Theoretical Computer Science 22 (1983) 1-17
North-Holland Publishing Company

THE COMPLETENESS THEOREM FOR TYPING λ-TERMS

Roger HINDLEY
Department of Mathematics, University College of Swansea, Swansea SA2 8BP, United Kingdom

Communicated by D.M.R. Park
Received November 1981

Dedicated to Haskell B. Curry in his 81st year

Abstract. Rules for assigning type-schemes to untyped λ-terms are given, three different semantics are described, and the rules are proved complete with respect to two of these semantics. The type-schemes are built up from type-variables, not constants, by '$\to$'. The semantics are defined in arbitrary models of the untyped λ-calculus; such models do not come with a type-structure as part of their definition.

The fact that two distinct semantics are completely captured by one set of rules says that the usual type-language, with '$\to$' as its only connective, is not expressive enough to describe the differences between them.

I conjecture that the rules are also complete with respect to the third semantics.

Introduction

There are two ways of introducing types into a system of λ-calculus. In the traditional way, the types are expressions built up by one connective, '$\to$', from atomic constants. These constants are interpreted as fixed sets, for example '0' for the set of natural numbers, '1' for the set of truth-values, etc. And a type $(\alpha \to \beta)$ denotes a set of functions from α into β. Each λ-term comes with a unique built-in type, given to it as part of its definition. And this typing of terms restricts the class of terms that are well formed: (XY) is only called a term when the type of X has form $(\alpha \to \beta)$ and the type of Y is α.

Following Milner [14], I shall call the traditional method a *monomorphic* approach. It is the approach taken in Andrews [1] and the references therein, for example. In it, each term with type $(\alpha \to \beta)$ represents a function with domain α and range $\subseteq \beta$.

This is all very well, if we are using our λ-calculus to formalize the mathematicians' usual concept of function, where each function is a set of ordered pairs, equipped with a pre-defined domain and range.

But there is another concept of function, historically just as important as the set-theoretic one, for which the monomorphic approach is inadequate. In the 1920s, when λ-calculus (and its sister-system, combinatory logic) was invented, logicians did not automatically think of functions as sets of ordered pairs. Just as much in their minds was the concept of a function as an operation-process which, when given an input, operates on it to produce an output. Such a process can be defined by describing the action of the operation on an arbitrary input, without necessarily saying whether this action will have a result for a particular input; that is, without necessarily defining its domain. Alonzo Church, the inventor of λ-calculus, explains this very clearly in the introduction to his book [5]. Nowadays, of course, one would think of an operator as a programmed computer, and computation theory has brought the operator concept back into prominence again. (Though the operator concept in general is not intended to have the finiteness and effectiveness limitations of computers.)

(Not only may the domain of an operator be left undefined, but we can go further; for some very simple operators, it is reasonable to allow the domain to be the whole universe. They can even accept themselves as arguments. The simplest example is I, the operation of doing nothing at all. If we accept this as a well-defined concept, then surely it can be applied to itself. And in fact, '$I(I)=I$' causes no inconsistency anywhere.)

Now, can we get a type-theory which fits the operator concept of function?

The answer was first given by Curry in his theory of functionality. (Curry and Feys [8, Section 8C and following].) It was developed further in Curry [7], Hindley [11], and Ben-Yelles [4]. It was also discovered independently by Milner, Reynolds and Strachey, who wanted a more flexible approach to programming than the monomorphic system gave; they too did not like having to specify the domain and range of an operator before they called it for use. (See Milner [14] for a discussion and motivation.) Following Milner, I shall call the new approach *polymorphic*.

First of all, instead of types the polymorphic approach uses *type-schemes*, which contain *type-variables* as well as constants. And instead of incorporating the types into the terms as the terms are built up, we take the terms of the untyped λ-calculus and assign type-schemes to them by a formal system of axioms and deduction-rules. For example, to $\boldsymbol{I}$ (that is, $\lambda x.x$) we assign the scheme $(a\to a)$, where a is a type-variable. By this we mean that $\boldsymbol{I}$ acts as a function from any set A into itself. Thus it says that $\boldsymbol{I}$ is a 'universal' operator, with unrestricted domain.

Systems of rules for assigning types were first given by Curry in Curry and Feys [8, Chapter 9]. (They were first formulated for combinatory logic, not the lambda-calculus.) The simplest system, which Curry called 'F-deductions', has been motivated and analyzed in Curry [7], Hindley [11], Curry et al. [9], and Ben-Yelles [4].

A precise semantics for polymorphic type-assignment was first suggested by Dana Scott and by J.C. Reynolds. ([19, Problem II 4] and [15].) In fact there are several different interesting ways of interpreting type-schemes in a model of untyped λ-calculus, and three will be described in this paper. The first person to take up the study of type-scheme semantics was Ben-Yelles, in his thesis [4].

With any semantics, and any deductive system, there is a completeness problem: can we prove by the deduction-rules all the statements that are true in the semantics?

Ben-Yelles in [4] conjectured the completeness of Curry's type-assignment rules with respect to the simplest semantics of the three; and he proved it in many special cases.

