Lambda Logic

Michael Beeson [*]

San José State University, San Jose, CA, USA

Abstract

Lambda logic is the union of first order logic and lambda calculus. We prove basic metatheorems for both total and partial versions of lambda logic. We use lambda logic to state and prove a soundness theorem allowing the use of second order unification in resolution, demodulation, and paramodulation in a first-order context.

KEYWORDS: automated deduction, computer proofs, unification, second order, Otter

1. Introduction

The twentieth century saw the flowering of first order logic, and the invention and development of the lambda calculus. When the lambda calculus was first developed, its creator (Alonzo Church) intended it as a foundational system, i.e., one in which mathematics could be developed and with the aid of which the foundations of mathematics could be studied. His first theories were inconsistent, just as the first set theories had been at the opening of the twentieth century. Modifications of these inconsistent theories never achieved the fame that modifications of the inconsistent set theories did. Instead, first order logic came to be the tool of choice for formalizing mathematics, and lambda calculus is now considered as one of several tools for analyzing the notion of algorithm.

The point of view underlying lambda logic is that lambda calculus is a good tool for representing the notion of function, not only the notion of computable function. First-order logic can treat functions by introducing function symbols for particular functions, but then there is no way to construct other functions by abstraction or recursion. Of course, one can consider set theory as a special case of first order logic, and define functions in set theory as univalent functions, but this requires building up a lot of formal machinery, and has other disadvantages as well. It is natural to consider combining lambda calculus with logic. That was

[*]Research supported by NSF grant number CCR-0204362.

done long ago in the case of typed logics; for example Gödel's theory T had what amounted to the ability to define functions by lambda-abstraction. But typed lambda calculus lacks the full power of the untyped (ordinary) lambda calculus, as there is no fixed-point theorem to support arbitrary recursive definitions.

In this paper, we combine ordinary first order lambda calculus with ordinary first order logic to obtain systems we collectively refer to as lambda logic. We are not the first to define or study similar systems.[†] The applicative theories proposed by Feferman in (6) are similar in concept. They are, however, different in some technical details that are important for the theorems proved here. Lambda logic is also related to the systems of illative combinatory logic studied in (2), but these are stronger than lambda logic.

Both ordinary and lambda logic can be modified to allow "undefined terms". In the context of ordinary logic this has been studied in (8; 4; 3). In the context of applicative theories, (3) defined and studied "partial combinatory algebras"; but application in the λ-calculus is always total. Moggi (7) was apparently the first to publish a definition of partial lambda calculus; see (7) for a thorough discussion of different versions of partial lambda calculus and partial combinatory logic.

The system Otter2 (5) uses second-order unification in an untyped context to enhance the capabilities of the automated theorem prover Otter. Inference rules such as resolution and paramodulation are allowed to use second-order unification instead of only first order unification. Higher order unification has in the past been used with typed systems. Lambda logic provides a theoretical foundation for its application in an untyped setting, as we show with a soundness theorem in this paper. Lambda logic answers the question, "In what formal system can the proofs produced by Otter2 be naturally represented?".

2. Syntax

We will not repeat the basics of first order logic or the basics of lambda calculus, which can be found in (9) and (1), respectively. We start with any of the usual formulations of first order logic with equality. This includes a stock of variables, constants, and function symbols, from which one can build up terms and formulas as usual. In addition, there is a distinguished unary function symbol Ap. As usual in lambda calculus, we optionally abbreviate $Ap(x, y)$ to $x(y)$ or even xy, with left association understood in expressions like xyz, which means $(xy)z$. But we do this less often than is customary in lambda calculus, since we also have $f(x)$ where f is a function symbol; and this is not the same as $Ap(f, x)$. Of course, theoretically there is only one syntactically correct way of reading the abbreviated term.

Lambda-terms are created by the following term formation rule:

If t is a term and x is a variable, then $\lambda x.\, t$ is a term.

The notion of a variable being free in a term is defined as usual: quantifiers

[†]For example, John McCarthy told me that he lectured on such systems years ago, but never published anything.

and lambda both bind variables. The notion of substitution is defined as in (1). The notation is $t[x := s]$ for the result of substituting s for the free variable x in t. Note that this is literal substitution, i.e. $t[x := s]$ does not imply the renaming of bound variables of t to avoid capture of free variables of s. We also define $t[x ::= s]$, which does imply an algorithmic renaming of bound variables of t to avoid capture of free variables of s.

Alpha-conversion means renaming a bound variable in such a way that there is no capture of formerly free variables by the renamed variable. The induced equivalence relation on terms is called alpha-equivalence. Beta-reduction is defined as usual.

