the answer literals may then involve $\vec{c}$. In addition, the τ_i may involve some tuple of other variables, which get universally quantified as before. We may then apply the following:

2.4. Theorem. Suppose $\vec{c}$ are Skolem constants, $\Sigma' = \Sigma \cup \{\forall \vec{x}(\neg \phi(\vec{c}, \vec{x}) \vee ANS(\vec{x}))\}$, and the τ_i are terms involving $\vec{c}$, along with some tuple of variables, say $\vec{y}$. Suppose

$$\Sigma' \models \forall \vec{y}(ANS(\vec{\tau}_1(\vec{y}, \vec{c}) \vee \cdots \vee ANS(\vec{\tau}_k(\vec{y}, \vec{c}))) \qquad (a')$$

Then

$$\Sigma \models \forall \vec{y} \forall \vec{w}(\phi(\vec{w}, \vec{\tau}_1(\vec{y}, \vec{w})) \vee \cdots \vee \phi(\vec{w}, \vec{\tau}_k(\vec{y}, \vec{w}))) \qquad (b')$$

Proof. Theorem 2.1 applies directly to get

$$\Sigma \models \forall \vec{y}(\phi(\vec{c}, \vec{\tau}_1(\vec{y}, \vec{c})) \vee \cdots \vee \phi(\vec{c}, \vec{\tau}_k(\vec{y}, \vec{c})))$$

Then (b') follows from the fact that Σ does not mention the $\vec{c}$. ■

So, the $\tau_i(\vec{y}, \vec{w})$ do give us an expression in the original language which tells us how $\vec{x}$ depends on $\vec{w}$. A concrete example of this is given in §3.

So far, we have just discussed semantics. If we fix a specific proof theory, we may address problems of soundness and completeness. We have no new results here, but just point out that care must be taken when applying known results – especially concerning completeness, where the literature often blurs the distinction between completeness and refutation completeness.

Let us assume that we have a definition of provability, $\vdash$. The underlying syntax need not be full first-order logic (for example, it may only deal with clauses), but we assume it always has some syntactic entity, $\Box$, which is always false (e.g., the empty clause in resolution, or any $\varphi \wedge \neg\varphi$ in standard predicate logic).

Definition. $\vdash$ is *fully complete* (or, just complete) iff

$$(\Sigma \models \chi) \text{ implies } (\Sigma \vdash \chi) \qquad (1)$$

holds for each Σ and χ.
$\vdash$ is *refutation complete* iff (1) holds whenever χ is $\Box$.

Definition. $\vdash$ is *fully sound* (or, just sound) iff

$$(\Sigma \vdash \chi) \text{ implies } (\Sigma \models \chi) \tag{2}$$

holds for each Σ and χ.
$\vdash$ is *refutation sound* iff (2) holds whenever χ is $\Box$.

Now, all the standard resolution techniques (binary resolution, paramodulation, etc.) are fully sound, so that, assuming the implementation is correct, if your resolution prover tells you that $\Sigma \vdash \chi$, you may conclude that $\Sigma \models \chi$ and then apply the semantic results (Theorem 2.1 or Theorem 2.4). The only case where full soundness fails is in the Skolemization step, which, if it is automated by the theorem prover (as does OTTER [9] if the user requests it), might be considered to be part of the deductive process. This step is refutation sound but not fully sound. That is, if your system Skolemizes Σ to $\hat{\Sigma}$ and then derives χ which depends on the Skolem functions, one cannot conclude that $\Sigma \models \chi$, but only that $\hat{\Sigma} \models \chi$.

For completeness, the problem is somewhat more tricky. There are many completeness results in the literature for various combinations of resolution-based techniques, but these results usually derive refutation completeness, not full completeness. Even very simple resolution techniques fail to be fully complete; for example, the clause $p \vee q$ is not provable from p by binary resolution. In some cases, the *lifting property* plus refutation completeness is sufficient to prove that all answers will be found. That is, we may delete the ANS literals, find a proof of the empty clause, and then insert the ANS literals back in again. This examination of the proof structure is the content of the arguments in [5, 8].

Of course, one must be careful that the rules are phrased so that re-inserting the ANS literal results in a legitimate proof. For example, in negative hyper-resolution, it is important that, as does OTTER, we ignore the ANS literals when deciding whether a clause is negative. Say we start with $\neg p(a)$, $q(a)$, and $p(x) \vee \neg q(x) \vee ANS(x)$. The first step must derive $\neg q(a) \vee ANS(a)$, and the prover must view this as a negative clause.

