Lambda Logic

Michael Beeson [*]

October 30, 2010

Abstract

Lambda logic is the union of first order logic and lambda calculus. The purpose of this note is to give a complete and precise definition of the syntax of lambda logic, and explain its relations to first order logic and to simple type theory. This is a revised and expanded version of [6].

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 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

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

are not the first to define or study similar systems.[1] The applicative theories proposed by Feferman in [7] 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], [3], but these are stronger than lambda logic. As far as we know, the systems defined in this paper have not been studied before.

Both ordinary and lambda logic can be modified to allow "undefined terms". In the context of ordinary logic this has been studied in [9], [5], [4]. In the context of applicative theories, [4] defined and studied "partial combinatory algebras"; but application in the λ-calculus is always total. Moggi [8] was apparently the first to publish a definition of partial lambda calculus; see [8] for a thorough discussion of different versions of partial lambda calculus and partial combinatory logic.

2 Syntax

As is usual in formalizing first-order logic, we suppose given an infinite list of variables. We also suppose given an infinite list of predicate symbols and an infinite list of function symbols. Unlike in first-order logic we require that every predicate symbol is also a function symbol. We do not require that the predicate and function symbols come with a prescribed arity (although semantically, if f occurs with different arities, the occurrences with different arities will be treated as if they were different symbols). We suppose given an infinite list of constant symbols, disjoint from the list of variables. The particular symbol *lambda* is not a function or predicate symbol, and the list of function symbols contains a distinguished element *Ap*.

If someone wishes a more precise specification of this basic apparatus, we could take the following completely precise definition:

```
digit:= 0-9.
variable:= [U-Z]digit* | [u-z]digit*.
constant:= [A-T]digit* | [a-t]digit* | true | false.
logicalSymbol := and | or | implies | neg | =.
functionSymbol:- Ap | [a-t]digit* | [A-T]digit* | logicalSymbol.
predicateSymbol:- Ap | [A-T]digit* | logicalSymbol.
```

Here is a grammar defining the syntax of lambda logic.

```
binder:= lambda | forall | exists.
functionTerm:= functionSymbol(term {,term}*).
term:= variable | constant | functionTerm | binder(variable, term).
atomicFormula:= predicateSymbol(term {, term}*) | true | false.
formula:= atomicFormula | compoundFormula.
```

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

```
compoundFormula:-  connective(formula,formula) |
                   neg(formula) |
                   quantifier(variable,formula).
connective:- and, or, implies.
quantifier:- forall | exists.
```

Note that because predicate symbols are also function symbols (here just called "symbols"), and *true* and *false* are constants, every formula is also a term. Thus, for example, $lambda(x, x+y = y+x)$ is a term, since $x+y = y+x$ is a term, since $=$ is a function symbol. Hence $Ap(lambda(x, x+y = y+x), z)$ is a formula, since Ap is a predicate symbol and $lambda(x, x+y = y+x)$ is a term. There will also be formulas that would not occur in first-order logic, such as $f(forall(x,a)) = b$ (where a and b are constants).

On the other hand, not every term is a formula. For example, a variable is not a formula. Note that for c a constant, $Ap(lambda(x,t),c)$ is a formula, since Ap is a symbol and $\lambda(x,t)$ is a term. As a term, this expression β-reduces to t, but that does not make t a formula. Indeed, when we get to the axioms, we will see that β-reduction applies to formulas, but only if both sides *are* formulas. Another difference between terms and formulas is that lambda terms do not count as formulas. In other words, we do not allow *predicate abstraction*, for example forming a new predicate $\lambda x.P(x,y)$ from the predicate P. This can be done indirectly in case P has a characteristic function, but is not allowed in general.

As is customary in logic textbooks, we regard infix notation as an abbreviation at the meta-level, so that $(x+y)$ really means $+(x,y)$. Similarly $A \land B$ is really $\land(A,B)$. $\mathbf{T}$ is regarded as a syntactic variant of *true*. That means that '$\mathbf{T}$' and *true* are two different names for the same symbol. Similarly, $\mathbf{F}$ is a syntactic variant of $false$. The term $lambda(x,t)$ can also be written as $\lambda x.\,t$; the latter notation is just an informal abbreviation for the offical notation $lambda(x,t)$. The notations $\forall x\, A$ and $(\forall x)\, A$ are meta-level abbreviations for $forall(x,A)$, and similarly for $\exists$. The symbols $\land$,$\lor$, $\to$, and $\neg$ are syntactic variants of `and`, `or`, `implies`, and `neg`, respectively. Other typographically convenient symbols, such as $<$ or $\leq$, can similarly be regarded as meta-level abbreviations for certain legal predicate symbols.

Each formula has two representations: as a string and as a labeled tree. One and only one tree representation corresponds to each string representation, and vice-versa. An *occurrence* of a variable in a formula is represented by a single leaf node in the tree representation. In that case there is a unique path from that node to the root of the tree representation. The *binding operators* are $\forall$, $\exists$, and λ. An occurrence of binding operator in a term or formula has a variable as its first argument (child in the tree). The concepts that a certain occurrence of a variable in a term or formula A is *free in* A or *bound in* A, and the concept that an occurrence of a binding operator *binds* certain occurrences of variables, are defined mutually by recursion as follows. "Bound" is a synonym for "not free".

In a formula $\forall x\, A$, this occurrence of $\forall$ binds all occurrences of x that are free in A. Those occurrences of x are bound in $\forall x\, A$. In a formula $\exists x\, A$, this occurrence of $\exists$ binds all occurrences of x that are free in A. Those occurrences of x are bound in $\exists x\, A$. In a term $lambda(x,t)$, this occurrence of *lambda* binds all occurrences of x that are free in t. Those occurrences of x are bound in $lambda(x,t)$. Occurrences of variables in compound terms and formulas constructed by the syntax rules not involving binding operators retain the free or bound status that they had in the component formulas. Note that the same variable can have both free occurrences and bound occurrences in a given formula.

Renaming bound variables. If z is a variable that does not occur in $\forall x\, A$, then the formula obtained by replacing all occurrences of x that are bound by this binding operator by z is said to be *α-convertible* to $\forall x\, A$; similarly for $\exists x\, A$ and $lambda(x,t)$. The relation of *α-equivalence* is the least reflexive and transitive relation containing α-convertibility and satisfying in addition the conditions that

(i) if t is α-equivalent to s then $f(t)$ is α-equivalent to $f(s)$, and similarly $f(r,t)$ is α-equivalent to $f(r,s)$, and similarly for each argument position of a term of any arity.