This paper will give a complete completeness proof for the simplest semantics, and for one of the other two. It will cover both λ-terms and combinators. The paper will be self-contained; reference will be made to results in Hindley [11] and Ben-Yelles [4], but reading these results will not be necessary. The reader should know about the syntax of λ-conversion, however, as explained for example in Barendregt [3] or Curry and Feys [8]. He should also know a bit about models of untyped λ-calculi. But he need not know much, the only model used in this proof is the term model.

By the way, a completeness proof [2] has also been obtained independently by Barendregt, Coppo and Dezani for the simplest semantics. It is more roundabout than the one below; they first prove completeness for the Coppo–Dezani–Sallé type-language [6], using a more complicated model than here, and then deduce completeness for '→'-types by a conservative extension result. But these results, and the model used, have interest in their own right. (An independent completeness proof for Coppo–Dezani–Sallé type-schemes, using the simple model below, is in [13].)

Other completeness-related results are in Ben-Yelles [4]. Ben-Yelles proves several special cases of the completeness theorem, but he also proves stronger results which the methods of the present paper do not give. (See the end of Remark 3 in Section 3 below.)

It is worth noting that for two of the semantics, the complete formal system is the same. (Added in proof: it is the same for the third, too; see later.) So the same set of type-assignment statements is valid in different semantics. Thus it seems that the usual type-language, whose only connective is '→', is not expressive enough to show the differences between at least two of the interpretations. The question of what extensions of this language express these differences in a neat and simple way, is still open. The Coppo–Dezani–Sallé language of [6] might be strong enough.

I am very grateful to Gordon Plotkin and Henk Barendregt for suggesting improvements in the exposition of this paper, and to the referees for spending a lot of time and effort on earlier versions. I am also very grateful to Mario Coppo for an improved completeness proof for the quotient-set semantics, which I shall use here, with his permission, instead of my original one.

1. Basic concepts

All the main concepts will be defined here; any ancillary details not explained here can be found in [11, pp. 30–33]. I shall use the notation conventions of Curry and Feys [8] and Hindley [11], except that identity will be called '=' instead of '≡', the 'obs' of [8] and [11] will be called 'Terms', and the '$F\alpha\beta$' of [8] and [11] will here be called '$\alpha \to \beta$'. The combinatory and lambda systems will be treated together.

Terms (denoted by capital letters): λ-*terms* are built up from variables (called *term-variables*) thus:

from x, M, N *build* $(\lambda x.M), (MN)$.

There are no atomic constants. *Combinatory terms* are built up from $\boldsymbol{S}$ and $\boldsymbol{K}$ and term-variables thus: *from* M, N *build* (MN). The combinatory analogue of $\lambda x.M$ is a term called by Curry '$[x].M$' which can be defined in many ways. (For example [11, p. 44].) I shall call it '$\lambda x.M$' here, and its particular definition will not matter. $\mathrm{FV}(X)$ will be the set of all free variables of X if X is a λ-term, and the set of all variables in X if X is a combinatory term. ($\mathrm{FV}(\lambda x.M) = \mathrm{FV}(M) - \{x\}$.) Note that the terms are untyped; type-schemes will be assigned to some of them by the rules below.

Type-schemes $(\alpha, \beta, \gamma, \ldots)$ are built up from an infinity of *type-variables* $(a, b, c, \ldots)$ by the rule:

from α *and* β *build* $(\alpha \to \beta)$.

The type-variables are distinct from the term-variables. There are no type-constants.

Type-assignment statements are expressions αX, where α is a type-scheme and X a term. (Read it as 'assign α to X' or 'put X into the set α'.) X is called the *subject* of the statement.

A *basis* $\mathcal{B}$ is any finite or infinite set of statements whose subjects are variables, and such that no two statements in $\mathcal{B}$ have the same subject. *Subjects*$(\mathcal{B})$ is the set of all the subjects of statements in $\mathcal{B}$.

Definition 1. *The simplest formal system for type-assignment* is a Gentzen Natural Deduction system with 2 rules, called $\to$-introduction and $\to$-elimination:

$$(\to i)\,\frac{\begin{matrix}[\alpha x]\\ \vdots \\ \beta Y\end{matrix}}{(\alpha \to \beta)(\lambda x.Y)} \qquad (\to e)\,\frac{(\alpha \to \beta)X \quad \alpha Y}{\beta(XY)}.$$

Deductions are trees as usual, with premises at the tops of branches, and the conclusion at the bottom of the tree. Each time we use rule $(\to i)$, we 'cancel' (shown here by '[ ]') all occurrences of αx (if any) at the tops of branches above βY, that have not been cancelled previously. We are allowed to use $(\to i)$ even when there are no occurrences of αx above βY; this is called 'vacuous cancellation'. We are forbidden from using $(\to i)$ when there is an uncancelled premise γx with $\gamma \neq \alpha$, above βY.

For any basis $\mathcal{B}$,

$$\mathcal{B} \vdash_{\mathrm{F}} \alpha X$$

means that there is a deduction whose uncancelled assumptions are in $\mathcal{B}$ and whose conclusion is αX. In the case of empty $\mathcal{B}$ one says

$$\vdash_F \alpha X.$$

('$\vdash_F$' denotes deducibility only in this particular system (same as Curry's F-deducibility in [8]); deducibility relations in general are called '$\vdash$'.)