Example: in first order logic we can formulate the theory of groups, using a constant e for the identity, and function symbols for the group operation and inverse. The use of infix notation $x \cdot y$ can either be regarded as official or as an informal abbreviation for $m(x, y)$, just as it can in first order logic. If we formulate the same theory in lambda logic, we use a unary predicate G for the group, and relativize the group axioms to that predicate, just as we would do in first order logic if we needed to study a group and a subgroup. Then in lambda logic we can define the commutator:

$$c \;:=\; \lambda x, y.((i(x) \cdot i(y)) \cdot x) \cdot y$$

and then we can derive the following in lambda logic:

$$G(x) \wedge G(y) \rightarrow c(x, y) = ((i(x) \cdot i(y)) \cdot x) \cdot y$$

3. Axioms

Lambda logic can be formulated using any of the usual approaches to predicate calculus. We distinguish the sequent-calculus formulation, the Hilbert-style formulation, and the resolution formulation. For definiteness we choose the Hilbert-style formulation as the definition (say as formulated in (9), p. 20), for the standard version. There will then be two further variants for different logics of partial terms, but these will be discussed in another section.

(*Prop*) propositional axioms

(Q) standard quantifier axioms

(α) $t = s$ if t and s are alpha-equivalent. The alternative would be to include the axiom only in case t alpha-converts to s.

(β) $Ap(\lambda x.\, t, s) \cong t[x ::= s]$

(ξ) (*weak extensionality*) $\forall x(Ap(t, x) \cong Ap(s, x)) \rightarrow \lambda x.\, Ap(t, x) \cong \lambda x.\, Ap(s, x)$

(*non-triviality*) $\mathbf{T} \neq \mathbf{F}$, where $\mathbf{T} = \lambda x \lambda y.\, x$ and $\mathbf{F} = \lambda x \lambda y.\, y$

The following additional formulae are not part of lambda logic, but are of interest.

(η) (*extensionality*) $\lambda x.\, Ap(t, x) \cong t$.

(*AC*) (*Axiom of Choice*) $\forall x \exists y P(x, y) \rightarrow \exists f \forall x P(x, Ap(f, x))$.

4. Semantics

There is a standard definition of λ-model that is used in the semantics of the lambda calculus (see (1), p. 86, with details on p. 93). There is also a well-known notion of a model of a first order theory. In this section our goal is to define the concept *M is a model of the theory T in lambda logic* in such a way that it will imply that, neglecting λ, M is a first order model of T, and also, neglecting the function symbols other than Ap, M is a λ-model.

The cited definition of λ-model involves the notion of terms, which we shall call M-terms, built up from Ap and constants c_a for each element a of the model. It requires the existence, for each term t of this kind, and each variable x, of another M-term $\lambda^*(x,t)$ such that M will satisfy

$$Ap(\lambda^*(x,t),x) = t.$$

Note that this does not yet make sense, as we must first define the notion of "the interpretation of a term t in M". We cannot simply refer to (1) for the definition, since we need to extend this definition to the situation in which we have a theory T with more function symbols than just Ap, although the required generalization is not difficult.

We first define a λ-structure. As usual in first order logic we sometimes use "M" to denote the carrier set of the structure M; and we use f_M or $\bar{f}$ for the function in the structure M that serves as the interpretation of the function symbol f, but we sometimes omit the bar if confusion is unlikely. $(M, \lambda*)$ is a λ-structure for T if (1) M is a structure with a signature containing all the function symbols and constants occurring in T, and another binary function Ap_M to serve as the interpretation of Ap, and (2) there is an operation λ^* on pairs (x,t), where t is an M-term and x is a variable, producing an element $\lambda^*(x,t)$ of M.

If (M,λ^*) is a λ-structure for T, then by a *valuation* we mean a map g from the set of variables to (the carrier set of) M. If g is a valuation, and $v = v_1,\ldots,v_n$ is a list (vector) of variables, then by $g(v)$ we mean $g(v_1),\ldots,g(v_n)$. If t is a term, then by $t[v := g(v)]$ we mean the M-term resulting from replacing each variable v_i by the constant $c_{g(v_i)}$ for the element $g(v_i)$ of M. If g is a valuation, then we can then extend g to the set of terms by defining

$$\begin{aligned} g[f(t_1,\ldots,t_n)] &= \bar{f}(g(t_1),\ldots,g(t_n)) \\ g[\lambda x.t] &= \lambda^*(x, t[v := g(v)]) \end{aligned}$$

Now we have made sense of the notion "the interpretation of term t under valuation g". We define the notion $M \models \phi$ as it is usually defined for first order logic. We are now in a position to define λ-model. This definition coincides with that in (1) in case T has no other function symbols than Ap.

Definition (λ-model): (M,λ^*) is a λ-model of a theory T in lambda logic if (M,λ^*) satisfies the axioms α, β, and ξ, and M satisfies the axioms of T.

5. Basic Metatheorems

Define the relation $t \equiv s$ on terms of T to mean that t and s have a common reduct (using β and α reductions). The Church-Rosser theorem for λ calculus ((1), p. 62) implies that this is an equivalence relation, when the language includes only Ap and λ. The following theorem says that adding additional function symbols does not destroy the Church-Rosser property.