(ii) if t is α-equivalent to s then $lambda(x,t)$ is α-equivalent to $lambda(x,s)$.

If x is any variable, and A is any term or formula, then we can find a term or formula B such that B is α-equivalent to A and B does not contain any bound occurrences of x. We say such a B is "free for x". Specifically, an algorithm $freeFor(A,x)$ is defined by recursion on the term-or-formula A. The crucial clause is when the main symbol of A is a binding operator and the variable it binds is x. In that case, the algorithm is called recursively on the second argument of the term-or-formula and then, (i) a "fresh" variable y is chosen (the lexicographically least one beginning with the same letter as x and not occurring in the result of the recursive call), and then (ii) each occurrence of x bound by this occurrence of the binding operator is replaced by y.

Substitution. By $A[x := t]$ we mean the result of the following algorithm: Initialize P to A. Then for each variable z having at least one free occurrence in t, replace P by $freeFor(P,z)$. The final value of P has no bound occurrences of any variables free in t, and is α-equivalent to A. Then replace each free occurrence of x in P by t. That is the value of $A[x := t]$. In other words, we first rename bound variables in A to avoid capture of the free variables of t, and then replace x by t in A.

β reduction. The term $Ap(lambda(x,t),q)$ is said to β-reduce immediately to $t[x := q]$. The relation "t β-reduces to s" is defined as the least reflexive and transitive relation containing immediate β-reduction and satisfying:

(i) if t β-reduces to s then $f(t)$ reduces to $f(s)$, and similarly $f(r,t)$ reduces to $f(r,s)$, and similarly for each argument position of a term of any arity.

(ii) if t β-reduces to s then $lambda(x,t)$ β-reduces to $lambda(x,s)$.

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 derive the following:

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

The hypothesis $G(x) \wedge G(y)$ is needed because we relativized the group axioms to G. We want to formulate the theory of groups, not the theory of models of the lambda calculus that can be turned into groups. Alternately, we can replace $G(x)$ by $Ap(G, x) = \mathbf{T}$, and $x \cdot y$ by $Ap(Ap(\cdot, x), y)$.

If we want to discuss more than one group at a time, then instead of using a constant e for the identity, we can a variable; and we can even use a variable for the group operation (although then we can no longer write $x \cdot y$ in infix notation; we must write $Ap(Ap(o, x), y)$. Unofficially we could abbreviate that as (xoy) for readability in the context of group theory.

That formulation allows us to discuss subgroups, homomorphisms, etc. quite naturally.

Specifically, a homomorphism from a group $(G, e, \times)$ to $(H, 1, +)$, where here $\times$ and $+$ are variables being used for the two group operations, would be an object u such that

$$Ap(G, x) \rightarrow (H(Ap(u, x))$$

$$Ap(G, x) \wedge Ap(G, y) \rightarrow Ap(u, x) + Ap(u, y) = Ap(u, x \cdot y)$$

Now one can quantify over homomorphisms, for example to state that a homomorphism from G to H is one-to-one if and only if its kernel is the identity.

Note that we could also have used two predicate symbols G and H instead of variables. Then we would have written $G(x)$ instead of $Ap(G, x)$. These are not the same. Using predicate symbols, we would not be able to quantify over groups.

Example: Peano's axioms. Peano's axioms for number theory say that the natural numbers N constitute the least set containing 0 and closed under successor, where successor is a one-to-one function from N to N that takes every value except 0. It is common to consider the recursive definitions of addition and multiplication as part of Peano's axioms, too. An equivalent formulation is that mathematical induction holds for all sets X of integers: if $0 \in X$ and $\forall n(n \in X \rightarrow s(n) \in X)$ then $\forall n(n \in X)$. The first-order theory PA is obtained by taking the induction schema, in which $n \in X$ is replaced by any first-order formula $\phi(n)$. In lambda logic we can replace $n \in X$ by $Ap(X, n) = \mathbf{T}$, where X is a variable. Specifically, the induction schema in lambda logic is

$$Ap(X, 0) \wedge \forall x\,(N(x) \wedge Ap(X, x) \rightarrow Ap(X, s(x))) \rightarrow \forall z\,(N(x) \rightarrow Ap(X, z)).$$

Here N is a unary predicate symbol; the other Peano axioms would be relativized to N. In making proofs by induction, we would want to instantiate X to a term such as $\lambda(z, x+z = z+x)$. To construct such a term, we use the grammar rule `term := formula`.

This is somewhat analogous to a "weak second-order theory" in which there are two sorts of first-order variables. The proof-theoretic strength of such theories depends on what comprehension axioms are included to allow the definition of sets to use in inductive proofs. The same presumably is true in lambda logic; without axioms even to define the characteristic functions of arithmetic predicates, this form of Peano's axioms will be weak, probably only as strong as Π^0_0 induction in the language of PA.

3 Axioms and inference rules

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 [10], p. 20), for the standard version. Since every formula is a term, the concepts "normal term" (a term admitting no reductions) and "strongly normalizable term" (every reduction sequence terminates) apply to formulas as well. We note that there do exist formulas that are not normalizable, such as $Ap(\ell, \ell)$, where ℓ is $\lambda x. \neg\, Ap(x, x)$. Indeed this term reduces to $\neg\, Ap(\ell, \ell)$, so the term $Ap(\ell, \ell)$ is a formula A such that A reduces to $\neg A$. The axioms of lambda logic permit us to derive $A = \neg A$, but they do not permit us to derive $A \leftrightarrow \neg A$, thanks to restricting β-reduction on formulas to apply only to strongly normalizable formulas. Here is the list of axioms and inference rules of lambda logic:

(*Prop*) propositional axioms (see [10], p. 20)

(Q) standard quantifier axioms and rules (see [10], p. 20)

(α) $t = s$ if t and s are alpha-equivalent terms.

(α) $A \rightarrow B$ if A and B are alpha-equivalent formulas.

(β) $Ap(\lambda x.\, t, s) = t[x := s]$ where t is any term.

(β) $Ap(\lambda x.\, A, s) \leftrightarrow A[x := s]$ where $A[x := s]$ is a strongly normalizable formula.