With equality, many of the combinations with paramodulation do not satisfy the lifing property, and we cannot in general guarantee that all the answers get computed. For example, paramodulation plus binary resolution plus factoring is *refutation* complete whenever the input clauses contain $x = x$. If we start with $x = x$, $a = b$, and $(x \neq y) \vee ANS(x, y)$, then we will derive $ANS(x, x)$ and $ANS(a, b)$, but not $ANS(f(a), f(b))$, even though this is a correct answer and is not subsumed by the other two. We do not know if there is a theorem to the effect that the answer literal method is complete in all "mathematically interesting" cases.

In practice, soundness is usually more important than completeness, since many resolution runs include some bound to clause or term size, which invalidates completeness anyway. However, occasionally (see, e.g., Lemma 2.3 of [7]), the completeness of the Knuth-Bendix procedure is taken as a proof of non-provability (that is, of the existence of a counter-model). In the next section, we use a similar argument as a proof that all answers have been computed; in which case one must be careful to verify the correctness of this assertion in the specific problem at hand.

§3. **Algebraic equations.** We work two examples here. The first involves solving an equation in a group. The second involves Moufang loops, and illustrates an example where the Skolem functions arise in the axioms.

For the first example, say we are in a group, with identity e, and suppose the group has exponent 3 (that is, $x^3 = e$ for all x). What are the solutions to the equation: $axb = bxa$? The natural OTTER input file here is:

```
    assign(max_proofs, 40).  % find 40 answers, if you can
    op(400, xfy, *).  % right associate
    set(knuth_bendix).
    list(sos).
        (x * y) * z = x * (y * z).  % associativity
    % exp 3 groups:
        x * (x * x) = e.
        (x * x) * x = e.
        e * x = x.
        x * e = x.
    % negation of conclusion:
        a * (x * b) != b * (x * a) | $ans(x).
    end_of_list.
```

OTTER produces 40 proofs fairly quickly, but there is a lot of redundancy, since multiple proofs are frequently obtained for the same answer. The proofs are all fairly short. For example, the first answer, ba, is derived as follows:

```
    2,1 [] (x*y)*z=x*y*z.
    3 [] x*x*x=e.
    8,7 [] e*x=x.
    10,9 [] x*e=x.
    11 [] a*x*b!=b*x*a|$ans(x).
    14 [para_into,1.1.1.1,3.1.1,demod,8,2,flip,1] x*x*x*y=y.
    16 [para_into,1.1.1,3.1.1,demod,2,flip,1] x*y*x*y*x*y=e.
    21 [para_into,11.1.1.2,1.1.1,demod,2] a*x*y*b!=b*x*y*a|$ans(x*y).
    22 [copy,21,flip,1] b*x*y*a!=a*x*y*b|$ans(x*y).
    25 [para_from,16.1.1,14.1.1.2.2,demod,10,flip,1] x*y*x*y*x=y*y.
    29 [para_from,25.1.1,14.1.1.2.2] x*x*y*y=y*x*y*x.
    30 [binary,29.1,22.1] $ans(b*a).
```

Observe how this fits in the framework of our discussion of Skolem functions in Theorem 2.4. Let Σ be the axioms for exponent 3 groups, expressed in the language $\mathcal{L} = \{*, e\}$. To prove $\forall uv \exists x(uxv = vxu)$, *and* retrieve x as an expression in $u, v, *, e$, we Skolemized the negation to $axb \neq bxa \vee ANS(x)$, and derived $ANS(ba)$, which means we have shown as a theorem of Σ that $\forall uv(u\,(vu)\,v = v\,(vu)\,u)$.

Eliminating repeated answers from the 40, we obtained the following 15 in the order indicated: ba, ab, bb, aa, $baabb$, $ababb$, $bbabb$, $abbaa$, $babaa$, $aabaa$, $bbaab$, $aabba$, $bbaba$, $aabab$, $bbaaa$. Even by hand, we see some redundancies in this list; for example, in a group of exponent 3, the last answer, $bbaaa$ is equal to bb. Less obviously, the answer $bbaab$ is equal to $ababb$ since $b^2a^2b = (ab)^3 \cdot b^2a^2b = (ab)^2 \cdot ab \cdot b^2a^2b = (ab)^2 \cdot b$.