Exercise. Let $\boldsymbol{K}_\lambda = \lambda x.\lambda y.x$ and $\boldsymbol{S}_\lambda = \lambda x.\lambda y.\lambda z.xz(yz)$; then

$$\vdash_F (a \to (b \to a))\boldsymbol{K}_\lambda, \qquad \vdash_F ((a \to (b \to c)) \to ((a \to b) \to (a \to c)))\boldsymbol{S}_\lambda.$$

Remark 1. Note that we really have 2 systems here; one for λ-terms and one for combinatory terms. For combinatory terms the axiom-based system in [7] and [11, p. 32] is more natural than this one, but it is easy to see that the two systems are equivalent. ($\mathcal{B} \vdash \alpha X$ in one system iff $\mathcal{B} \vdash \alpha X$ in the other; use (15) on p. 44 of [11].)

Remark 2. In any decent semantics, interconvertible terms will be given the same interpretation. But in the above formal system, inter-convertible terms can have different type-schemes. (In fact there even exist two terms $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ such that $\boldsymbol{I}_1$ conv $\boldsymbol{I}_2$ and the sets

$$\{\alpha : \vdash \alpha \boldsymbol{I}_1\}, \qquad \{\alpha : \vdash \alpha \boldsymbol{I}_2\}$$

do not intersect, [11, p. 53].) Hence the above 2 rules cannot be complete with respect to any decent semantics. We must add a rule of invariance of type under conversion. For λ-terms, conversion may be β or $\beta\eta$-equality. For combinatory terms, one has 3 conversion relations; weak equality, β-strong equality, and $\beta\eta$-strong equality. The first is too weak to describe funtionality (see Section 3, Remark 4 after the completeness theorem), and the second has not a well-enough developed theory, so I shall stick to the third here (except in Section 3, Remark 3). The conversion relations will be called $=_{\lambda\beta}$, $=_{\lambda\beta\eta}$, $=_{c\beta\eta}$, and the corresponding reductions $\geqslant_{\lambda\beta}$, $\geqslant_{\lambda\beta\eta}$, $\geqslant_{c\beta\eta}$. (The definition of $=_{c\beta\eta}$ is in [8, Section 6C, p. 203]; and $\geqslant_{c\beta\eta}$ is the strong reduction of [8, Section 6F].) For an arbitrary one of these I shall say 'conv', 'reduces to'.

Definition 2. *The three type-assignment systems with equality* (one for each of $=_{\lambda\beta}$, $=_{\lambda\beta\eta}$, $=_{c\beta\eta}$) are defined by the following rules:

$$(\to i)\,\frac{\begin{matrix}[\alpha x]\\ \vdots \\ \beta Y\end{matrix}}{(\alpha \to \beta)(\lambda x.Y)} \qquad (\to e)\,\frac{(\alpha \to \beta)X \quad \alpha Y}{\beta(XY)}$$

$$(\text{eq})\,\frac{\alpha X}{\alpha Y}\,(\textit{if } X \textit{ conv } Y).$$

Deducibility will be called $\vdash_{\lambda\beta}$, $\vdash_{\lambda\beta\eta}$, $\vdash_{c\beta\eta}$, depending on the convertibility relation used in the equality rule. (Subscripts will be omitted whenever this causes no confusion.)

Substitution Lemma (Holds for $\lambda\beta$, $\lambda\beta\eta$, $c\beta\eta$). *If* $\mathcal{B} \vdash \alpha X$, *then* $\mathcal{B}^* \vdash \alpha^* X$, *where* $*$ *denotes substitution of any type-scheme for a type-variable.*

Equality Postponement Theorem (Holds for $\lambda\beta$, $\lambda\beta\eta$, $c\beta\eta$). *All applications of the equality rule can be pushed down to the end of a deduction. That is, if* $\mathcal{B} \vdash \alpha X$ *then there is an* X' *such that* X' *conv* X *and* $\mathcal{B} \vdash_F \alpha X'$.

Proof. Straightforward (see [4] and [8]). □

Incidentally, this theorem would fail for combinators if the conversion used was only weak equality. (Exercise.) But it would hold for weak equality if we had used the axiom-based system of [7] and [11] instead of the Gentzen system above.

Subject Reduction Theorem (Holds for $\lambda\beta$, $\lambda\beta\eta$, $c\beta\eta$). *If* $\mathcal{B} \vdash_F \alpha X$ *and* X *reduces to* Y, *then* $\mathcal{B} \vdash_F \alpha Y$.

Proof. Straightforward (see [4] and [8]). □

Corollary. *If* $\mathcal{B} \vdash_{\lambda\beta} \alpha X$ *and* $X \geq_{\lambda\beta\eta} Y$, *then* $\mathcal{B} \vdash_{\lambda\beta} \alpha Y$.

