6

A Note on Mechanizing Higher Order Logic

J. A. Robinson
College of Liberal Arts
Syracuse University

§1. Any description of a function in the lambda-calculus notation can be translated automatically into a description of the same function in a purely applicative notation which makes no use whatever of abstraction or bound variables.

For example, the function $\sqrt{[x(x+1)]}$ might be described as

$$\lambda x(\text{SQRT}((\text{TIMES } x)((\text{PLUS } x)\text{ONE}))). \tag{1}$$

But by making use of 'Schönfinkel's functions' A, K, and I, defined by

$$(((\text{A}x)y)z) = ((xz)(yz)) \tag{2}$$

$$((\text{K}x)y) = x \tag{3}$$

$$(\text{I}x) = x \tag{4}$$

we can describe $\sqrt{[x(x+1)]}$ as

$$((\text{A}(\text{K SQRT}))((\text{A}((\text{A}(\text{K TIMES}))\text{I}))((\text{A}((\text{A}(\text{K PLUS}))\text{I}))(\text{K ONE})))). \tag{5}$$

The description (5) contains no bound variables or abstraction operations. It is a purely applicative combination of the objects A, K, I, SQRT, TIMES, PLUS, and ONE. Of course, it is not at all obvious that (5) and (1) describe the same function. In order to prove that they do, however, it is necessary only to check that the result of applying (5) to an arbitrary object z is the same as the result of applying (1) to z. In fact, we have

$$(\lambda x(\text{SQRT}((\text{TIMES } x)((\text{PLUS } x)\text{ONE})))z)$$
$$= (\text{SQRT}((\text{TIMES } z)((\text{PLUS } z)\text{ONE})))$$

when (1) is applied to z. When (5) is applied to z, we have

$$(((\text{A}(\text{K SQRT}))((\text{A}((\text{A}(\text{K TIMES}))\text{I}))((\text{A}((\text{A}(\text{K PLUS}))\text{I}))(\text{K ONE}))))z)$$
$$= (((\text{K SQRT})z)(((\text{A}((\text{A}(\text{K TIMES}))\text{I}))((\text{A}((\text{A}(\text{K PLUS}))\text{I}))(\text{K ONE})))z))$$
by (2);

$=$(SQRT(((A((A(K TIMES))I))((A((A(K PLUS))I))(K ONE)))z))
by (3);
$=$(SQRT((((A(K TIMES))I)z)(((A((A(K PLUS))I))(K ONE))z)))
by (2);
$=$(SQRT((((K TIMES)z)(Iz))(((A((A(K PLUS))I))(K ONE))z)))
by (2);
$=$(SQRT((TIMES(Iz))(((A((A(K PLUS))I))(K ONE))z)))
by (3);
$=$(SQRT((TIMES z)(((A((A(K PLUS))I))(K ONE))z)))
by (4);
$=$(SQRT((TIMES z)((((A(K PLUS))I)z)((K ONE)z))))
by (2);
$=$(SQRT((TIMES z)((((K PLUS)z)(Iz))((K ONE)z))))
by (2);
$=$(SQRT((TIMES z)((PLUS(Iz))((K ONE)z))))
by (3);
$=$(SQRT((TIMES z)((PLUS z)((K ONE)z))))
by (4);
$=$(SQRT((TIMES z)((PLUS z)ONE)))
by (3). Therefore (5) describes the same function as (1).

It seems most unlikely that one could in general write purely applicative 'Schönfinkel descriptions', like (5), of functions already known to one in some other form. Fortunately there is a general procedure – the Schönfinkel procedure – which, when applied to any expression written in the more intuitive lambda-calculus notation, will produce a correct translation of it into the Schönfinkel notation. The procedure consists simply of repeatedly applying the three following rules until no further applications can be made:

replace λxx by I; (6)

replace λxB by (KB), provided the expression B contains no occurrences of x; (7)

replace λx(BC) by ((AλxB)λxC) if (BC) contains one or more occurrences of x. (8)

For example, this procedure translates (1) into (5). We have:

(1) $=\lambda x$(SQRT((TIMES x)((PLUS x)ONE)));
$=$((A λx SQRT)λx((TIMES x)((PLUS x)ONE)))
by (8);
$=$((A(K SQRT))λx((TIMES x)((PLUS x)ONE)))
by (7);
$=$((A(K SQRT))((A λx(TIMES x))λx((PLUS x)ONE)))
by (8);
$=$((A(K SQRT))((A((A λx TIMES)λxx))λx((PLUS x)ONE)))
by (8);
$=$((A(K SQRT))((A((A(K TIMES))λxx))λx((PLUS x)ONE)))
by (7);
$=$((A(K SQRT))((A((A(K TIMES))I))λx((PLUS x)ONE)))
by (6);
$=$((A(K SQRT))((A((A(K TIMES))I))((A λx(PLUS x))λx ONE)))
by (8);

$=((A(K\ SQRT))((A((A(K\ TIMES))I))((A((A\ \lambda x\ PLUS)\lambda xx))\lambda x\ ONE)))$
by (8);
$=((A(K\ SQRT))((A((A(K\ TIMES))I))((A((A(K\ PLUS))\lambda xx))\lambda x\ ONE)))$
by (7);
$=((A(K\ SQRT))((A((A(K\ TIMES))I))((A((A(K\ PLUS))I))\lambda x\ ONE)))$
by (6);
$=((A(K\ SQRT))((A((A(K\ TIMES))I))((A((A(K\ PLUS))I))(K\ ONE))))$
by (7);
$=(5)$.

We saw previously, by a special calculation, that (1) and (5) describe the same function. More generally, whenever the Schönfinkel procedure produces a translation F of an expression $\lambda x B$, we may prove that

$$(\lambda x B z) = (F z) \tag{9}$$

holds for an arbitrary z by noting that:

(i) if B is x then $(\lambda x B z) = z$
$= (I z)$ by (4)
$= (F z)$ by (6).

(ii) if B does not contain x then $(\lambda x B z) = z$
$= ((K B) z)$ by (3)
$= (F z)$ by (7).

(iii) if $B = (C D)$ contains x then $(\lambda x B z) = (\lambda x (C D) z)$
$= (C\{z/x\} D\{z/x\})$
$= ((\lambda x C z)(\lambda x D z))$
$= (((A \lambda x C)\lambda x D) z)$ by (2)
$= (F z)$ by (8).

§2. The phenomena described in the previous section were discovered fifty years ago by Schönfinkel (1924). They have since been studied in great detail by Curry and Feys (1958) under the heading *combinatory logic.* An excellent summary can be found in Rosenbloom (1950, Chapter 3, section 4).

For our present purposes the situation can be summed up by saying that whatever can be expressed in a language based on application *and* abstraction as fundamental notions can be expressed in a far simpler language *based on application alone.*

The purpose of this note is to provoke investigations into the use of such simple 'Schönfinkel languages' as the vehicles of meaning in the mechanization of higher order theorem-proving problems. The extreme syntactic and semantic simplicity of these languages makes it seem likely that it will be much easier to develop proof procedures and interactive deductive systems for them than for the lambda-calculi suggested in Robinson (1969). For one thing, they can be treated as first-order equation calculi, permitting the use of resolution and related inference principles (such as paramodulation) which are well understood.

These possibilities are explored in the next section.

§3. Such a first-order system might look like the following. The *terms* are either *variables* x, y, z, etc., or *constants* A, B, C, PLUS, SQRT, etc., or else are *applications* (α, β) in which α and β are terms.

The only *literals* in the system are equations $\alpha = \beta$ and inequations $\alpha \neq \beta$, where α and β are terms.

One makes assertions in the system by writing *clauses*, i.e., finite collections of literals considered as disjunctions of their members, universally quantified with respect to all variables.

In other words, this is a first-order language in which there is only one relation symbol, namely equality; only one function symbol, namely application; and a collection of individual constants.