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

(*true and false*) $\mathbf{T} = \lambda x \lambda y.\, x$ and $\mathbf{F} = \lambda x \lambda y.\, y$

(*non-triviality*) $\mathbf{T} \neq \mathbf{F}$

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". If ϕ is a formula, we recursively define $M \models_g \phi$ as follows:

$$\begin{aligned}
M &\models_g \forall x\, A \ \text{ iff } \ M \models_h A \text{ whenever } h \text{ extends } g \text{ and is defined on } x \\
M &\models_g \exists x\, A \ \text{ iff } \ M \models_h A \text{ for some } h \text{ that extends } g \text{ and is defined on } x \\
M &\models_g \neg A \ \text{ iff } \ \text{it is not the case that } M \models_g A \\
M &\models_g A \wedge B \ \text{ iff } \ M \models_g A \ \text{ and } \ M \models_g B \\
M &\models_g A \vee B \ \text{ iff } \ M \models_g A \ \text{ or } M \models_g B \\
M &\models_g t = s \ \text{ iff } \ g(t) \text{ and } g(s) \text{ are the same element of } M \\
M &\models_g P(t_1,\ldots,t_n) \ \text{ iff } \ P_M(g(t_1,\ldots,g(t_n)) \ \text{ where } P \text{ is not } Ap \\
M &\models_g Ap(lambda(x,B),t) \ \text{ iff } \ B \text{ is a formula and } M \models_g B[x := t]
\end{aligned}$$

We regard these clauses as an inductive definition, in the following way. The definition defines the satisfaction relation $\models_g$ as a union of "levels", and simultaneously defines a subset of the complement of the relation $\not\models_g$ by levels. For example, the clause for negation specifies that if $M \models_g A$ at level n, then $M \not\models_g \neg A$ at level $n+1$, and vice-versa, if $M \not\models_g A$ at level n then $M \models_g \neg A$ at level $n+1$. The relation $\models_g$ obtained as the union of all the levels (indexed by positive integers) is the "minimal satisfaction relation."

If B is a strongly normalizing formula, then $M \models B$ or $M \not\models B$ at the n-th stage, where n is the complexity of the formula B. We note, however, that the minimal satisfaction relation is not defined on all formulas. For example, if ℓ is $\lambda x\, \neg Ap(x,x)$, then $Ap(\ell,\ell)$ is a formula, although it is not a strongly normalizing formula. Thus $Ap(\ell,\ell) \leftrightarrow \neg Ap(\ell,\ell)$ is not an axiom, since the right side has no normal form. But nevertheless $Ap(\ell,\ell)$ is a formula, and we have neither $M \models Ap(\ell,\ell)$ nor $M \not\models Ap(\ell,\ell)$. Hence, using the minimal satisfaction relation, the law of the excluded middle is not satisfied. Similarly, $\neg\neg Ap(\ell,\ell) \to Ap(\ell,\ell)$ is not satisfied.

We define a "satisfaction relation" on M to be a relation $\models_g$ satisfying the above clauses, and a "classical satisfaction relation" to be a satisfaction relation that also satisfies the law of the excluded middle. (Technically, a satisfaction relation takes two arguments: a formula and a valuation; our notation sometimes uses $\models_g$ for the relation of two variables, and sometimes for the relation of one variable with g fixed.) The existence of such relations can be proved by continuing the "levels" past the ordinal ω. If, for limit ordinal μ, there is a formula A such that neither $M \models_g A$ nor $M \not\models_g A$ at a level below μ, then let A be the least such formula and specify that $M \models_g A$ at level μ. Then the next ω stages are defined as above. Since there are only as many formulas as M has elements, this process eventually closes off.

We note that it is more or less arbitrary whether, at the "choice points" in the construction, $M \models_g A$ or not; there will be different classical satisfaction relations making different choices about the satisfiability of, for example, $Ap(\ell,\ell)$; but all classical satisfaction relations will agree on strongly normalizing formulas, and they agree with the minimal satisfaction relation on such formulas.

We define $M \models A$ if for all valuations g, $M \models_g A$. Note that for formulae not involving λ, the definition agrees with the usual definition of satisfaction in first order logic. Note also that for equality statements $t = s$, there is no ambiguity about the satisfaction relation. For example, $Ap(\ell,\ell) = \neg Ap(\ell,\ell)$ is satisfied, since the left side β-reduces to the right side; but this fact coexists with the fact that one and only one of the formulas $Ap(\ell,\ell)$ and $\neg Ap(\ell,\ell)$ can be satisfied, and "it can go either way." Such ambiguity does not arise for strongly normalizable formulas.

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 1 (λ-model) $(M, \lambda^*, \models_g)$ *is a λ-model of a theory T in lambda logic if $\models_g$ is a satisfaction relation on (M, λ^*), and under that satisfaction* *relation, (M, λ^*) satisfies the axioms α, β, and ξ, and M satisfies the axioms of T.*

5 Consistency of lambda logic

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 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:

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

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.

Theorem 2 *Lambda logic is consistent, i.e. does not derive* $\mathbf{T} = \mathbf{F}$.

Proof. We define a model $(M, Ap_M, \models_g)$ for lambda logic. The elements of M are the equivalence classes $[t]$ of terms t under the relation $t \equiv s$ defined above. We define $Ap_M([t],[s]) := [Ap(t,s)]$. Then let $\models_g$ be a satisfaction relation, i.e. a fixed point of the inductive clauses given above.

One proves that, for valuations g with values in M, and x in the domain of g, and r any term in the equivalence class $g(x)$, we have

$$M \models_g A \text{ if and only if } M \models A[x := r]. \qquad (1)$$

The β axiom for terms is satisfied since $[Ap(lambda(x,t), r)]$ is $[t[x := r]]$. The β axiom for formulas is automatically satisfied by the definition of $M \models A$. The axiom $\mathbf{T} \neq \mathbf{F}$ is satisfied since $\mathbf{T}$ and $\mathbf{F}$ are distinct normal terms.

The weak extensionality axiom (ξ) is satisfied, as follows: Suppose M satisfies $\forall x\, (Ap(t, x) = Ap(s, x))$. Let g be the valuation assigning x the value $[x]$. Then by (1), M satisfies $Ap(t, x) = Ap(s, x)$. Hence M also satisfies $lambda(x, Ap(t, x)) = lambda(x, Ap(s, x))$, the conclusion of ($\xi$). That completes the proof.