Proof. By equality-postponement, there is an $X' =_{\lambda\beta} X$ such that $\mathcal{B} \vdash_F \alpha X'$. By the Church–Rosser theorem, there is a W such that

$$X \geq_{\lambda\beta} W, \qquad X' \geq_{\lambda\beta} W.$$

Now apply to the reductions $X \geq_{\lambda\beta\eta} Y$ and $X \geq_{\lambda\beta} W$ the proof of the Church–Rosser theorem ([8, pp. 113–114]); there is a Z such that

$$Y \geq_{\lambda\beta} Z, \qquad W \geq_{\lambda\beta\eta} Z.$$

Then $X' \geq_{\lambda\beta\eta} Z$, so by the subject-reduction theorem $\mathcal{B} \vdash_F \alpha Z$. Hence by rule (eq), since the reduction from Y to Z is β,

$$\mathcal{B} \vdash \alpha Y. \quad \square$$

Normal Form Theorem (Holds for $\lambda\beta$, $\lambda\beta\eta$, $c\beta\eta$). *If* $\mathcal{B} \vdash \alpha X$, *then* X *has a normal form* X^* *and* $\mathcal{B} \vdash_F \alpha X^*$.

Proof. By equality-postponement, it is enough to prove the theorem for $\vdash_F$. Such a proof was first given for λ by Turing [10], and there is a detailed proof in [1, Proposition 2.7.3]. For combinators there is a proof in [8, Section 9F, Corollary

9.2], using a cut-elimination technique. (Incidentally, the theorem is also true for combinators and weak normal forms; this was shown by Sanchis [16].) □

Corollary (Holds for $\lambda\beta$, $\lambda\beta\eta$, $c\beta\eta$). *If $\mathcal{B} \vdash \alpha X$, and we define $\mathcal{B} \restriction X$ by*

$$\mathcal{B} \restriction X = \{\beta y : \beta y \in \mathcal{B} \text{ and } y \text{ free in } X\},$$

then $\mathcal{B} \restriction X \vdash \alpha X$.

Proof. Let $\mathcal{B} \vdash \alpha X$. Then by the normal form theorem, X has a normal form X^* and $\mathcal{B} \vdash_F \alpha X^*$. Let $\mathcal{B}_0$ be the set of all statements in $\mathcal{B}$ which actually occur as uncancelled premises of this deduction. Then by induction on $\vdash_F$, it is easy to see that

$$\mathit{Subjects}(\mathcal{B}_0) = \mathrm{FV}(X^*).$$

Now X reduces to X^*, and reduction does not introduce new free variables, so $\mathrm{FV}(X) \supseteq \mathrm{FV}(X^*)$. Hence

$$\mathcal{B} \restriction X \supseteq \mathcal{B}_0.$$

Therefore $\mathcal{B} \restriction X \vdash_F \alpha X^*$. Then by rule (eq), we get

$$\mathcal{B} \restriction X \vdash \alpha X. \quad \Box$$

Incidental note. Further syntactic results on these systems can be found in [11] and [4], though they will not be needed here. [11] was written in terms of combinators, but [4] has shown that the results hold for λ-terms too.

2. The simple semantics

Models: The semantics is based on the concept of *model of untyped λ-calculus*. Different authors define this concept slightly differently (the issues are discussed in [12]), but the differences will not matter here. Common to all definitions is that every model $\mathcal{D}$ has three things: a non-empty set D called the *domain*, a map $\bullet : D^2 \to D$ called *application*, and an *interpretation map* $[\![\ ]\!]$ which assigns to each term X and each map ρ (ρ : term-variables $\to D$), a member $[\![X]\!]_\rho$ of D such that

(i) $[\![x]\!]_\rho = \rho(x)$;

(ii) $[\![XY]\!]_\rho = [\![X]\!]_\rho \bullet [\![Y]\!]_\rho$;

(iii) *if* $\sigma(x) = \rho(x)$ *for all x free in X, then* $[\![X]\!]_\rho = [\![X]\!]_\sigma$;

(iv) $X\ \mathit{conv}\ Y \Rightarrow (\forall\rho)[\![X]\!]_\rho = [\![Y]\!]_\rho$.

The above properties are all that will be needed in the future proofs. (Most definitions require also that D have at least 2 members, but this will not be needed here.)

In fact, the only models needed in the completeness proof are the *term models* $\mathcal{T\!M}(\lambda\beta)$, $\mathcal{T\!M}(\lambda\beta\eta)$, $\mathcal{T\!M}(c\beta\eta)$. The term model of a convertibility relation has the set of all convertibility-classes of terms for its domain. (For all X let $[X]=\{Y: Y$ converts to $X\}$.) Then the map $\bullet$ is defined by

$$[X]\bullet[Y]=[XY].$$

And $[\![\quad]\!]$ is defined by

$$[\![X]\!]_\rho=[[Y_1,\ldots,Y_n/x_1,\ldots,x_n]X],$$

where $x_1,\ldots,x_n$ are the free variables of X, and $\rho(x_i)=[Y_i]$, and $[\cdots/\cdots]$ is simultaneous substitution.

In a term model, the simplest ρ is the following ρ_0:

$$\rho_0(x)=[x] \quad \textit{for all } x.$$

For this ρ_0 we have

$$[\![X]\!]_{\rho_0}=[X].$$

Valuations: Given any model $\mathscr{D}$, a *valuation of the type-variables* is any map V which assigns to each type-variable a subset of D. Any such V determines an *interpretation* $[\![\quad]\!]_V$ *of all the type-schemes* by the rules

(i) $[\![a]\!]_V=V(a)$;

(ii) $[\![\beta\to\gamma]\!]_V=\{d\in D: (\forall e)e\in[\![\beta]\!]_V \Rightarrow d\bullet e\in[\![\gamma]\!]_V\}$.