Among the constants there will be: A, K, I, TRUE, FALSE, NOT, OR, AND, IMPLIES, EQUAL, ALL, EXISTS, CHOICE, IF and FAIL. The first three of these will intuitively denote the three Schönfinkel functions; the next seven will denote what their mnemonic properties suggest; the next two will denote the quantifiers; CHOICE will denote Hilbert's epsilon operator or selection function, which assigns to every nonempty set some member of it (and, to the empty set, some arbitrary object); IF denotes the operator which corresponds to the 'if ... then ... else ...' construction of conditional expressions in languages like ALGOL, LISP or POP-2; FAIL denotes the dual of CHOICE.

These informal and intuitive *semantic* rules for the constants are embodied in a set of clauses called SEM, which plays the role of a set of axioms. They are:

SEM 1. $(((\mathrm{A}x)y)z) = ((xz)(yz))$
SEM 2. $((\mathrm{K}x)y) = x$
SEM 3. $(\mathrm{I}x) = x$
SEM 4. $x \neq \mathrm{TRUE}\ x \neq \mathrm{FALSE}$
SEM 5. $x \neq \mathrm{TRUE}\ (\mathrm{NOT}\ x) = \mathrm{FALSE}$
SEM 6. $x = \mathrm{TRUE}\ (\mathrm{NOT}\ x) \neq \mathrm{FALSE}$
SEM 7. $x \neq \mathrm{FALSE}\ (\mathrm{NOT}\ x) = \mathrm{TRUE}$
SEM 8. $x = \mathrm{FALSE}\ (\mathrm{NOT}\ x) \neq \mathrm{TRUE}$
SEM 9. $x \neq \mathrm{TRUE}\ y \neq \mathrm{TRUE}\ ((\mathrm{AND}\ x)y) = \mathrm{TRUE}$
SEM 10. $x \neq \mathrm{TRUE}\ y \neq \mathrm{FALSE}\ ((\mathrm{AND}\ x)y) = \mathrm{FALSE}$
SEM 11. $x \neq \mathrm{FALSE}\ y \neq \mathrm{TRUE}\ ((\mathrm{AND}\ x)y) = \mathrm{FALSE}$
SEM 12. $x \neq \mathrm{FALSE}\ y \neq \mathrm{FALSE}\ ((\mathrm{AND}\ x)y) = \mathrm{FALSE}$
SEM 13. $((\mathrm{AND}\ x)y) \neq \mathrm{TRUE}\ x = \mathrm{TRUE}$
SEM 14. $((\mathrm{AND}\ x)y) \neq \mathrm{TRUE}\ y = \mathrm{TRUE}$
SEM 15. $((\mathrm{AND}\ x)y) \neq \mathrm{FALSE}\ x = \mathrm{FALSE}\ y = \mathrm{FALSE}$
SEM 16. $x \neq \mathrm{TRUE}\ y \neq \mathrm{TRUE}\ ((\mathrm{OR}\ x)y) = \mathrm{TRUE}$
SEM 17. $x \neq \mathrm{TRUE}\ y \neq \mathrm{FALSE}\ ((\mathrm{OR}\ x)y) = \mathrm{TRUE}$
SEM 18. $x \neq \mathrm{FALSE}\ y \neq \mathrm{TRUE}\ ((\mathrm{OR}\ x)y) = \mathrm{TRUE}$
SEM 19. $x \neq \mathrm{FALSE}\ y \neq \mathrm{FALSE}\ ((\mathrm{OR}\ x)y) = \mathrm{FALSE}$
SEM 20. $((\mathrm{OR}\ x)y) \neq \mathrm{FALSE}\ x = \mathrm{FALSE}$
SEM 21. $((\mathrm{OR}\ x)y) \neq \mathrm{FALSE}\ y = \mathrm{FALSE}$
SEM 22. $((\mathrm{OR}\ x)y) \neq \mathrm{TRUE}\ x = \mathrm{TRUE}\ y = \mathrm{TRUE}$
SEM 23. $x \neq \mathrm{TRUE}\ y \neq \mathrm{TRUE}\ ((\mathrm{IMPLIES}\ x)y) = \mathrm{TRUE}$
SEM 24. $x \neq \mathrm{TRUE}\ y \neq \mathrm{FALSE}\ ((\mathrm{IMPLIES}\ x)y) = \mathrm{FALSE}$