6 Some consequences of the fixed point theorem

The fixed point theorem of lambda calculus says that for every F, there is an Ω such that $F\Omega = \Omega$. Namely, we can take $\Omega = Ap(\omega, \omega)$, where $\omega = lambda(x, Ap(F, Ap(x, x)))$. Another form of the fixed point theorem says that for every term H, we can find a term f such that $Ap(f, x) = H(f, x)$. (To prove this, apply the first theorem with $F = \lambda x.\, H(f, x)$, and take f to be the Ω produced by the first theorem.)

Russell's paradox as a fixed point. Since every F has a fixed point, there cannot be (in lambda calculus or in lambda logic) any term *not* such that for all x, $not(x) \neq x$. This is the essence of Russell's paradox: Suppose there was such a *not*. If, following Church, we identify $x \in y$ with $Ap(y, x) = \mathbf{T}$, then following the proof of the fixed-point theorem we would set $\omega = \lambda x.\, Ap(not, Ap(x, x))$, which we recognize as the Russell set $R = \{x : x \notin x\}$, since $\omega(x) = \mathbf{T}$ if and only if $Ap(x, x) \neq \mathbf{T}$. Then considering the value of $\Omega = Ap(\omega, \omega)$ amounts to Russell's famous question whether $R \in R$ or not, and the fixed-point argument shows $not(\Omega) = \Omega$, which in Russell's terms says $R \in R$ if and only if $R \notin R$. Of course in lambda logic, this is not a paradox, since no such term *not* exists. Although one may find it counterintuitive that no such term exists, this is not new with lambda logic, but is a feature of lambda calculus known for over seventy years now.

Inconsistency of AC. Consider the following version of the axiom of choice:

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

AC is inconsistent with lambda logic: Simply take $P(x, y)$ to be $x \neq y$. Then by the non-triviality axiom of lambda logic, we can prove $\forall x \exists y\, x \neq y$ (take y to be T if $x \neq \mathbf{T}$, and $\mathbf{F}$ if $x = \mathbf{T}$). But then any choice function f would satisfy the equation for *not*, and hence there is no such f.

In [6], the last theorem is only vacuously true, since its conclusion says that something is provable in lambda logic plus AC.

Failure of Skolemization. The same example shows that Skolemization does not work in lambda logic. That is, in first order logic, $\forall x\, \exists y\, A(x, y)$ is satisfiable if and only if $\forall x\, A(x, f(x))$ is satisfiable. But if we take $x \neq y$ for $A(x, y)$, the former is satisfiable in lambda logic, but the latter is not, since then $\lambda x\, f(x)$ must have a fixed point. We show in the next section that this is not a serious problem–we just have to not allow lambda-abstraction when it would capture free variables in Skolem terms. That extended version of lambda logic can be used to prove a completeness theorem.

An example in group theory. Here is a more mathematical example using the fixed point theorem. Consider the theory in which we have a binary function $\cdot$ (written in infix notation), a constant e, and the axioms saying that $\cdot$ is a group operation identity e. Take $H(f, x) = c \cdot Ap(f, x)$, where c is anything. Applying the second form of the fixed point theorem we get $Ap(f, x) = c \cdot Ap(f, x)$. It follows from the axioms of group theory that c is the group identity. Since c was anything, we could have taken c to be $\mathbf{T}$ or $\mathbf{F}$, but this implies $\mathbf{T} = \mathbf{F}$, contradicting the non-triviality axiom of lambda logic. This theory is inconsistent.

Looked at semantically, this is not a surprising result: it only means that it is impossible, given a lambda model M, to define a binary operation on M and an identity element of M that make M into a group. If we use a unary predicate G, and put in axioms saying that $\cdot$ is a group operation on G with identity e, then the above argument only shows that $Ap(f, x)$ is not in G, i.e. $\neg G(Ap(f, x))$.

Extensionality. The following additional formula is sometimes added to lambda calculus.

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

It is not an axiom of lambda logic. It is, however, consistent with lambda logic since in the consistency proof in the previous section, we could have used η-reduction as well as β-reduction, appealing to the Church-Rosser theorem for η-reduction in lambda calculus.

Terms and formulas. Suppose we have deduced $\neg A$ for some formula A. The rules of lambda logic do not allow us to deduce $A = \mathbf{F}$ as a term. Similarly, just because we have proved A does not entitle us to deduce $A = \mathbf{T}$. Indeed, making A equivalent to $A = T$ and $\neg A$ equivalent to $A = \mathbf{F}$ would lead to contradictions. Freek Wiedijk gave the following example: let A be $Ap(\lambda w\, \lambda x\, \lambda y.\, \mathbf{T}, w)$. Then A is a formula, so we have $A \vee \neg A$. If we were able to deduce from this, $A = \mathbf{T} \vee A = \mathbf{F}$, we could obtain a contradiction, as follows: As a term, A reduces to $\lambda x\, \lambda y.\, \mathbf{T}$. Hence $\lambda x\, \lambda y.\, \mathbf{T} = \mathbf{T} \vee \lambda x\, \lambda y.\, \mathbf{T} = \mathbf{F}$. But each disjunct can be refuted, since $Ap(Ap(\mathbf{T}, \mathbf{F}), \mathbf{F}) = \mathbf{F}$ and $Ap(Ap(\mathbf{F}, \mathbf{F}), \mathbf{F}) = \mathbf{F}$ according to the lambda-calculus definitions of $\mathbf{T}$ and $\mathbf{F}$, but $Ap(Ap(\lambda x\, \lambda y.\, \mathbf{T}, \mathbf{F}), \mathbf{F}) = \mathbf{T}$, and one of the axioms of lambda logic is $\mathbf{T} \neq \mathbf{F}$.

Note that $\lambda x\, \lambda y.\, \mathbf{T}$ is not a formula, so we are not able to β-reduce A as a formula, and the contradiction cannot be carried out in lambda logic. A formula only β-reduces if it reduces to another formula (and indeed, that formula must be strongly normalizable).

7 Completeness of lambda logic

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. (A contradiction is a formula of the form $A \wedge \neg A$.) In this section, we will prove the lambda completeness theorem: any consistent theory has a λ-model. First, we need some preliminaries.

Theorem 3 (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. Our plan is to imitate the usual proof (Henkin's method) of the completeness theorem, as set out for example in [10] pp. 43-48, can be imitated for lambda logic. However, the first step in the classical proof is Skolemization, which we have seen is problematic in lambda logic. The reason why it is problematic is that lambda logic permits lambda abstraction to be used on any function symbol, so for example we cannot construct a model with a Skolem function for $\forall x \exists y\, x \neq y$. The remedy is to enlarge lambda logic to allow "Skolem function symbols" with a restriction on their use in lambda terms. This enlarged logic presumes a list of "Skolem symbols" disjoint from the list of function symbols of lambda logic. A "Skolem term" is a term whose main symbol is a Skolem symbol. The restriction is that the term $lambda(x, t)$ can only be formed when t does not contain a Skolem term with x free. Otherwise Skolem symbols are used just like ordinary function symbols and constants in forming terms.