Satisfaction: A statement αX is *satisfied* by $\mathscr{D}$, ρ, V iff $[\![X]\!]_\rho\in[\![\alpha]\!]_V$. A set $\mathscr{B}$ is satisfied iff all its members are satisfied. Then we define

$$\mathscr{B}\models\alpha X \Leftrightarrow \textit{every } \mathscr{D}, \rho, V \textit{ satisfying } \mathscr{B} \textit{ also satisfies } \alpha X.$$

There are in fact 3 concepts here; $\models_{\lambda\beta}$, $\models_{\lambda\beta\eta}$, $\models_{c\beta\eta}$, according as 'every $\mathscr{D}$' ranges over models of the three equalities.

Discussion. The above semantics first appeared in print in Reynolds [15], and in [19, Problem II 4], which I believe was proposed by Dana Scott. I shall call it the *simple semantics*.

Two other possible semantics have also been proposed, which I shall call the *F-semantics* and the *quotient-set semantics*. These will be defined and discussed in Sections 4 and 5.

We shall see that the formal system consisting of $(\to i)$, $(\to e)$, (eq) is complete for the simple and the quotient-set semantics.

3. Completeness for the simple semantics

Soundness Theorem (Holds for $\lambda\beta$, $\lambda\beta\eta$, $c\beta\eta$). *If* $\mathscr{B}\vdash\alpha X$, *then* $\mathscr{B}\models\alpha X$ *in the simple semantics.*

Proof. Straightforward (see [4, Theorem 4.17]). □

Completeness Theorem (Holds for $\lambda\beta$, $\lambda\beta\eta$, $c\beta\eta$). *If $\mathcal{B} \vDash \alpha X$ in the simple semantics, then $\mathcal{B} \vdash \alpha X$.*

Proof. Note that $\mathcal{B}$ may be infinite, and $\mathcal{B}$ need not contain all the free variables of X. There will be two cases, according as the language has an infinity of term-variables that are not in $\mathcal{B}$, or not. Let *Termvars* be the set of all term-variables in the language. Now suppose $\mathcal{B}$, α, X are given such that $\mathcal{B} \vDash \alpha X$.

Case 1: *Termvars* $-$ *Subjects*$(\mathcal{B})$ *is infinite*. First extend $\mathcal{B}$ to a set $\mathcal{B}^+$ of statements in which each type-scheme in the type-language is assigned to an infinity of term-variables, and no term-variable is the subject of more than one statement, and no term-variable in $\mathcal{B}^+ - \mathcal{B}$ occurs in X. (For each type-scheme δ, take an infinity of distinct term-variables $y_{\delta,i}$ $(i = 1, 2, \dots)$ not in $\mathcal{B}$ or X, and such that

$$\delta \neq \gamma \Rightarrow (\forall i, j) y_{\delta,i} \neq y_{\gamma,j};$$

then define $\mathcal{B}^+$ to be $\mathcal{B} \cup \{\delta y_{\delta,i}: \mathit{all}\ \delta,\ \mathit{all}\ i\}$.)

Then take the term model for the equality in question ($\lambda\beta$, $\lambda\beta\eta$, or $c\beta\eta$), and define a valuation V by setting

$$V(a) = \{[Y]: \mathcal{B}^+ \vdash aY\}$$

for each type-variable a. Then when V is extended to all type-schemes, we have for all δ and all Y,

$$[Y] \in [\![\delta]\!]_V \Leftrightarrow \mathcal{B}^+ \vdash \delta Y. \tag{1}$$

(Proof later.)

Now $\mathcal{B} \vDash \alpha X$, so in particular, for the term model and ρ_0 and the above V, we have

$$[X] \in [\![\alpha]\!]_V.$$

By (1), $\mathcal{B}^+ \vdash \alpha X$. Hence by the corollary to the normal form theorem,

$$\mathcal{B}^+ \restriction X \vdash \alpha X.$$

But $\mathcal{B}^+ \restriction X = \mathcal{B} \restriction X$, because the only term-variables in $\mathcal{B}^+$ that are not in $\mathcal{B}$ are new variables not in X. Hence

$$\mathcal{B} \restriction X \vdash \alpha X.$$

Therefore, a fortiori, $\mathcal{B} \vdash \alpha X$, which is what we want for completeness. It only now remains to prove (1).

Proof of (1). Induction on δ. The basis is true by definition of V. For the induction step, let $\delta = (\beta \to \gamma)$. Then

$$\mathcal{B}^+ \vdash (\beta \to \gamma)Y \Rightarrow \forall Z(\mathcal{B}^+ \vdash \beta Z \Rightarrow \mathcal{B}^+ \vdash \gamma(YZ))$$

by $\to$-elimination

$$\Leftrightarrow \forall Z([Z]\in[\![\beta]\!]_V \Rightarrow [YZ]\in[\![\gamma]\!]_V)$$

by induction hypothesis

$$\Leftrightarrow [Y]\in[\![\beta\to\gamma]\!]_V$$

by definition of $[\![\ \ ]\!]_V$.