THEOREM 5.1: *The Church-Rosser theorem is valid for lambda logic.*

Proof. For each function symbol f we introduce a constant $\bar{f}$. We can then eliminate f in favor of $\bar{f}$ as follows. For each term t we define the term t° as follows:

$$\begin{array}{rcll}
x^\circ & = & x & \text{for variables } x \\
c^\circ & = & c & \text{for variables } c \\
f(t)^\circ & = & Ap(\bar{f}, t^\circ) & \\
f(t,r)^\circ & = & Ap(Ap(\bar{f}, t^\circ), r^\circ)) & \\
Ap(t,r)^\circ & = & Ap(t^\circ, r^\circ) & \\
(\lambda x.\, t)^\circ & = & \lambda x.\, t^\circ &
\end{array}$$

and similarly for functions of more than two arguments. Since there are no reduction rules involving the new constants, t reduces to q if and only if t° reduces to q°. Moreover, if t° reduces to v, then v has the form u° for some u. (Both assertions are proved by induction on the length of the reduction.) Suppose t reduces to q and also to r. Then t° reduces to q° and to r°. By the Church-Rosser theorem, q° and r° have a common reduct v, and v is u° for some u, so q and r both reduce to u. That completes the proof.

The following theorem is to lambda logic as Gödel's completeness theorem is to first order logic. As in first order logic, a theory T in lambda logic is called *consistent* if it does not derive any contradiction. In this section, we will prove the lambda completeness theorem: any consistent theory has a λ-model. First, we need some preliminaries.

THEOREM 5.2 (LAMBDA COMPLETENESS THEOREM): *Let T be a consistent theory in lambda logic. Then T has a λ-model.*

Remark. There is a known "first order equivalent" of the notion of λ-model, namely *Scott domain.* See (1)(section 5.4). However, we could not use Scott domains to reduce the lambda completeness theorem to Gödel's first order completeness theorem, because there is no syntactic interpretation of the theory of Scott domains in lambda logic.

Proof of the completeness theorem. The usual proof (Henkin's method) of the completeness theorem, as set out for example in (9) pp. 43-48, can be imitated for lambda logic. If T does not contain infinitely many constant symbols, we begin by adding them; this does not destroy the consistency of T since in any proof of contradiction, we could replace the new constants by variables not occurring elswhere in the proof. We construct the "canonical structure" M for a theory T. The elements of M are equivalence classes of closed terms of T under the equivalence relation of provable equality: $t \sim r$ iff $T \vdash t = r$. Let $[t]$ be the equivalence class of t. We define the interpretations of constants, function symbols, and predicate symbols in M as follows:

$$\begin{aligned} c^M &= [c] \\ f^M([t_1], \ldots, [t_n]) &= [f(t_1, \ldots, t_n)] \\ P^M([t_1], \ldots, [t_n)] &= [P(t_1, \ldots, t_n)] \end{aligned}$$

In this definition, the right sides depend only on the equivalence classes $[t_i]$ (as shown in (9), p. 44).

Exactly as in (9) one then verifies that M is a first order model of T. To turn M into a λ-model, we must define $\lambda^*(x, [t])$, where x is a variable and t is an M-term, i.e. a closed term with parameters from M. The "parameters from M" are constants $c_[q]$ for closed terms q of T. If t is an M-term t, let $[t]^\circ$ be the closed term of T obtained from t by replacing each constant $c_[q]$ by a closed term q in the equivalence class $[q]$. Then $[t]^\circ$ is well-defined. Define $\lambda^*(x, [t]) = [\lambda x.\, [t]^\circ]$. By axiom (ξ), this is a well-defined operation on equivalence classes: if T proves $t = s$ then T proves $[t]^\circ = [s]^\circ$ and hence $\lambda x.\, [t]^\circ = \lambda x.\, [s]^\circ$.

We verify that the axioms of lambda logic hold in M. First, the (β) axiom: $Ap(\lambda x.\, t, r) = t[x := r]$. It suffices to consider the case when t has only x free. The interpretation of the left side in M is the equivalence class of $Ap(\lambda x.\, t, r)$. The interpretation of the right side is the class of $t[x := r]$. Since these two terms are provably equal, their equivalence classes are the same, verifying axiom (β). Now for axiom (ξ). Suppose t and s are closed terms and $Ap(t, x) = Ap(s, x)$ is valid in M. Then for each closed term r, we have tr provably equal to sr. Since T contains infinitely many constant symbols, we can select a constant c that does not occur in t or s, so $tc = sc$ is provable. Replacing the constant c by a variable in the proof, $tx = sx$ is provable. Hence by axiom (ξ), $\lambda x.\, t = \lambda x.\, s$ is probable, and hence that equation holds in M. In verifying axiom (ξ), it suffices to consider the case when s and t are closed terms. The axiom (α) holds in M since it simply asserts the equality of pairs of provably equal terms. The axiom $\mathbf{T} \neq \mathbf{F}$ holds since T does not prove $\mathbf{T} = \mathbf{F}$, because T is consistent. That completes the proof.