SEM 25. $x \neq \text{FALSE}\ y \neq \text{TRUE}\ ((\text{IMPLIES}\ x)y) = \text{TRUE}$
SEM 26. $x \neq \text{FALSE}\ y \neq \text{FALSE}\ ((\text{IMPLIES}\ x)y) = \text{TRUE}$
SEM 27. $((\text{IMPLIES}\ x)y) \neq \text{FALSE}\ x = \text{TRUE}$
SEM 28. $((\text{IMPLIES}\ x)y) \neq \text{FALSE}\ y = \text{FALSE}$
SEM 29. $((\text{IMPLIES}\ x)y) \neq \text{TRUE}\ x = \text{FALSE}\ y = \text{TRUE}$
SEM 30. $x \neq y\ ((\text{EQUAL}\ x)y) = \text{TRUE}$
SEM 31. $x = y\ ((\text{EQUAL}\ x)y) = \text{FALSE}$
SEM 32. $((\text{EQUAL}\ x)y) \neq \text{TRUE}\ x = y$
SEM 33. $((\text{EQUAL}\ x)y) \neq \text{FALSE}\ x \neq y$
SEM 34. $x \neq \text{TRUE}\ (((\text{IF}\ x)y)z) = y$
SEM 35. $x \neq \text{FALSE}\ (((\text{IF}\ x)y)z) = z$
SEM 36. $(\text{ALL}\ x) \neq \text{TRUE}\ (xy) = \text{TRUE}$
SEM 37. $(xy) \neq \text{TRUE}\ (\text{EXISTS}\ x) = \text{TRUE}$
SEM 38. $(x(\text{FAIL}\ x)) = \text{TRUE}\ (\text{ALL}\ x) = \text{TRUE}$
SEM 39. $(\text{EXISTS}\ x) \neq \text{TRUE}\ (x(\text{CHOICE}\ x)) = \text{TRUE}$
SEM 40. $(\text{EXISTS}\ x) \neq \text{FALSE}\ (xy) = \text{FALSE}$
SEM 41. $(x(\text{CHOICE}\ x)) \neq \text{FALSE}\ (\text{EXISTS}\ x) = \text{FALSE}$
SEM 42. $(xy) \neq \text{FALSE}\ (\text{ALL}\ x) = \text{FALSE}$
SEM 43. $(\text{ALL}\ x) \neq \text{FALSE}\ (x(\text{FAIL}\ x)) = \text{FALSE}$
SEM 44. $x = x$
SEM 45. $x \neq y\ y = x$
SEM 46. $x \neq y\ y \neq z\ x = z$
SEM 47. $x \neq y\ u \neq v\ (xu) = (yv)$.

Any particular theorem-proving problem can then be treated by writing a set PROB of clauses in this language and seeking to deduce $\square$ (the empty clause) from the set: SEM$\cup$PROB. The deduction can proceed according to any valid principles of inference which apply to equality clauses. In particular the resolution principle may be used as sole principle; or the resolution principle together with paramodulation (Robinson and Wos 1969); or Sibert's system (Sibert 1969); or the E-resolution system of Morris (1969). In any of these systems, if SEM$\cup$PROB is unsatisfiable (in the usual first-order sense) then a deduction of $\square$ is automatically obtainable from SEM$\cup$PROB as premises; and conversely.

The underlying assumption here is that the (first-order) unsatisfiability of SEM$\cup$PROB is equivalent to the (intuitive, higher order) unsatisfiability of PROB alone. Whether or not this is so depends on the 'correctness' of the set SEM as a specification of the fundamental semantics. For our present purposes we will simply postulate the correctness of SEM, and confine our attention to the problem of designing suitable deductive machinery.