Conversely, suppose $\forall Z(\mathscr{B}^+\vdash\beta Z\Rightarrow\mathscr{B}^+\vdash\gamma(YZ))$. Choose Z to be a term-variable z not occurring in Y and such that $\mathscr{B}^+$ contains the statement βz. Then $\mathscr{B}^+\vdash\beta z$, so

$$\mathscr{B}^+\vdash\gamma(Yz).$$

Hence by $\to$-introduction,

$$\mathscr{B}^+-\{\beta z\}\vdash(\beta\to\gamma)(\lambda z.Yz),$$

and so, a fortiori,

$$\mathscr{B}^+\vdash(\beta\to\gamma)(\lambda z.Yz).$$

Hence

$$\mathscr{B}^+\vdash(\beta\to\gamma)Y.$$

(This comes by Rule (eq) if equality is $\beta\eta$; and if it is β, we use the corollary to the subject-reduction theorem.) This proves (1) and ends Case 1.

Case 2: Termvars $-$ Subjects($\mathscr{B}$) is finite. Project $\mathscr{B}$ and X into a subset of the variables whose complement is infinite, thus: list all the term-variables as $v_1, v_2, \ldots$, and let $\mathscr{B}'$ and X' be the result of replacing each v_i by v_{2i}. Then by routine calculations,

$$\mathscr{B}\vDash\alpha X \Leftrightarrow \mathscr{B}'\vDash\alpha X',$$

$$\mathscr{B}\vdash\alpha X \Leftrightarrow \mathscr{B}'\vdash\alpha X'.$$

Then Case 1 applied to $\mathscr{B}'$ and X' gives the result. □

Corollary 1 (Ben-Yelles [4]). *If in a term model, $[\![\beta]\!]_V\subseteq[\![\gamma]\!]_V$ for all valuations V, then $\beta=\gamma$.*

Proof. Let $\boldsymbol{I}$ be $\lambda x.x$ or $\boldsymbol{SKK}$. Then $\boldsymbol{I}X$ conv X for all X, so by the definition of $[\![\beta\to\gamma]\!]_V$, we have $[\boldsymbol{I}]\in[\![\beta\to\gamma]\!]_V$ for all V. Hence by completeness, $\vdash(\beta\to\gamma)\boldsymbol{I}$. Therefore by the principal type-scheme theorem ([4, Theorem 2.5] or [7, Theorem 1] or [11, Theorem 1]), $\beta\to\gamma$ is a substitution-instance of the principal type-scheme for $\boldsymbol{I}$. This type-scheme is $a\to a$, by [4] or [7] or [11]. Hence $\beta=\gamma$. □

Corollary 2 (Ben-Yelles [4]). *For all α and all X, the following are not valid in the term models*:

$$(a \to b)X,$$

$$aX,$$

$$\alpha(\lambda x.xx),$$

$$\alpha \boldsymbol{Y} \quad (\boldsymbol{Y} = \lambda x.(\lambda y.x(yy))(\lambda y.x(yy))).$$

Proof. The above statements are not provable in the formal systems. ($(a \to b)$ and a are not propositional tautologies, $\lambda x.xx$ is easily proved unstratified, and $\boldsymbol{Y}$ has no normal form.) □

Remark 3. The completeness theorem holds also for combinatory β-equality [8, p. 203]. *Proof*: translate to λ-terms, using the translation suggested by the $[x]$-definition in [8, p. 44], and then use completeness for $\lambda\beta$.

Remark 4. In formulating the equality rule for combinatory terms, could we have got away with using only weak equality instead of strong? This seems attractive because weak equality is so simple to define.

But the answer is 'no'. Weak equality does not have the property

$$X \text{ conv } Y \Rightarrow \lambda x.X \text{ conv } \lambda x.Y;$$

in particular, if X is $\boldsymbol{K}x(\boldsymbol{SII})$ and Y is x, then $X \geq_w Y$, but $\lambda x.X$ is

$$\boldsymbol{S}(\boldsymbol{S}(\boldsymbol{KK})\boldsymbol{I})(\boldsymbol{S}(\boldsymbol{S}(\boldsymbol{KS})(\boldsymbol{KI}))(\boldsymbol{KI}))$$

which is in normal form with respect to weak reduction ($\geq_w$). In any model $\mathscr{D}$ (of weak or strong equality), for any ρ and any V, if e is any member of $V(a)$ then

$$[\![\lambda x.X]\!]_\rho \bullet e = e \quad \text{by weak conversion}$$

$$\in V(a).$$

Hence

$$\vDash (a \to a)(\lambda x.X).$$

On the other hand, $\lambda x.X$ cannot be assigned a type-scheme by rules $(\to i)$ and $(\to e)$. (Because if it was, then $\boldsymbol{K}x(\boldsymbol{SII})$ and hence $\boldsymbol{SII}$ would have a type-scheme, which is impossible.) Thus, since $\lambda x.X$ is in normal form, a weak equality rule will not give it a type either. So a system with weak equality rule would be incomplete.

Remark 5. Given completeness, what happens to the problems listed at the end of [4]?

Problem 1 asked whether completeness would hold when the terms were restricted to being Church's λI-terms. The answer is yes, because the above proof stays valid under this restriction. (Under this restriction, vacuous cancellation is forbidden in rule ($\to i$).)

Problem 3 (the same as [19, Problem II 4]) asked about decidability and normal form theorems for the $\models$ relation. Given completeness, these now follow from the corresponding theorems for $\vdash$.