THEOREM 5.3 (AXIOMATIZATION OF FIRST-ORDER THEOREMS): *Let T be a first order theory, and let A be a first order sentence. Then T proves A in lambda logic if and only if for some positive integer n, T plus "there exist n distinct things" proves A in first order logic.*

Proof. First suppose A is provable from T plus "there exist n distinct things". We show A is provable in lambda logic, by induction on the length of the proof of A. Since lambda logic includes first order logic, the induction step is trivial. For the basis case we must show that lambda logic proves "there exist n distinct things" for each positive integer n. The classical constructions of numerals in lambda calculus produce infinitely many distinct things. However, it must be checked that their distinctness is provable in lambda logic. Defining numerals as on p. 130 of (1) we verify by induction on n that for all $m < n$, $\ulcorner m \urcorner \neq \ulcorner n \urcorner$ is provable in lambda logic. If $m < n+1$ then either $m = n$, in which case we are done by the induction hypothesis, or $m = n$. So what has to be proved is that for each n, lambda logic proves $\ulcorner n \urcorner \neq \ulcorner n+1 \urcorner$. This in turn is verifiable by induction on n.

Conversely, suppose that A is not provable in T plus "there exist n distinct things" for any n. Then by the completeness theorem for first order logic, there is an infinite model M of $\neg A$; indeed we may assume that M has infinitely many elements not denoted by closed terms of T. Then M can be expanded to a lambda model $\hat{M}$ satisfying the same first order formulas, by defining arbitrarily the required operation λ^* on M-terms, and then inductively defining relations $E(x,y)$ and Ap_M to serve as the interpretations of equality and Ap in $\hat{M}$. A detailed proof is available at the author's web site; space does not permit it to be reproduced here as it requires three pages.

6. Skolemization

We now consider the process of Skolemization. This is important for automated deduction, since it is used to prepare a problem for submission to a theorem prover that requires clausal form. This can be extended from ordinary logic to lambda logic, but unlike in ordinary logic, the axiom of choice is required:

Theorem 6.1 (Skolemization): *Let T be a theory in lambda logic. Then we can enlarge T to a new theory S by adding new function symbols (Skolem symbols) such that (1) the axioms of S are quantifier-free and (2) in lambda logic + AC, S and T prove the same theorems in the language of T.*

Proof. The proof is the same as for ordinary logic–we eliminate one alternating quantifier at a time. Consider $\forall x \exists y P(x,y)$. We add a new function symbol f and the (Skolem) axiom

$$\forall x \exists y P(x,y) \rightarrow \forall x P(x, f(x)).$$

In the presence of this Skolem axiom, if $\forall x \exists y P(x,y)$ is an axiom of T it can be eliminated in favor of $\forall x P(x, f(x))$, which has one quantifier fewer. It remains to prove that adding such a Skolem axiom to $T + AC$ does not add any new theorems in the language of T. By the lambda completeness theorem, it suffices to show that any λ-model of $T + AC$ can be expanded to a λ-model of the Skolem axiom.

Let (M, λ^*) be a λ-model of $T + AC$. Assume M satisfies $\forall x \exists y P(x, y)$. Since M satisfies AC, there is some u in M such that M satisfies $\forall x P(x, Ap(c_u, x))$. (Here c_u is a constant denoting the element u of M; technically λ^* is defined on M-terms.) Interpret the Skolem function symbol f as the function $x \mapsto Ap(u, x)$. Similarly, if r is any M-term possibly involving f, let r' be defined as follows:

$$\begin{aligned} r' &= r \qquad \text{if } r \text{ does not contain } f \\ f(t)' &= Ap(c_u, t') \end{aligned}$$

Now we extend λ^* to a function λ' defined on terms involving the new symbol f as well as the other symbols of T and constants for elements of M, by defining

$$\lambda'(x, t) = \lambda^*(x, t')$$

The right side is defined since t' does not contain f. Note that when t does not contain f, we have $t' = t$ and hence $\lambda^*(x, t) = \lambda'(x, t)$.

Consider the λ-structure (M', λ'), where M' is M augmented with the given interpretation of f. We claim that this λ-structure is actually a λ-model of $T + AC$ that satisfies the Skolem axiom. Formulas which do not contain the Skolem symbol f are satisfied in (M, λ^*) if and only they are satisfied in (M', λ'), since on such terms we have $\lambda^*(x, t) = \lambda'(x, t)$ as noted above. Therefore (M', λ') is a λ-model of $T + AC$, and we have already showed that it satisfies the Skolem axiom. That completes the proof.

7. The Logic of Partial Terms

In the group theory example near the end of the introduction, it is natural to ask whether $x \cdot y$ needs to be defined if x or y does not satisfy $G(x)$. In first order logic, $\cdot$ is a function symbol and hence in any model of our theory in the usual sense of first order logic, $\cdot$ will be interpreted as a function defined for all values of x and y in the model. The usual way of handling this is to say that the values of $x \cdot y$ for x and y not satisfying $G(x)$ or $G(y)$ are defined but irrelevant. For example, in first order field theory, $1/0$ is defined, but no axiom says anything about its value. As this example shows, the problem of "undefined terms" is already of interest in first order logic, and two different (but related) logics of undefined terms have been developed. We explain here one way to do this, known as the *Logic of Partial Terms* (LPT). See (4) or (3), pp. 97-99.