Example. If $f(x)$ is a Skolem term with the associated axiom $f(x) \neq x$, then we cannot form $\lambda(x, f(x))$ (so we cannot derive a contradiction from the existence of a fixed point of that term).

The axioms of Skolem lambda logic are the same as those of lambda logic, except (of course) that in the (β) and (ξ) axiom schemata, only those instances are taken that are legal formulas in Skolem lambda logic.

We refer to this more elaborate system as "Skolem lambda logic". We will use it only in the proof of the completeness theorem. The semantics of Skolem lambda logic generalizes the semantics of lambda logic defined above, by interpreting the Skolem terms as they would be interpreted in first order logic. That is, the model must specify a function $\bar{f}$ from M to M to serve as the interpretation of each Skolem function symbol f. (If the same symbol occurs with different arities, it is interpreted by a different function for each arity.) The function λ^*, which takes a variable x and an M-term t as arguments, does not need to be defined on terms t containing Skolem terms with x free.

We show that a single quantifier can be eliminated. Consider a theorem T containing an axiom $\forall x \exists y\, A(x, y)$, and consider the theory T^* containing a new a Skolem function symbol f and the axiom $\forall x\, A(x, f(x))$ instead of the axiom $\forall x \exists y\, A(x, y)$. Note that this axiom is a formula of Skolem lambda logic since $A(x, f(x))$ means $A[y := f(x)]$ and any bound variables that occur free in $f(x)$ are renamed as part of the substitution process, so no free variables of the Skolem term $f(x)$ are captured by a lambda-binding.

We claim T has a model if and only if T^* has a model. A model of T^* already satisfies T, so only one direction requires proof. Suppose M satisfies T. Then for every $x \in M$ there is a $y \in M$ such that M satisfies $A(x, y)$. Using the axiom of choice (at the meta-level) there is a function $\bar{f} : M \to M$ such that M satisfies $A(x, \bar{f}(x))$ for every x. We take $\bar{f}$ as the interpretation of f. Then the axiom $\forall x\, A(x, f(x))$ is satisfied. We have to extend the function λ^* of M (which takes a variable and an M-term and produces an M-term) to be defined on M-terms in the language of T^*. We also have to define the interpretations $\bar{r}$ of terms in Skolem lambda logic in M. These two tasks are accomplished simultaneously by mutual recursion; the function λ^* is used just as for lambda calculus to define the interpretation of lambda-terms, and now we show how to extend λ^* to those lambda terms that are legal in Skolem lambda logic. Let t be an M-term of T^*, and x a variable, and suppose that t contains some Skolem M-subterms, but does not contain any Skolem subterms with x free. Then t can be written as $t'[u := r]$, where u stands for several variables and r for the list of all Skolem subterms of t, and t' does not contain any Skolem terms. Since the terms r do not contain x free, the variable x does not get renamed when we substitute r for u in $\lambda(x, t')$. Thus

$$\lambda(x, t')[u := r] = \lambda(x, t'[u := r]) = \lambda(x, t)$$

We define

$$\lambda^*(x, t) := \lambda^*(x, t'[u := c_{\bar{r}}])$$

where $\bar{r}$ is the interpretation in M of the Skolem term r and $c_{\bar{r}}$ is the constant denoting that element of M.

Now we check the validity of axiom (β). Let r be a list of all Skolem terms occurring in t and u a corresponding list of variables not occurring in t. Let t' be a term such that $t = t'[u := r]$, and t' contains no Skolem terms. In M we have

$$\begin{aligned}
&Ap(lambda(x, t), q) \\
&\quad = Ap(\lambda^*(x, t), q) \\
&\quad = Ap(\lambda^*(x, t'[u := c_{\bar{r}}], \bar{q}) \qquad \text{by definition of } \lambda^*(x, t) \\
&\quad = Ap(lambda(x, t')[u := c_{\bar{r}}]), \bar{q}) \qquad \text{since } x \text{ does not occur free in } \bar{r} \\
&\quad = Ap(lambda(x, t'), \bar{q})[u := c_{\bar{r}}] \qquad \text{since } c_{\bar{r}} \text{ is a constant term} \\
&\quad = t'[x : \bar{q}][u := c_{\bar{r}}] \qquad \text{since axiom } (\beta) \text{ holds in } M \\
&\quad = t[x := \bar{q}]
\end{aligned}$$

Finally we check the validity of the weak extensionality axiom (ξ). Recall that the axiom in question says that if for all x, $Ap(t, x) = Ap(s, x)$, then $\lambda(x, Ap(t, x)) = \lambda(x, Ap(s, x))$. This is an axiom of Skolem lambda logic only in case t does not contain any Skolem subterms with x free. In that case, let t' and s' be terms without Skolem subterms, and r a list of Skolem terms, such that $\lambda^*(x, t) = \lambda^*(x, t'[u := c_{\bar{r}}])$ and $\lambda^*(x, s) = \lambda^*(x, s'[u := c_{\bar{r}}])$ Then in M, by hypothesis we have $Ap(t, x) = Ap(s, x)$ for all x, so also $Ap(t'[u := r], x) =$

$Ap(s'[u := r], x)$. Since the interpretation in M of $t'[u := r]$ is the same as that of the M-term $t'[u := c_{\bar{r}}]$, we also have

$$Ap(t'[u := c_{\bar{r}}], x) = Ap(s'[u := c_{\bar{r}}], x).$$

But these terms contain no Skolem subterms. Hence, applying axiom (ξ) in M, we have

$$lambda(x, t'[u := c_{\bar{r}}]) = lambda(x, s'[u := c_{\bar{r}}]).$$

Therefore M satisfies

$$\lambda(x, t'[u := r]) = \lambda(x, s'[u := r]),$$

which is the same as to say M satisfies $\lambda(x, t) = \lambda(x, s)$. That completes the verification of axiom (ξ).