So, one question is: exactly which of these 15 answers are really different in all groups of exponent 3? A potentially more difficult question is: did we get all the answers? Perhaps, running this to get 80 proofs, we would get more answers. In general, problems of this nature are difficult to answer by automated reasoning techniques, since we are asking about arbitrary models – given any other combination of a's and b's, is there a model for the axioms where it fails to be an answer? However, in this particular case, we can in fact get a complete solution, since the free group of exponent 3 with two generators is finite (with 27 elements), and we can modify the input file to force OTTER to run through this group and output all the answers; at the same time, we get an enumeration of the group. The following input does that:

```
set(ur_res).
set(print_lists_at_end).  % enumerate group at end of file
assign(max_literals, 1).  % this count ignores the answer literals
assign(max_proofs, 40).  % find 40 answers, if you can
op(400, xfy, *).  % right associate
set(knuth_bendix).
lex([e,a,b, x * x]).
weight_list(pick_and_purge).
    weight(elt($(1)),100).
    % so we get a complete set of reductions
    % before generating any elements
end_of_list.
list(sos).
    elt(a).  elt(b).  elt(e).
    -elt(x) | -elt(y) | elt(x*y).
    -elt(x) | x * x * x = e.
        % So that Knuth-Bendix works, we
        % use exp3 only for named group elts
    (x * y) * z = x * (y * z).  % associativity
    e * x = x.
    x * e = x.
    a * (x * b) != b * (x * a) | -elt(x) | $ans(x).
end_of_list.
```

Running this, the sos becomes empty after all the elements are generated, and we see that there are exactly 9 distinct answers: $bb, ba, ab, aa, ababb, baabb, aabab, aabaa, abaab$. More precisely, if $F(a, b)$ is the free group of exponent 3 generated by a, b, we are claiming that:

1. In any group of exponent 3, each of these 9 is a solution to $axb = bxa$.
2. In $F(a, b)$, these 9 are distinct.
3. In $F(a, b)$, every solution to $axb = bxa$ is one of these 9.

Thus, no product involving a and b, other than the 9 listed, could represent a different solution in all groups of exponent 3. Our claim is not based on any abstract completeness theorem (which, as pointed out in §2, we do not have), but rather on the fact that in this particular instance, we have forced OTTER to enumerate all of $F(a, b)$.

The second example comes from Moufang loops. Consider the language to be $\mathcal{L} = \{*\}$. A *loop* is a structure with a left and right identity in which the maps $x \mapsto a*x$ and $x \mapsto x*a$ are bijections for each a. In the 1930's, Moufang studied loops satisfying the following three *Moufang identities*:

$$(x * y) * (z * x) = (x * (y * z)) * x. \tag{M1}$$
$$((x * y) * z) * y = x * (y * (z * y)). \tag{M2}$$
$$x * (y * (x * z)) = ((x * y) * x) * z. \tag{M3}$$

By the 1950's, Bruck and others (see [1]) had shown that these identities are actually equivalent in loops, a fact which is now trivial to derive on OTTER. See Chein [3] for more on the structure of finite Moufang loops, and Wos [12] for more on OTTER derivations involving these identities. Note that an associative loop is a group, and in a group, the Moufang identities are trivial consequences of the associative law.

The axioms M1 – M3 alone do not imply the structure is a loop, and in fact, probably have no interesting consequences at all, since they are valid in any model of $\forall xy(x * y = y)$. However, now consider adding the additional axiom:

$$\forall xy \exists z (x * (z * x) = y) \tag{A}$$

(A) plus M1 – M3 does imply that the structure is a loop. To prove this, Skolemize (A) to $x * (f(x, y) * x) = y$; it is not clear yet whether the z in (A) is determined uniquely from x, y (although this will follow once we prove the structure is a loop). As a first step in investigating the subject, we tried to prove that there is a right identity, $\exists u \forall x (x * u = x)$, *and* obtain an explicit expression for the identity. Following the usual procedure, we try to derive a contradiction from $\forall u(\neg \forall x (x * u = x) \vee ANS(u))$. To feed this to OTTER, we Skolemize it to $h(u) * u \neq h(u) \vee ANS(u)$, and then try to derive an expression of the form $ANS(\tau)$. If τ involves the Skolem function h, we could say nothing, as pointed out in §2, but in fact the answers obtained involved only f, not h, so they did yield some information. The input file was:

```
assign(max_weight, 40).
op(400, xfx, *).        % don't associate
assign(max_proofs, 10).
set(knuth_bendix).
list(sos).
% Moufang identities:
     (x * y) * (z * x)  = (x * (y * z)) * x.
     ((x * y) * z) * y  = x * (y * (z * y)).
     x * (y * (x * z))  = ((x * y) * x) * z.
x * ( f(x,y) * x) = y.                     % additional axiom
g(u) * u != g(u) | $ans(u).         % negation of conclusion
end_of_list.
```