The various deductive systems mentioned above, while theoretically adequate, are no use in practice. The difficulty lies in the fact that the deductions produced in any of these systems are too long. Their length is caused by the very small size of the individual inferences they contain; they correspond to a 'micro' level of analysis of the reasoning, in which each 'macro' inference is broken down into a sequence of extremely elementary steps.

The way to deal with this problem seems clear: identify the 'macros', and set up deductive machinery in which they are the basic inference principles.

§4. An example will illustrate the ideas set forth in the previous section.

We are to show that the statement:

for all y, if (Ry) then (yM) (1)

follows from the statements:

for all x, if (Qx) then (xM); (2)

for all p, if (pQ) then (pR). (3)

We begin by constructing terms in our language corresponding to each statement. This is conveniently done in two stages: first, construct a term using lambda-notation; then apply Schönfinkel's procedure. For example, (1) first becomes:

(ALL λy((IMPLIES (Ry))(yM))) (4)

and then, by Schönfinkel's procedure:

(ALL((A((A(K IMPLIES))((A(KR))I)))((AI)(KM)))). (5)

In similar fashion (2) and (3) become respectively:

(ALL((A((A(K IMPLIES))((A(KQ))I)))((AI)(KM)))); (6)

(ALL((A((A(K IMPLIES))((AI)(KQ))))((AI)(KR)))). (7)

Now in order to prove that (1) follows from (2) and (3) we will *assert* (2) and (3) and *deny* (1), and seek to deduce a contradiction. In our language this is done by equating the corresponding terms to TRUE and to FALSE and asserting the equations. This yields three unit clauses, comprising PROB:

PROB 1. (ALL((A((A(K IMPLIES))((A(KR))I)))((AI)(KM)))) = FALSE

PROB 2. (ALL((A((A(K IMPLIES))((A(KQ))I)))((AI)(KM)))) = TRUE

PROB 3. (ALL((A((A(K IMPLIES))((AI)(KQ))))((AI)(KR)))) = TRUE

We now deduce $\square$ from the set SEM$\cup$PROB, the deduction comprising a set of clauses labelled DED. From SEM 36 and PROB 3 we obtain, by *resolution*:

DED 1. (((A((A(K IMPLIES))((AI)(KQ))))((AI)(KR)))y) = TRUE

whence we immediately infer, by *normalization*:

DED 2. ((IMPLIES(yQ))(yR)) = TRUE.

What is *normalization*? It is a 'macro' inference principle, which consists of *repeatedly applying the equations* SEM 1, SEM 2 and SEM 3 *until no further applications are possible.* In the present case the 'micro' steps by which DED 2 is obtained from DED 1 are:

DED 1. (((A((A(K IMPLIES))((AI)(KQ))))((AI)(KR)))y) = TRUE

⇒DED 1.1. ((((A(K IMPLIES))((AI)(KQ)))y)(((AI)(KR))y)) = TRUE
using SEM 1;

⇒DED 1.2. (((((K IMPLIES)y)(((AI)(KQ))y))(((AI)(KR))y)) = TRUE
using SEM 1;

⇒DED 1.3. ((IMPLIES(((AI)(KQ))y))(((AI)(KR))y)) = TRUE
using SEM 2;

⇒DED 1.4. ((IMPLIES((Iy)((KQ)y)))(((AI)(KR))y)) = TRUE
using SEM 1;

⇒DED 1.5. ((IMPLIES(y((KQ)y)))(((AI)(KR))y)) = TRUE
using SEM 3;

⇒DED 1.6. ((IMPLIES(yQ))(((AI)(KR))y)) = TRUE
using SEM 2;

⇒DED 1.7. ((IMPLIES(yQ))((Iy)((KR)y))) = TRUE
using SEM 1;
⇒DED 1.8. ((IMPLIES(yQ))(y((KR)y))) = TRUE
using SEM 3;
⇒DED 2. ((IMPLIES(yQ))(yR)) = TRUE
using SEM 2.

Normalization yields exactly one conclusion when applied to any clause, and corresponds, in the Schönfinkel notation, to *conversion to normal form* in the lambda calculus. It is exceedingly easy to carry out in the machine.