Now, as usual in first-order logic, we can eliminate quantifiers one by one in favor of Skolem functions, so that to every theory T in lambda logic, we can construct a theory T^* in Skolem lambda logic with quantifier-free axioms, such that T has a model if and only if T^* has a model. Hence it suffices to prove the completeness theorem for quantifier-free Skolem 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:

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

In this definition, the right sides depend only on the equivalence classes $[t_i]$ (as shown in [10], p. 44).

Exactly as in [10] 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, and t does not contain Skolem subterms in which x occurs free. 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 first (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).

Consider the second (β) axiom:

$$Ap(\lambda x.\, A, s) \leftrightarrow A[x := s] \text{ where } A[x := s] \text{ is a strongly normalizable formula.}$$

One of the inductive clauses defining $\models_g$ says that if the right-hand side is satisfied, then the left-hand side is satisfied. Since $\models_g$ is a fixed point of these inductive clauses, and since no other clause in the definition provides for the satisfaction of a formula of the form $Ap(lambda(x, B))$, then if the left-hand side is satisfied so is the right-hand side. Hence the axiom is satisfied.

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.

8 The first order fragment of lambda logic

Theorem 4 (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 rivial. 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 $m < n$, $\lceil m \rceil \neq \lceil n \rceil$ 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 $\lceil n \rceil \neq \lceil n + 1 \rceil$. 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. We will show that 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 $\hat{M}$. To complete the construction of a λ-model, we supply any classical satisfaction relation on $\hat{M}$.

To do this we define the relation Ap_M and a binary relation E on M by simultaneous induction. Ap_M will serve as the interpretation of Ap and E will serve as the interpretation of equality.[2] Since M is infinite we can define an element 0 and a pairing function $\langle a, b\rangle$ on M in such a way that the interpretations of the losed terms of T are never pairs, and 0 is not a pair. Define $< a, b, c > = < a, < b, c >>$, etc. Define the successor of x to be $s(x) = < 0, x >$, and define the "numeral" $\bar{m}$ inductively by $\bar{0} = 0$ and $m \bar{+} 1 = s(\bar{m})$. Henceforth we drop the bars, writing for example $\langle 1, k\rangle$ instead of $\langle \bar{1}, \bar{k}\rangle$. An element of the form $< 1, j, k >$ will be used as an "index" of the k-th function symbol of arity j in T, which we denote by f^j_k.

Tuples are defined from pairs by $\langle x_1, \ldots, x_{n+1}\rangle = \langle\langle x_1, \ldots, x_n\rangle, x_{n+1}\rangle$. The *members* of a tuple $\langle x_1, \ldots, x_n$ are the x_i for $i = 1, \ldots, n$. When y and w are two tuples of length at least m we write $E_m(y, w)$ for the conjunction of the formulas $E(y_i, w_i)$ for $i \leq m$.

To produce a λ-model we must define an operation λ^*, which takes a variable and an M-term. An M-term (as explained in [1], p. 86 *ff.*), is a term with "parameters from M"; more precisely, a closed term in a language containing a constant c_a for each element a of M. A convenient notation for M-terms is $t[y]$, where y is a tuple of elements of M whose length is the number of free variables of t. This means the following: if $x_1, \ldots, x_n$ are the free variables of t, in order of their occurrence, then $t[y]$ is $t[x_i := c_{y_i}]$. We also need the following notation: $t[y, x]$ where x is a variable, and y is a tuple of elements of M whose length is the number of free variables of t different from x, means the following: if $x_1, \ldots, x_n$ are the free variables of t different from x, in order of their occurrence, then $t[y]$ is $t[x_i := c_{y_i}]$. Note that x may or may not occur free in t.

The operation λ^* is given by

$$\lambda^*(x, t[y, x]) := < 2, j, \ulcorner t \urcorner, y >$$

where y is a tuple whose length is the number of free variables of $\lambda x.\, t$, and x is the j-th variable, and $\ulcorner t \urcorner$ is the (numeral for the) Gödel number of the closed term t. The definitions of Ap, E, and the interpretation $t[y]_M$ of each M-term $t[y]$ are given in one simultaneous inductive definition. The inductive conditions are as follows. In (iii) and (iv), m is the number of free variables of t.

(i) $E(x, y)$ if $x = y$.

(ii) $E(\lambda^*(x, t[y, x]), \lambda^*(z, t[x := z][y]))$.

(iii) $Ap_M(\lambda^*(x, t[y, x]), r_M) = t[x := r][y]_M$.

(iv) $E(\lambda^*(x, t[y, x]), \lambda^*(x, s[w, x])$ if $E(t[x := a][y]_M, s[x := a][w_M)$, and $E_m(y, w)$ where a is a the first constant not occurring in t or s.

[2]If one insists on interpreting equality as identity instead of by an equivalence relation, one may use the equivalence classes of E as the elements of the model.

(v) $E(Ap_M(a,b), Ap_M(c,d))$ if $E(a,c)$ and $E(b,d)$.
(vi) $Ap(a,b)_M = Ap_M(a_M, b_M)$.
(vii) $E(f(t[y])_M, f(s[w])_M)$ if $E(t[y]_M, s[w]_M)$, and similarly for several variables x.
(viii) $(\lambda x.\, t[y,x])_M = \lambda^*(x, t[y,x])$.
(ix) $f(x)_M = f_M(x_M)$ and similarly for several variables x.
(x) $(c_a)_M = a$ where c_a is the constant for a.
(xi) $E(a,c)$ if $E(a,b)$ and $E(b,c)$.

Since E and Ap occur only positively in these clauses, this is a legitimate inductive definition. We interpret equality in M as the relation E. For each predicate P of arity n in the language of T, we define a relation $\hat{P}$ on M by

$$\hat{P}(x_1, \ldots, x_n) := M \models P(y_1, \ldots, y_n) \wedge E_n(x,y)$$

and we define $\hat{M} \models P(x_1, \ldots, x_n)$ if and only if $\hat{P}(x_1, \ldots, x_n)$. Then let $\models_g$ be a classical satisfaction relation, introduced as a fixed point of the inductive clauses for satisfaction. That makes the second (β) axiom automatically satisfied.

We start with the following lemma: if $E(r,q)$ then $E(t[x := r], t[x := q])$ for terms t, q, and r. This is proved by induction on the complexity of the M-term t. When t begins with λ or Ap, the corresponding induction step follows from (iv) and (v). When t begins with a function symbol f, we use (vii). When t is atomic, either it is a constant c_a for an element $a = y_1$ of M, in which case there is nothing to prove, or else it is a variable, in which case we have to prove that $E(r,q)$ from the assumption $E(r,q)$, which is immediate.