Problem 8 asked about the completeness of the Coppo–Dezani–Sallé system in [6]. It turns out that the rules in [6] are not complete, but they become complete when 3 extra ones (two trivial, one not) are added; [13]. Also Barendregt, Coppo and Dezani have a completeness proof [2] using a different method.

Problem 5 was a technical conjecture which follows trivially from completeness (though it would have been an interesting one if completeness had failed).

Problems 2, 4, 6, 7 are still open; I shall comment on them in turn.

Problem 2 asked, in essence, whether completeness, or any special case of it for a particular term X, could be proved using only valuations V such that $V(a)$ is a singleton set.

Problem 4 asked whether the usual decidability and normalization results for $\vdash$ remained true for the relation $\mathcal{D} \models \alpha X$, for a fixed $\mathcal{D}$, for example Scott's D_∞.

Problem 6 suggested that extending the type-language to allow certain infinitary types might provide a way of typing wider classes of terms, alternative to the Coppo–Dezani–Sallé system.

Problem 7 suggested studying type-scheme representable sets of terms; sets of form

$$\{X : \vdash \alpha X\}$$

for various α. One could also study, for various $\mathcal{D}$, ρ, V, the sets of the form

$$\{d : (\exists X) d = [\![X]\!]_\rho \in [\![\alpha]\!]_V\}.$$

(For the particular sets

$$\mathit{Terms}(\alpha) := \{X : \vdash \alpha X \text{ and } X \text{ in normal form}\},$$

Ben-Yelles has shown in his thesis that there is an algorithm which, given α, will decide what the cardinality of $\mathit{Terms}(\alpha)$ is $(0, 1, 2, 3, \ldots$ or $\infty)$, and will enumerate all its members.)

Remark 6 Here is an extra problem arising out of Ben-Yelles' work. Ben-Yelles proved completeness for several closed terms X, including $\boldsymbol{K}$, $\boldsymbol{S}$, $\boldsymbol{I}$ and the Church numerals $\lambda xy.x^n y$. But for these particular terms, his completeness proof gave also a stronger result, which the methods of the present paper do not give. For any model $\mathcal{D}$, let '$\mathcal{D} \models \alpha X$' mean

$$\forall \rho \forall V \{[\![X]\!]_\rho \in [\![\alpha]\!]_V\};$$

then what Ben-Yelles proved, for the above term X, was

$$\forall\alpha\{(\exists\mathcal{D}.\mathcal{D}\models\alpha X)\Rightarrow\ \vdash\alpha X\}. \tag{2}$$

(As a corollary, by soundness one gets that $(\exists\mathcal{D}.\mathcal{D}\models\alpha X)$ implies $(\forall\mathcal{D}.\mathcal{D}\models\alpha X)$.) The question is, for what terms X does (2) hold?

4. The F-semantics

In this section I shall assume that the reader has read some account of λ-calculus models; for example, Hindley and Longo [12]. I shall use the notation of [12] here.

As Dana Scott has pointed out, the key to a model of λ-calculus is the set F of all the objects that can be interpretations of terms of form $\lambda x.M$. To be precise, let $\mathcal{D}=\langle D,\bullet,[\![\ \ ]\!]\rangle$ be a model of $\lambda\beta$; then F is defined by

$$F=\{d\in D: d=[\![\lambda x.M]\!]_\rho \textit{ for some } x, M, \rho\}. \tag{3}$$

(M need not be closed.) By [12, Lemma 3.4], if d_1 and d_2 are in F and $d_1\bullet e=d_2\bullet e$ for all $e\in D$, then $d_1=d_2$. Thus F has at most one member in each extensional-equivalence-class in D. Also, by the same lemma, we have

$$d\in F\ \Leftrightarrow\ d=[\![\lambda x.zx]\!]_\rho \textit{ where } \rho(z)=d.$$

If $\mathcal{D}$ satisfies (η), then $F=D$. Because for all $d\in D$,

$$\begin{aligned} d &= [\![z]\!]_\rho && \text{where } \rho(z)=d,\\ &= [\![\lambda x.zx]\!]_\rho && \text{by } (\eta). \end{aligned}$$

Another way of looking at F is this. Each one-place function ϕ from D into D that is representable in D, has a representative in F. Although every member d of D represents a one-place function ϕ_d ('represents' means $\forall e\in D\ \phi_d(e)=d\bullet e$), the definition of $[\![\ \ ]\!]$ in the particular model in question picks out certain members as, in a sense, 'canonical representatives'. These form F. Thus the members of F can be regarded as functions in a stronger sense than the other members of D.

The *F-semantics* is a way of interpreting type-schemes that takes this into account. It is not as straightforward as the simple semantics, but if one views the members of F as *the* functions in D, then it seems more natural. In it, all type-schemes $\beta\to\gamma$ are interpreted as subsets of F, that is, as sets of 'functions'.

Definition 3. *The F-Semantics.* Let $\mathcal{D}$ be a model of $\lambda\beta$, and let F be defined by (3) above. For any map V assigning subsets of D to type-variables, $[\![\ \ ]\!]_V$ is defined by:

(i) $[\![a]\!]_V=V(a)$,

(ii′) $[\![\beta\to\gamma]\!]_V=\{d\in F: (\forall e)e\in[\![\beta]\!]_V\Rightarrow d\bullet e\in[\![\gamma]\!]_V\}$.