LPT has a term-formation operator $\downarrow$, and the rule that if t is a term, then $t \downarrow$ is an atomic formula. One might, for example, formulate field theory with the axiom $y \neq 0 \rightarrow x/y \downarrow$ (using infix notation for the quotient term). Thereby one would avoid the (sometimes) inconvenient fiction that $1/0$ is some real number, but it doesn't matter which one because we can't prove anything about it anyway; many computerized mathematical systems make use of this fiction. Taking this approach, one must then modify the quantifier axioms. The two modified axioms are as follows:

$$\forall x\, A \wedge t \downarrow \to A[x := t]$$
$$A[x := t] \wedge t \downarrow \to \exists x\, A$$

Thus from "all men are mortal", we are not able to infer "the king of France is mortal" until we show that there *is* a king of France. The other two quantifier axioms, and the propositional axioms, of first order logic are not modified. We also add the axioms $x \downarrow$ for every variable x, and $c \downarrow$ for each constant c.

In LPT, we do not assert anything involving undefined terms, not even that the king of France is equal to the king of France. The word "strict" is applied here to indicate that subterms of defined terms are always defined. LPT has the following "strictness axioms", for every atomic formula R and function symbol f. In these axioms, the x_i are variables and the t_i are terms.

$$R(t_1, \ldots, t_n) \to t_1 \downarrow \wedge \ldots \wedge t_n \downarrow$$
$$f(t_1, \ldots, t_n) \downarrow \to t_1 \downarrow \wedge \ldots \wedge t_n \downarrow$$
$$t_1 \downarrow \wedge \ldots \wedge t_n \downarrow \wedge f(x_1, \ldots, x_n) \downarrow \to f(t_1, \ldots, t_n) \downarrow$$

Remark. In LPT, while terms can be undefined, formulas have truth values just as in ordinary logic, so one never writes $R(t) \downarrow$ for a relation symbol R. That is not legal syntax.

We write $t \cong r$ to abbreviate $t \downarrow \vee r \downarrow \to t = r$. That is, For example, a special case of this schema is

$$t = r \to t \downarrow \wedge r \downarrow$$

It follows that $t \cong r$ really means "t and r are both defined and equal, or both undefined."

The equality axioms of LPT are as follows (in addition to the one just mentioned):

$$x = x$$
$$x = y \to y = x$$
$$t \cong r \wedge \phi[x := t] \to \phi[x := r]$$

8. Partial Lambda Calculus

In lambda calculus, the issue of undefined terms arises perhaps even more naturally than in first order logic, as it is natural to consider partial recursive functions, which are sometimes undefined.[‡]

[‡]There is, of course, an alternative to λ-calculus known as *combinatory logic*. Application in combinatory logic is also total, but in (3), the notion of a *partial combinatory algebra* is introduced and studied, following Feferman, who in (6) first introduced partial applicative theories. See (7) for some relationships between partial combinatory logic and partial lambda calculus

We now define the *partial lambda calculus*. It is like the lambda calculus, except that Ap is not required to be total. There can then be "undefined terms."

Partial lambda calculus is a system similar to lambda calculus, but in which Ap is not necessarily total. Since lambda calculus is a system for deducing (only) equations, the system has to be modified. We now permit two forms of statements to be deduced: $t \cong r$ and $t\downarrow$. The axioms (α), (β), and (ξ) are modified by changing $=$ to $\cong$, and the following rules for deducing $t\downarrow$ are specified:

$$x\downarrow \qquad \text{for each variable } x$$

$$\frac{t \cong s \qquad t\downarrow}{s\downarrow} \qquad \frac{t \cong s \qquad s\downarrow}{t\downarrow}$$

$$\frac{t\downarrow \qquad r\downarrow}{t[x := r]\downarrow} \qquad \lambda x.\, t\downarrow$$

Note that we do not have strictness of Ap. As an example, we have $(\lambda y.\, a)b) \cong a$, whether or not $b\downarrow$. We could have formulated a rule "strict (β)" that would require deducing $r\downarrow$ before concluding $Ap(\lambda x.\, t, r) \cong t[x := r]$, but not requiring strictness corresponds better to the way functional programming languages evaluate conditional statements. Note also that $\lambda x.\, t$ is defined, whether or not t is defined.