We next verify the substitutivity of equality, namely: if $E(r,s)$ and $\hat{M} \models A[x := r]$ then $\hat{M} \models A[x := s]$, where A is a formula of lambda logic with constants for elements of M. We prove this by induction on the complexity of A. Since substitution for free variables commutes with the logical connectives and quantifiers, only the case of atomic A needs a proof. If A is an equality $t = q$, then $A[x := r]$ is $t[x := r] = q[x := r]$ and $A[x := s]$ is $t[x := s] = q[x := s]$. By the lemma we have $t[x := r] = t[x := s]$ in M, and $q[x := r] = q[x := s]$. Assume $\hat{M} \models A[x := r]$. Then by (xi) we have $\hat{M} \models t[x := s] = q[x := s]$. The remaining case is when A is an atomic formula $P(x)$ (x can be several variables.) This is taken care of by the definition of $\hat{P}$ above.

Because of (ii) and (iii), axioms (α) and (β) are satisfied. Now to verify axioms (ξ). Let $t[y]$ be an M-term, i.e. a term with constants for elements y of M substituted for its free variables. Suppose $\hat{M} \models \forall x(t[y]x = s[y]x)$. Then let c be the first constant not occurring in t or s; we have $\hat{M} \models t[x := c] = s[x := c]$. Then by (iv), we have $\hat{M} \models \lambda x.\, t[y] = \lambda x.\, s[y]$. Hence (ξ) holds in M.

We have now proved that $\hat{M}$ is a model of lambda logic. We still must prove it satisfies the theory T in first order logic. This follows from the following lemma: For each M-formula $A(x_1, \ldots, x_n)$ with n parameters from M, we have $\hat{M} \models A(x)$ if and only if there exists y with $E_n(x,y)$ and $M \models A(y)$. To prove the lemma: The case when A is an atomic formula $P(x)$ is true by definition of $\hat{P}$. The case of an atomic formula $t = q$ follows from what has been proved above. The proof is completed by induction on the complexity of A; the quantifiers do not offer any difficulty since the carrier sets of M and $\hat{M}$ are the same. That completes the proof of the theorem.

9 Lambda Unification

Above we defined $t[x := s]$ as the result of substituting term s for free variable x in t, after renaming bound variables in t to avoid capture of free variables of s. At that time we did not extend this definition to simultaneous (or "parallel") substitution for several variables $x_1, \ldots, x_n$, but this can be done as usual. The free variables of t are renamed to avoid clashes with free variables in $s_1, \ldots, s_n$, and then each variable x_i is replaced by s_i at the same time (rather than sequentially).

A *substitution* is a function σ from a set of variables to the set of terms. It is traditional to write $x\sigma$ instead of $\sigma(x)$. A substitution σ has a natural extension to a function from terms to terms, which we also denote by σ, given by $t\sigma = t[x := x\sigma]$, where x stands for the list of all free variables in t, and $t[x := x\sigma]$ denotes a simultaneous substitution.

In first order logic, a substitution σ is said to be a *unifier* of terms t and s if $t\sigma = s\sigma$. Two terms in first order logic (with equality) are provably equal if and only if they are identical, so it doesn't matter if we interpret this to mean that $t\sigma = s\sigma$ is provable in first order logic with equality or just that the terms $t\sigma$ and $s\sigma$ are identical. When we go to generalize this to lambda logic, it does matter, since terms can be syntactically different but provably equal (for example if one β-reduces to the other).

We define σ to be a *lambda unifier* of terms t and s if $t\sigma = s\sigma$ is provable in lambda logic. Similarly, σ is defined to be a lambda unifier of formulas A and B if $A \leftrightarrow B$ is provable in lambda logic.

Our aim here is to give another axiomatization of lambda logic, based on resolution, factoring, and paramodulation. The usual formulation of these rules involves unifiers. We extend these rules to lambda unification as follows:

Paramodulation

$$\frac{\alpha = \beta \qquad P[x := \gamma] \qquad \alpha\sigma = \gamma\sigma}{P[x := \beta\sigma].} \text{ paramodulation}$$

provided that the free variables of $\beta\sigma$ either occur in γ or are not bound in P. This differs from the first order version of paramodulation only in the extra condition about the free variables of $\beta\sigma$. Note that inferences by β-reduction are included in the paramodulation rule, taking $\alpha = \beta$ to be the axiom of β-reduction and γ to be α.

Binary Resolution

$$\frac{A\sigma \leftrightarrow B\sigma \qquad A \mid U \qquad -B \mid V}{U\sigma \mid V\sigma} \text{ binary resolution}$$

Here U and V are sets of literals and $U \mid V$ is their union; $-B$ is a negative literal, i.e. a negated atom; $A \mid U$ means the union of $\{A\}$ with U; and $U\sigma$ means $\{P\sigma : P \in U\}$.

Factoring

$$\frac{A\sigma \leftrightarrow B\sigma \qquad A \mid B \mid U}{A\sigma \mid U\sigma} \text{ factoring}$$

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 5 (Soundness of lambda unification) *(i) Suppose there is a proof of clause C from a set of clauses Γ using binary resolution, factoring, and paramodulation, and the clause $x = x$. Then there is a proof of C from Γ in lambda logic.*

Remark. In [6], "demodulation" is mentioned, but it is not necessary to consider demodulation as a separate rule of inference—from the purely logical viewpoint, demodulation is a special case of paramodulation.
Proof. 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. 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.

10 The Logic of Partial Terms

In the group theory example, 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 [5] or [4], 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:

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

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.

$$R(t_1, \ldots, t_n) \rightarrow t_1\downarrow \wedge \ldots \wedge t_n\downarrow$$
$$f(t_1, \ldots, t_n)\downarrow \rightarrow t_1\downarrow \wedge \ldots \wedge t_n\downarrow$$

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

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.

For example, one of the strictness axioms is

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

We write $t \cong r$ to abbreviate $t \downarrow \vee r \downarrow \to t = r$. It follows from the strictness axiom just stated 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):

$$\begin{aligned}
&x = x \\
&x = y \to y = x \\
&t \cong r \wedge \phi[x := t] \to \phi[x := r]
\end{aligned}$$

11 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.[3]

Partial lambda calculus is a system similar to lambda calculus, but in which *Ap* is not necessarily total. There can then be "undefined terms." 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 rules for deducing $t \downarrow$ are specified as follows: First, we can always infer (without premise) $x \downarrow$ when x is a variable. Second, we can apply the inference rules

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

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

Note that we do not have strictness of *Ap*. As an example, we have $Ap(\lambda y.\, a, b) \cong a$, whether or not $b \downarrow$. We could have formulated a rule "strict (β)" that would

[3]There is, of course, an alternative to λ-calculus known as *combinatory logic*. Application in combinatory logic is also total, but in [4], the notion of a *partial combinatory algebra* is introduced and studied, following Feferman, who in [7] first introduced partial applicative theories. See [8] for some relationships between partial combinatory logic and partial lambda calculus

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.