Continuing with the deduction, we obtain from DED 2 and SEM 29, by *resolution*:

DED 3. (yQ) = FALSE(yR) = TRUE.

whence, *resolving* with SEM 4, we get:

DED 4. (yQ) ≠ TRUE(yR) = TRUE

and *resolving* with SEM 4 again:

DED 5. (yQ) ≠ TRUE(yR) ≠ FALSE.

Our next inference involves another 'macro' principle. From PROB 1 we obtain, by *abstraction*, the clause:

DED 6. (((A(K ALL))((A((A(KA))((A(K(A(K IMPLIES))))
((A((A(KA))((A(KK))I)))(KI)))))(K((AI)(KM)))))R) = FALSE

and by *abstraction*, likewise, from PROB 2, we get:

DED 7. (((A(K ALL))((A((A(KA))((A(K(A(K IMPLIES))))
((A((A(KA))((A(KK))I)))(KI)))))(K((AI)(KM)))))Q) = TRUE.

What is *abstraction*? One way to characterize it is to say that it is just *the reverse of normalization*: equations SEM 1, SEM 2, SEM 3 being applied repeatedly, in a chain of substitutions. However, it is the *right hand* sides of these equations, not the left hand sides, which get replaced; just the opposite way round from the normalization replacements. For example, the chain of 'micro' steps by which DED 6 is reached from PROB 1 is shown in figure 1.

In the sequence of equality inferences we have indicated which subterm is replaced in each line, and which term replaces it to form the next line; and in each replacement we have indicated by which of SEM 1, SEM 2, or SEM 3 these two terms are equal. It will be noted that this same chain, *when read from bottom to top*, is a normalization of DED 6; that is, PROB 1 *can be inferred from* DED 6 *by normalization.*

A theorem-proving program based on elementary equality inferences would have to 'discover' this chain of steps by isolating it from all other chains of elementary inferences. In fact there is something quite special about this particular chain, but what is special about it cannot be expressed at the 'micro' level.

The overall transaction in this inference is *the solution of the following problem*:

find $\mathscr{F}$ such that:

(ALL((A((A(K IMPLIES))((A(KR))I)))((AI)(KM)))) = ($\mathscr{F}$R).