9. Partial Lambda Logic

Partial lambda logic results if we make similar modifications to lambda logic instead of to first order logic or lambda calculus. In particular we modify the rules of logic and the equality axioms as in LPT, add the strictness axiom, and modify the axioms (α), (β), and (ξ) by replacing $=$ with $\cong$. In LPT, $\cong$ is an abbreviation, not an official symbol; in partial lambda calculus it is an official symbol; in partial lambda logic we could make either choice, but for definiteness we choose to make it an official symbol, so that partial lambda logic literally extends both LPT and partial lambda calculus. The first three rules of inference listed above for partial lambda calculus are superfluous in the presence of LPT. The fourth one is replaced in partial lambda logic by the following axiom:

$$t\downarrow \to \lambda x.\, t\downarrow .$$

We do not apply the strictness axiom of *LPT* to the function symbol Ap. Instead we rely only on (β) for deducing theorems of the form $Ap(t, r)\downarrow$. There is one additional axiom, as in partial lambda calculus:

$$\lambda x.\, t\downarrow \qquad \text{for each term } t.$$

We review the semantics of LPT as given in (4; 3). A model consists of a set and relations on that set to interpret the predicate symbols; the function symbols are interpreted by partial functions instead of total functions. Given such a *partial* *structure* one defines simultaneously, by induction on the complexity of terms t, the two notions $M \models t\downarrow$ and t_M, the element of M that is denoted by t.

We now discuss the semantics of partial lambda logic. The definition of partial λ-model is similar to that of λ-model, except that now Ap and the other function symbols can be interpreted by partial functions instead of total functions. The function λ^* in the definition of λ-model (which takes a variable and an M-term as arguments) is required to be total, so that the axiom $\lambda x.\, t\downarrow$ will be satisfied.

Definition (Partial lambda model): (M, λ^*) is a partial λ-model of a theory T in partial lambda logic if (M, λ^*) satisfies the axioms (α), (ξ), and (β), and M satisfies all the axioms of T and LPT, except that Ap need not be strict.

The following theorem generalizes the completeness theorem for LPT to partial lambda logic.[§]

Theorem 9.1 (Lambda Completeness Theorem for LPT): *Let T be a consistent theory in partial lambda logic. Then T has a partial lambda model.*

The proof can be found in a supplement to this paper, available at the author's web site. The theorem on Skolemization also generalizes to partial lambda logic, with the same proof.

10. Soundness of Second Order Unification

We define σ to be a *second order unifier* of t and s if $t\sigma = s\sigma$ is provable in lambda logic. The usual rules of inference used in first-order logic, such as resolution and paramodulation, are extended by allowing second order unifiers. We give a proof of the soundness of such extended rules of inference. This proof does not refer to any specific algorithm for finding second order unifiers, but it does apply to such algorithms: all one has to do is verify that the algorithm in question does indeed produce second order unifiers. This theorem thus provides a theoretical basis for the practical use of such algorithms.

We define (second order) paramodulation to be the following rule: if $\alpha = \beta$ (or $\beta = \alpha$) has been deduced, and $P[x := \gamma]$ has been deduced, and for some substitution σ we have $\alpha\sigma = \gamma\sigma$, and the free variables of $\beta\sigma$ either occur in γ or are not bound in P, then we can deduce $P[x := \beta\sigma]$. This differs from the first order version of paramodulation only in the extra condition about the free variables of $\beta\sigma$.

The rules of inference that we prove sound are paramodulation (as just defined), demodulation (including β-reduction), binary resolution (using second order unification), and factoring. Specifically binary resolution allows the inference from $P|Q$ and $-S|R$ to $Q\sigma|R\sigma$ provided that $P\sigma = R\sigma$ is provable in λ-logic, or if **cases** terms are involved, in λ-D logic. Here of course Q and R

[§]In (4) there is a completeness theorem for LPT, generalizing Gödel's completeness theorem. The strictness axiom is important in the proof.

stand for lists of literals, and the literals P and S can occur in any position in the clause, not just the first position as shown. Factoring allows us to unify different literals in the same clause: specifically we can infer $P\sigma|R\sigma$ from $P|Q|R$ when $P\sigma = R\sigma$ is provable in λ-logic. It is well-known that for completeness of binary resolution, we need to allow factoring, i.e. unifying different literals in the same clause. Here we are proving only soundness, not completeness, and in Otter2, we turn second order unification off during factoring, since we seemed to get only useless inferences with it on. Nevertheless factoring is a sound inference rule, so we might as well include it in the statement of the theorem.

In (5), a second order unification algorithm is introduced in an untyped theorem prover Otter2. Since second order unification has usually been considered in a typed setting, there is a natural question about its soundness, considering that (a) the algorithm is not a subset of typed λ-unification, in that it can solve problems whose solution is not typeable, and (b) it may surprise non-experts that the axiom of choice is provable (the theorem shows there will be no more such surprises), and (c) the paramodulation rule has to be carefully formulated to work correctly with λ-bound variables. Lambda logic is the correct tool to analyze this algorithm, as shown by the generality of the soundness theorem. To verify the soundness of Otter2, we need only verify that the rules of inference implemented are special cases of the rules mentioned above; in particular the extra condition on paramodulation has been enforced, and the unifiers produced are second order unifiers in the sense defined above.[¶]

We identify a clause with the formula of λ-logic which is the disjunction of the literals of the clause. If Γ is a set of clauses, then Γ can also be considered as a set of formulas in λ-logic.