Our first answer was the ugly $(f(f(y, y)*y, f(y, y)*y)*(f(y, y)*y))*(f(y, y)*y)$, but the fourth was $f(y, y) * y$ and the sixth was $y * f(y, y)$, so *regardless* of the choice of Skolem function f, we know that $f(y, y) * y$ and $y * f(y, y)$ are right identities. It follows that every model for axioms M1 – M3 and A satisfies $\forall yzw(y*(z*y)=y \Rightarrow w*(z*y)=w)$ and $\forall yzw(y*(z*y)=y \Rightarrow w*(y*z)=w)$. By a second OTTER run (or by symmetry), we get the same results about left identities. Adding all these facts, OTTER easily proves the rest of the loop axioms.

§4. **Algebraic inequations.** We start with some more details about the example mentioned in the Introduction: In a semigroup with identity e, which squares being equal to e will enable us to conclude that $ab = ba$? Here, the input file is:

```
op(400, xfy, *).  % right associate
set(knuth_bendix).
assign(max_proofs, 10).
list(sos).
    % semigroups with identity:
        x * e = x.           e * x = x.
        x * (y * z) = (x * y) * z.
    a * b != b * a.
    x * x = e | $ans(x).
end_of_list.
list(usable).
    x = x.
end_of_list.
list(demodulators).
    x * e = x.           e * x = x.
end_of_list.
```

Note the difference in logical form here. In the first example of §3, we were trying to prove the existence of the solution to an *equation*, $\exists x(axb = bxa)$. Now, we are assuming $ab \neq ba$, and trying to prove the existence of the solution to an *inequation*, $\exists x(xx \neq e)$. Here, we get a *disjunctive* answer: either a or b or ab. OTTER's first proof is:

```
1 [] x*e=x.
2 [] e*x=x.
4 [] x*y*z= (x*y)*z.
5 [copy,4,flip,1] (x*y)*z=x*y*z.
7 [] a*b!=b*a.
8 [copy,7,flip,1] b*a!=a*b.
9 [] x*x=e|$ans(x).
10 [para_into,5.1.1.1,9.1.1,demod,2,flip,1] x*x*y=y|$ans(x).
11 [para_into,5.1.1,9.1.1,flip,1] x*y*x*y=e|$ans(x*y).
16 [para_from,11.1.1,10.1.1.2,demod,1,flip,1]
    x*y*x=y|$ans(y)|$ans(y*x).
20 [para_from,16.1.1,10.1.1.2] x*y=y*x|$ans(x)|$ans(y)|$ans(y*x).
21 [binary,20.1,8.1] $ans(b)|$ans(a)|$ans(a*b).
```

Now that we know this answer, it is easy to check by hand that it is best possible – that is, even in groups, no two of $a^2 = e$, $b^2 = e$, $(ab)^2 = e$ implies that $ab = ba$.

A less elementary example regards Robbins algebras; this topic is discussed in some detail in the text of Wos, Overbeek, Lusk, and Boyle [13]. In a Boolean algebra, we may consider OR and NOT to be basic, with the other Boolean operations defined from them. It is easy to see that the following four equations are valid in every Boolean algebra (we are using o for OR and n for NOT):

A: $x\ o\ (y\ o\ z) \ = \ (x\ o\ y)\ o\ z$
C: $x\ o\ y \ = \ y\ o\ x$
H: $n(x\ o\ n(y))\ o\ n(n(x)\ o\ n(y)) \ = \ y$
R: $n(n(x\ o\ y)\ o\ n(n(x)\ o\ y)) \ = \ y$

In 1933, Huntington [6] showed that in fact (A),(C), and (H) together imply all equations valid in Boolean algebras. Somewhat later, Robbins formulated (R). Let us call a *Robbins algebra* any system satisfying (A), (C), and (R). The *Robbins algebra problem*, whether every Robbins algebra must be Boolean, is still open.