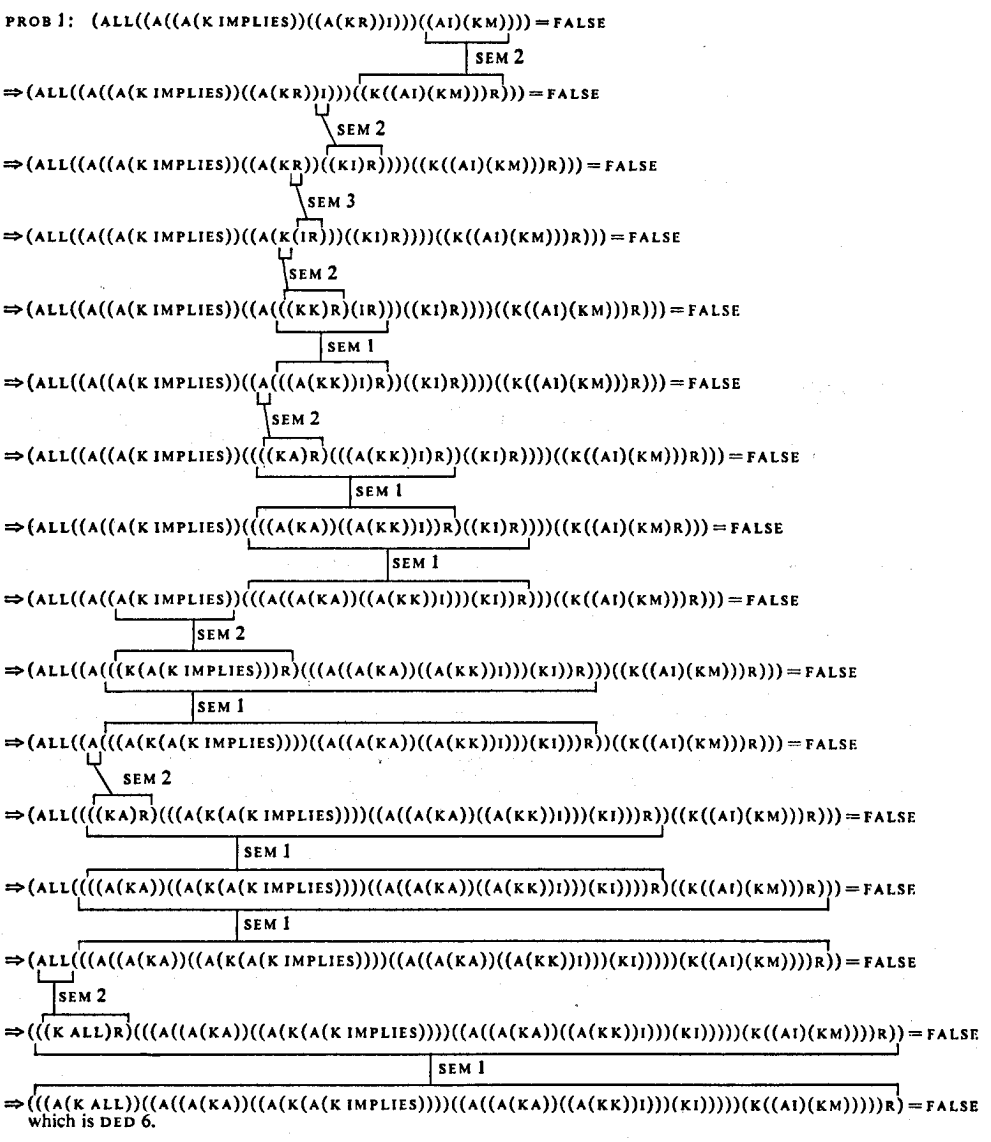

Figure 1

The left hand side of this equation is the left hand side of PROB 1. The problem intuitively is therefore: *express* PROB 1 *as*: $(\mathscr{F}\,\mathrm{R}) = \mathrm{FALSE}$. This amounts to asking 'what does the left hand side of PROB 1 say about R?' The left hand side of DED 6 has the form $(\mathscr{F}\,\mathrm{R})$, and its $\mathscr{F}$ in fact solves the problem.

Once the matter is seen in this light, however, a direct solution presents itself: first write

(ALL((A((A(K IMPLIES))((A(KR))I)))((AI)(KM))))

as: (λx(ALL((A((A(K IMPLIES))((A(Kx))I)))((AI)(KM))))R)

and then *apply Schönfinkel's algorithm to eliminate the lambda notation.* The reader will easily verify that this produces the left hand side of DED 6.

The inference principle we are here calling *abstraction* can therefore be stated at the 'macro' level as follows: *from a clause* $C(t)$ (*i.e., a clause* C *in which there is a particular occurrence of a term* t *singled out for attention*) *infer the clause* $C(t')$ (*i.e., the result of replacing the singled-out occurrence of* t *by an occurrence of the term* t'); *where* t' *is the result of applying Schönfinkel's procedure to the expression* $(\lambda x t\{x/s\}s)$, *where* s *is a constant occurring in* t.

(In the above, $t\{x/s\}$ is the result of substituting the variable x for each occurrence in t of the constant s.)

In the application of this rule which we have been examining, $C(t)$ is PROB 1; t is its left hand side:

(ALL((A((A(K IMPLIES))((A(KR))I)))((AI)(KM))));

and s is R. The reader will easily verify that $C(t')$ is then DED 6.

There are only finitely many ways in which abstraction can be applied to a given clause C, corresponding to the finitely many ways in which t and s can be chosen. For each choice of t and s, the resulting clause is uniquely determined.

With this explanation of abstraction, the reader is now in a position to check that the next step in our deduction, DED 7, is indeed obtained by abstraction from PROB 2. That is, PROB 2 is C, its left hand side is t, and Q is s. It follows that the result is as stated.