THEOREM 10.1 (SOUNDNESS OF SECOND ORDER UNIFICATION): *(i) Suppose there is a proof of clause C from a set of clauses Γ using binary resolution, factoring, demodulation (including β-reduction), and paramodulation, and the clause $x = x$, allowing second order unification in place of first order unification. Then there is a proof of C from Γ in lambda logic.*

(ii) Let P be a formula of lambda logic. Suppose there is a refutation of the clausal form of the negation of P as in part (i). Then P is a theorem of lambda logic plus the axiom of choice AC.

Proof. (We treat demodulation as an inference rule.) Ad (i): We proceed by induction on the lengths of proofs, In the base case, if we have a proof of length 0, the clause C must already be present Γ, in which case certainly $\Gamma \vdash C$ in lambda logic.

For the induction step, we first suppose the last inference is by paramodulation.

[¶]This is the case when the command *set(cases)* is not used in Otter2. Using that command enables the generation of unifiers involving **cases** terms. To analyze such unifiers we need an extension of lambda logic called lambda-D logic instead of lambda logic in the soundness theorem. Space does not permit the discussion of lambda-D logic in this paper.

Then one of the parents of the inference is an equation $\alpha = \beta$ (or $\beta = \alpha$) where the other parent ϕ has the form $\psi[x := \gamma]$ where for some substitution σ we have $\gamma\sigma = \alpha\sigma$, and the free variables of $\beta\sigma$ either occur in γ or are not bound in ψ, according to the definition of paramodulation. Then the newly deduced formula is $\psi[x := \beta\sigma]$. We have to show that this formula is derivable in lambda logic from the parents ϕ and $\alpha = \beta$. Apply the substitution σ to the derived formula ϕ. We get

$$\begin{aligned} \phi\sigma &= \psi[x := \gamma]\sigma \\ &= (\psi\sigma)[x := \gamma\sigma] \\ &= (\psi\sigma)[x := \alpha\sigma] \end{aligned}$$

Now using the other deduced equation $\alpha = \beta$, we can deduce $\alpha\sigma = \beta\sigma$ and hence we can, according to the rules of lambda logic, substitute $\beta\sigma$ for $\alpha\sigma$, provided the free variables in $\beta\sigma$ either occur already in $\alpha\sigma$ or are not bound in $\psi\sigma$. Since $\gamma = \alpha\sigma$, this is exactly the condition on the variables that makes the application of paramodulation legal. That completes the induction step in the case of a paramodulation inference.

If the last inference is by factoring, it has the form of applying a substitution σ to a previous clause. This can be done in lambda logic. (In the factoring rule, as in lambda logic, substitution must be defined so as to not permit capture of free variables by a binding context.)

If the last inference is by binary resolution, then the parent clauses have the form $P|R$ and $-Q|S$ (using R and S to stand for the remaining literals in the clauses), and substitution σ unifies P and Q. The newly deduced clause is then $R\sigma|S\sigma$. Since the unification steps are sound, $P\sigma = Q\sigma$ is provable in λ-logic. By induction hypothesis, λ-logic proves both $P|R$ and $-Q|S$ from assumptions Γ, and since the substitution rule is valid in λ-logic, it proves $P\sigma|R\sigma$ and $-Q\sigma|S\sigma$. But since $P\sigma = Q\sigma$ is provable, λ-logic proves $R\sigma|S\sigma$ from Γ. This completes the induction step, and hence the proof of (i).

If the last inference is by demodulation, the corresponding inference can be made in lambda logic using an equality axiom or (β).

Ad (ii): We take Γ to be the clausal form of the negation of P, and C to be the empty clause. The theorem follows from (i) together with the following assertion: if P is refutable in λ-logic plus AC, then the clausal form Γ of the negation of P is contradictory in λ-logic plus AC. In first order logic, this is true without AC– see the remark following the proof. However, when AC is allowed, then every formula A is equivalent to its clausal form, in the following precise sense. To bring a formula A to clausal form, we first bring it to prenex form, with the matrix in conjunctive normal form, and then use Skolem functions to eliminate the existential quantifiers. We then arrive at a formula $\forall x_1, \ldots, x_n A^o$, where A^o contains new Skolem function symbols $f_1, \ldots, f_n$. Within lambda logic, using AC, we can perform these same manipulations, but instead of introducing new Skolem function symbols, we instead show the A is equivalent to $\exists g_1, \ldots, f_m A'$,

where A' is the same as A^o except that it has $Ap(g_i(x)$ where A^o has $f_i(x)$. (Here x stands for $x_1, \ldots, x_k$ for some k depending on i.) In particular, if P is refutable, then P^o is refutable. Replacing each Skolem function term $f_i(x)$ by $Ap(g_i, x)$, where g_i is a variable, and using the law for $\exists$ in first order logic, we see that $\exists g_1, \ldots, g_m P'$ is refutable. That completes the proof.
Remark. The proof was entirely syntactical. No discussion, or even definition, of models of lambda logic was needed; but it may help to consider why AC is needed in lambda logic but not in first order logic. In first order logic, every model of $\forall x \exists y A(x, y)$ can be expanded to a model of $\forall x A(x, f(x))$ by interpreting the new function symbol f appropriately. But if we are dealing with λ-models, this is no longer the case. The function required may not be represented by a λ-term in the model. For example, it could happen that all possible interpretations of the Skolem function are non-recursive, but all operations representable in the model are recursive.