The attacks on this problem have been mainly to study the properties of a non-Boolean Robbins algebra, in the hopes of either deriving a contradiction or learning enough about the structure to conjecture a model. Some things are easy to see by hand. Let $\mathcal{A}$ be a Robbins algebra. Axiom (R) implies immediately that the n function maps $\mathcal{A}$ *onto* $\mathcal{A}$. Now replacing y by $n(y)$ in (R) yields $n(n(x\ o\ n(y))\ o\ n(n(x)\ o\ n(y))) \ = \ n(y)$. If n is a 1-1 map, we could delete the outermost n on both sides, obtaining $n(x\ o\ n(y))\ o\ n(n(x)\ o\ n(y)) \ = \ y$. which is axiom (H), so the algebra is Boolean. That is, in a non-Boolean Robbins algebra, the map n is onto but not 1-1, which implies that $\mathcal{A}$ is infinite.

A much deeper result is due to Winker [10, 11], who showed that every non-Boolean Robbins algebra must satisfy the inequalities $\forall x \forall y (x\ o\ y \neq x)$ and $\forall x \forall y (n(x\ o\ y) \neq n(x))$. Using this, one may derive other structural properties of these algebras. For example, it is clear that in a non-Boolean Robbins algebra, (H) must fail somewhere; but where, exactly?

The natural way to answer this is to run OTTER with the following source file, which finishes in a few seconds.:

```
op(400, xfy, o).
assign(max_proofs, 4).
set(knuth_bendix).
assign(max_weight, 20).
assign(pick_given_ratio, 3).
lex([n(x), x, o(x,x)]).
list(sos).
    (x o y) o z = x o (y o z).  % (A)
    x o y = y o x.  % (C)
    n( n(x o y) o n( n(x) o y ) ) = y.  % (R)
    n(x o y) != n(x).  % by Winker
    n(x o n(y)) o n(n(x) o n(y)) = y | $ans(x,y).
end_of_list.
```

The third proof yields `$ans(v65 o n(v65 o v64),v64)`, which means that axiom (H) is false for x, y whenever x is of the form $z\ o\ n(z\ o\ y)$. Thus, we know more than just the fact that n is not 1-1, which asserts merely $\exists y \exists u(u \neq y \ \& \ n(u) = n(y))$. Rather, we have $\forall y \exists u(u \neq y \ \& \ n(u) = n(y))$. Namely, for any y, z, if we set

$$u = \quad n(z \ o \ n(z \ o \ y) \ o \ n(y)) \quad o \quad n(n(z \ o \ n(z \ o \ y)) \ o \ n(y)) \quad ,$$

then, as we have just shown, $u \neq y$, whereas by (R), $n(u) = n(y)$.

References

[1] R. H. Bruck, *A Survey of Binary Systems*, Springer-Verlag, 1958.

[2] C-L. Chang and R. C-T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.

[3] O. Chein, *Moufang Loops of Small Order*, American Mathematical Society (*Memoirs no. 197*), 1978.

[4] M. Fitting, *First-Order Logic and Automated Theorem Proving*, Springer-Verlag, 1990.

[5] C. Green, Theorem Proving by Resolution as a Basis for Question-Answering Systems, in *Machine Intelligence*, Volume 4, B. Meltzer and D. Michie, eds, 1969, pp. 183 – 205.

[6] E. Huntington, New Sets of Independent Postulates for the Algebra of Logic, with Special Reference to Whitehead and Russell's Principia Mathematica, *AMS Transactions* 35 (1933) 274 – 304.

[7] K. Kunen, The Shortest Single Axioms for Groups of Exponent 4, *Computers and Mathematics with Applications* 29 (1995) 1 – 12.

[8] D. Luckham and N. J. Nilsson, Extracting Information from Resolution Proof Trees, *Artificial Intelligence* 2 (1971) 27 – 54.

[9] W. W. McCune, OTTER 3.0 Reference Manual and Guide, Technical Report ANL-94/6, Argonne National Laboratory, 1994.

[10] S. Winker, Robbins Algebra: Conditions That Make a Near-Boolean Algebra Boolean, *J. Automated Reasoning* 6 (1990) 465 – 489.

[11] S. Winker, Absorption and Idempotency Criteria for a Problem in Near-Boolean Algebras, *J. Algebra* 153 (1992) 414 – 423.

[12] L. Wos, OTTER Solves the Moufang Axiom Problem, *preprint.*

[13] L. Wos, R. Overbeek, E. Lusk, and J. Boyle, *Automated Reasoning*, Second Edition, McGraw-Hill, 1992.