12 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 (except for Ap), 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 (actually an axiom, not a rule) is included in partial lambda logic.

Here for reference are the axioms of partial lambda logic:

(λ *terms always defined*) $\lambda x.\, t\downarrow$ for each term t.

(*Prop*) propositional axioms (see [10], p. 20)

(Q) quantifier axioms for LPT as given above

(*Strictness*)

$$\begin{array}{ll} R(t_1, \ldots, t_n) \to t_1\downarrow \wedge \ldots \wedge t_n\downarrow & \\ f(t_1, \ldots, t_n)\downarrow \to t_1\downarrow \wedge \ldots \wedge t_n\downarrow & \text{if } f \text{ is not } Ap \\ t_1\downarrow \wedge \ldots \wedge t_n\downarrow \wedge f(x_1, \ldots, x_n)\downarrow \to f(t_1, \ldots, t_n)\downarrow & \end{array}$$

(α) $t \cong s$ if t and s are alpha-equivalent.

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

(β) $Ap(\lambda x.\, B, s) \leftrightarrow B[x := s]$ when B is a formula.

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

(*true and false*) $\mathbf{T} = \lambda x \lambda y.\, x$ and $\mathbf{F} = \lambda x \lambda y.\, y$

(*non-triviality*) $\mathbf{T} \neq \mathbf{F}$

We review the semantics of LPT as given in [5], [4]. 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 2 (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.[4]

Theorem 6 (Lambda Completeness Theorem for LPT) *Let T be a consistent theory in partial lambda logic. Then T has a partial lambda model.*

Proof. As for (total) lambda logic, we have to extend partial lambda logic to allow Skolem functions, not allowing the construction of lambda terms that capture free variables in Skolem terms. Exactly as for lambda logic, every theory T in partial lambda logic has a Skolemized version T^* in Skolem partial lambda logic, such that T has a model if and only if T^* has a model. It therefore suffices to prove the completeness of quantifier-free Skolem partial lambda logic.

To do that, we again imitate the Henkin proof of completeness for first order 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 $T \vdash t = r$. But now, we take only those closed terms t for which T proves $t \downarrow$; that ensures the validity of the axiom $x = x$, and of the axioms $x \downarrow$ for variables x and of the axioms $c \downarrow$ for constants c. 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 [10], p. 44). The atomic formula $t \downarrow$ is satisfied in M if and only if T proves $t \downarrow$. It follows by induction on the complexity of the closed first-order term t that if T proves $t \downarrow$ then the interpretation t^M of t in M is the equivalence class $[t]$ of t. The strictness axiom for first order function symbols is used in the

[4]In [5] there is a completeness theorem for LPT, generalizing Gödel's completeness theorem. The strictness axiom is important in the proof.

induction step, so this does not apply to terms containing Ap unless we have strictness for Ap.

Exactly as in [10] one then verifies that M is a first order model of T in the sense of LPT. Thus the completeness theorem for first order LPT (with strictness) is proved (again–it was first proved in [5]).

To turn M into a λ-model, we must define $\lambda^*(x,t)$, where t is an M-term, i.e. a closed term with parameters from M. The "parameters from M" are constants $c_[q]$, where $[q]$ is the equivalence class of a closed term q of T. If $[t]$ is the equivalence class of an M-term t, let $[t]^\circ$ be a closed term of T obtained from t by replacing each constant $c_[q]$ by some closed term q in the equivalence class $[q]$. Which closed term we select does not affect the equivalence class $[t]^\circ$. In other words, $[t]^\circ$ is a well-defined operation on equivalence classes.

We define $\lambda^*(x,t) = [\lambda x.\,[t]^\circ]$. Because of the axiom $\lambda x.\,t\downarrow$, $\lambda^*(x,t)$ is indeed defined for M-terms t. We must show that the value depends only on the equivalence class $[t]$. Suppose T proves $t = s$. Then by axiom (xi), T proves $\lambda x.\,[t] = \lambda x.\,[s]$.

We verify that the axioms of partial lambda logic hold in M. First, the beta axiom: $Ap(\lambda x.\,t, r) \cong t[x := r]$. It suffices to consider the case when t has only x free. Suppose that the left hand side is defined in M. Its interpretation is the equivalence class of $Ap(\lambda x.\,t, r)$. By axiom (β), the right side is provably equal to the left side, so it is defined in M, and its interpretation is the class of $t[x := r]$. On the other hand if the right hand side is defined in M, then it is provably equal to the left side and so both are defined and equal in M. This verifies axiom (β).

Now for axiom (ξ). Suppose t and s are closed terms defined in M, and $Ap(t,x) \cong Ap(s,x)$ is valid in M. Then for each closed term r such that T proves $r\downarrow$, 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 $\mathbf{T} \neq F$ holds since T does not prove $\mathbf{T} = \mathbf{F}$, because T is consistent. That completes the proof.

13 Partial lambda logic in terms of resolution and paramodulation

In the resolution and paramodulation axiomatization of lambda logic (and first order logic for that matter) we use the axiom $x = x$, since paramodulation does not include reflexivity. In partial lambda logic, we replace the axiom $x = x$ by the clause $-E(x)|x = x$, where $E(t)$ is a syntactic variant of $t\downarrow$. 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 7 (Soundness of lambda unification for LPT)
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.

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 $a\downarrow$.

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., Systems of illative combinatory logic complete for first order propositional and predicate calculus *Journal of Symbolic Logic* **58** (3), 89-108, 1993.

[3] 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.

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

[5] 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).

[6] Beeson, M., Lambda logic, in Basin, David; Rusinowitch, Michael (eds.) Automated Reasoning: Second International Joint Conference, IJCAR 2004, Cork, Ireland, July 4-8, 2004, Proceedings. Lecture Notes in Artificial Intelligence 3097, pp. 460-474, Springer (2004).

[7] 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).

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

[9] 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).

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