Corollary 10.1: *If the system Otter2 (5) refutes the clausal form of the negation of P, using the clause $x = x$ and some other axioms Γ, but with the option to generate* **cases** *terms in unification off, then P is provable from the conjunction of Γ in lambda logic plus AC.*

Proof. The second order unification algorithm implemented in Otter2 produces second order unifiers; the restriction on the application of paramodulation is also implemented; the rules of hyperresolution, etc. are all theoretically reducible to binary resolution (they are only used to make the proof search more efficient). Otter2 also uses demodulation β-reduction to post-process (simplify) deduced conclusions. These correspond to additional infererences in lambda logic using the equality axioms and the β axiom. Note: we have not formally verified any specific computer program; this is an informal proof.

Some of the interesting applications of second order unification involve undefined terms, and therefore we need a soundness theorem relative to partial lambda logic. When we use LPT in the resolution-based theorem prover Otter2, we replace the axiom $x = x$ by the clause $-E(x)|x = x$, where we think of $E(t)$ as representing $t\downarrow$. We also add the clause $x \neq x|E(x)$, thus expressing that $t\downarrow$ is equivalent to $t = t$. The soundness theorem takes the following form:

Theorem 10.2 (Soundness of second order unification for LPT):
(i) Suppose there is a proof of clause C from a set of clauses Γ using binary resolution, factoring, demodulation (including β-reduction), and paramodulation, the clauses $x = x|-E(x)$ and $-E(x)|x = x$, and clauses expressing the strictness axioms, allowing second order unification in place of first order unification. Then there is a proof of C from Γ in partial lambda logic.
(ii) Let P be a formula of partial lambda logic. Suppose there is a refutation of the clausal form of the negation of P as in part (i). Then P is a theorem of partial lambda logic plus AC.

Proof. The proof is similar to the proof for lambda logic, except for the treatment of β-reduction steps. When β-reduction is applied, we only know that $\cong$ holds in partial lambda logic. But since in partial lambda logic, we have the substitutivity axioms for $\cong$ as well as for $=$, it is still the case in partial lambda logic that if $P[x := Ap(\lambda z.\, t, r)]$ is used to deduce $P[x := t[z := r]]$, and if (by induction hypothesis) the former is derivable in partial lambda logic plus AC, then so is the latter. For example if $Ap(\lambda z.\, a, \Omega) = a$ is derivable, then $a = a$ is derivable. Each of these formulas is in fact equivalent to $a\downarrow$.

COROLLARY 10.2: *If the system Otter2 (5) refutes the clausal form of the negation of P, using the clauses $x = x| - E(x)$ and $-E(x)|x = x$, and some other axioms Γ, but with the option to generate* **cases** *terms in unification off, then P is provable from the conjunction of Γ in partial lambda logic plus AC.*

References

1. Barendregt, H., *The Lambda Calculus: Its Syntax and Semantics*, Studies in Logic and the Foundations of Mathematics **103**, Elsevier Science Ltd. Revised edition (October 1984).

2. Barendregt, H., Bunder, M., and Dekkers, W., Completeness of two systems of illative combinatory logic for first order propositional and predicate calculus *Archive für Mathematische Logik* **37**, 327–341, 1998.

3. Beeson, M., *Foundations of Constructive Mathematics*, Springer-Verlag, Berlin Heidelberg New York (1985).

4. Beeson, M., Proving programs and programming proofs, in: Barcan, Marcus, Dorn, and Weingartner (eds.), *Logic, Methodology, and Philosophy of Science VII, proceedings of the International Congress, Salzburg, 1983*, pp. 51-81, North-Holland, Amsterdam (1986).

5. Beeson, M., Otter Two System Description, submitted to IJCAR 2004.

6. S. Feferman, Constructive theories of functions and classes, pp. 159-224 in: M. Boffa, D. van Dalen, and K. McAloon (eds.), *Logic Colloquium '78: Proceedings of the Logic Colloquium at Mons*, North-Holland, Amsterdam (1979).

7. E. Moggi. The Partial Lambda-Calculus. PhD thesis, University of Edinburgh, 1988. http://citeseer.nj.nec.com/moggi88partial.html

8. Scott, D., Identity and existence in intuitionistic logic, in: Fourman, M. P., Mulvey, C. J., and Scott, D. S. (eds.), *Applications of Sheaves*, Lecture Notes in Mathematics **753** 660-696, Springer–Verlag, Berlin Heidelberg New York (1979).

9. Shoenfield, J. R., *Mathematical Logic*, Addison-Wesley, Reading, Mass. (1967).