The deduction ends with

DED 8. □

which follows by *hyper-resolution* from DED 5, DED 6, and DED 7.

§5. The example of the previous section was also used in Robinson (1969). The main point of the proof is to spot that what (2) says about Q, (1) says about R; while (3) states that whatever can be said truly about Q can also be said truly about R. The abstraction rule is thus crucial in automatically generating the proof, and corresponds to a characteristically 'human' capability, namely, the ability to 'spot what a sentence says' about each thing which is mentioned in the sentence, and to see that some other sentence says the same about some other thing.

§6. The relation between Schönfinkel notation and the lambda-calculus notation is quite closely analogous to the relation between machine language and source language, in the context of programming. Schönfinkel's procedure, in this analogy, corresponds to compiling. Rules of inference such as normalization and abstraction are like routines written in machine code, whose function is viewed holistically at the source language level, yet analysed in detail at the machine level.

It would clearly be very useful to have, in terms of this analogy, a *decompilation* procedure – a procedure which translates a given piece of Schönfinkel notation into lambda notation by reversing the direction of application of the three replacement rules (6), (7) and (8) of section 1. There appears to be no difficulty about designing such a procedure. It would, for example, translate DED 6 into:

$$(\lambda x(\text{ALL } \lambda y((\text{IMPLIES } (xy))(y\text{M})))\text{R}) = \text{FALSE}.$$

§7. It is hoped that this brief discussion of the use of Schönfinkel notation to express higher order theorem-proving problems within first-order equation calculi will serve as a starting point for further investigations. There are a number of interesting questions that arise:

(1) What, besides normalization and abstraction, are appropriate 'macro' rules of inference? For example, one might well wish to broaden the notion of 'computing out' to cover not only the 'evaluation' of A, K and I – which is essentially what normalization is – but also the evaluation of other functions, such as the boolean functions, the standard arithmetic functions, or indeed any functions for which there exists information enough in the system to replace *applications* of them by the *results* of those applications.

(2) Is SEM adequate? There may be further constants which ought to be included and axiomatized as 'system constants' – e.g., CAR, CDR, CONS, and ATOM, of the LISP system. It may be that there are better ways of axiomatizing the present set of system constants than those incorporated in the present SEM.

(3) Ought we to reimpose the *type* classification (*see* Robinson 1969) or not? What purposes does this classification really serve?

(4) Might it be useful to allow relation symbols other than *equality*, and function symbols other than *application*, into the first-order calculi we use to express higher order problems? If so, what relations and functions should be represented? Usefulness apart, is it *theoretically necessary* to introduce further notions?

Acknowledgement

The work described was completed while the author held a Senior Visiting Fellowship in the Metamathematics Unit, Edinburgh University, financed by the Science Research Council.

REFERENCES

Curry, H.B. & Feys, R. (1958) *Combinatory Logic, Volume I.* Amsterdam: North Holland Publishing Company.

Morris, J.B. (1969) E-Resolution: extension of resolution to include the equality relation. *Proceedings of the First International Joint Conference on Artificial Intelligence.* Washington D.C.

Robinson, G., & Wos, L. (1969) Paramodulation and theorem-proving in first-order theories with equality. *Machine Intelligence 4*, pp. 135–50 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Robinson, J.A. (1969) Mechanizing higher order logic. *Machine Intelligence 4*, pp. 151–70 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Rosenbloom, P.C. (1950) *The Elements of Mathematical Logic.* New York: Dover Publications.

Schönfinkel, M. (1924) Uber die Bausteine der mathematischen Logik. *Mathematische Annalen*, **92**, 305–16. English version entitled: On the building blocks of mathematical logic. In *From Frege to Godel: A Source Book in Mathematical Logic*, pp. 355–66 (ed. van Heijenoort, J.). Harvard University Press, 1967.

Sibert, E.E. (1969) A machine-oriented logic incorporating the equality relation. *Machine Intelligence 4*, pp. 103–34 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.