Satisfaction, etc. are defined just as in Section 2.

Remark 7. What about combinators? Well, in a model of any combinatory equality, the set F can be defined by (3), provided we read '$[x].M$' for '$\lambda x.M$'. (And in models of c$\beta\eta$ we have $F = D$, because (η) holds.) But the concept of F no longer has the significance in a model of combinatory equality that it had for λ. For one thing, F no longer consists of at most one member out of each extensional-equivalence-class. (For example, take a term model of c$\beta\eta$ formed entirely from closed terms (terms not containing variables); this model satisfies (η) but is not extensional [12, Proposition 8.13]; thus $D = F$ by above, but some extensional-equivalence-classes have more than one member.)

Theorem. *$\mathcal{B} \models_{\lambda\beta\eta} \alpha X$ in the* F*-semantics iff $\mathcal{B} \models_{\lambda\beta\eta} \alpha X$ in the simple semantics, iff $\mathcal{B} \vdash_{\lambda\beta\eta} \alpha X$. And the same holds also for c$\beta\eta$.*

Proof. In any model of $\lambda\beta\eta$ or c$\beta\eta$ we have $F = D$ by above. So for such models the F-semantics becomes the same as the simple semantics. □

Soundness Theorem $(\lambda\beta)$. *If $\mathcal{B} \vdash_{\lambda\beta} \alpha X$, then $\mathcal{B} \models_{\lambda\beta} \alpha X$.*

Proof. Straightforward induction on $\vdash_{\lambda\beta}$. Note that in Rule $(\rightarrow i)$ the conclusion is $\lambda x.M$ and hence is in F. □

Corollary. *If $\mathcal{B} \vdash_{\lambda\beta} (\gamma \rightarrow \delta)X$, and $\mathcal{D}$, ρ, V satisfy all the statements in $\mathcal{B}$, then $[\![X]\!]_\rho \in F$.*

Completeness Conjecture $(\lambda\beta)$.[1] *If $\mathcal{B} \models_{\lambda\beta} \alpha X$ in the* F*-semantics, then $\mathcal{B} \vdash_{\lambda\beta} \alpha X$.*

5. The quotient-set semantics

Dana Scott describes this semantics in [17, Section 7, pp. 560 ff.], and proves some basic results about it in the model Pω. He discusses it in [18], at the end of Section 5.

When we interpret type-schemes as sets of 'functions', it may be that we would like the criterion for regarding different functions as 'equal' to be different for different types. The quotient-set semantics takes this into account

Given a model $\mathcal{D}$, let S be any subset of D and let $\sim$ be any equivalence relation on S. The *quotient set* $S/\sim$ is, as usual, the set of all the equivalence-classes of members of S.

Definition 4. *The quotient-set semantics.* Let V assign to each type-variable b a subset $V(b)$ of D, and an equivalence relation $\sim_b$ on $V(b)$. Then define

$$[\![b]\!]_V = V(b),$$

[1] (Added in proof). A proof is given in: R. Hindley, Curry's type-rules are complete with respect to F-semantics too, *Theoret. Comput. Sci.* 22 (1983) 127-133; this issue.

$$[\![\beta\to\gamma]\!]_V=\{d\in D:\forall e\in D\bullet e\in[\![\beta]\!]_V\Rightarrow d\bullet e\in[\![\gamma]\!]_V$$
$$\text{and }\forall e_1,e_2\in[\![\beta]\!]_V\bullet e_1\sim_\beta e_2\Rightarrow d\bullet e_1\sim_\gamma d\bullet e_2\};$$

$$d_1\sim_{\beta\to\gamma}d_2\Leftrightarrow\forall e_1,e_2\in[\![\beta]\!]_V(e_1\sim_\beta e_2\Rightarrow d_1\bullet e_1\sim_\gamma d_2\bullet e_2);$$

$$[\![\alpha]\!]^*_V=[\![\alpha]\!]_V/\sim_\alpha.$$

Satisfaction, etc. are defined as usual.

Informally, the set $[\![\beta\to\gamma]\!]$ is the set of all 'functions' from $[\![\beta]\!]$ into $[\![\gamma]\!]$ which respect the equivalence relation $\sim_\beta$; that is, such that equivalent arguments produce equivalent results.

For any α and any $d\in[\![\alpha]\!]$, let a' be the $\sim_\alpha$-equivalence-class containing d. Then define, for $d\in[\![\beta\to\gamma]\!]$ and $e\in[\![\beta]\!]$,

$$d'\bullet e'=(d\bullet e)'.$$

The above definition of $[\![\beta\to\gamma]\!]$ ensures that $d'\bullet e'$ is uniquely defined. Thus $d\in[\![\beta\to\gamma]\!]$ implies that d' acts as a function over the quotient-set $[\![\beta]\!]^*$. Every stratified term X (i.e. every X such that $(\exists\alpha)\vdash\alpha X$) can be interpreted as a member of a quotient-set, $[\![\alpha]\!]^*$, and for each $\mathscr{D}$ and V these sets form a model of typed combinatory logic.

Note. Dana Scott defines the above semantics not in terms of quotient sets, but simply by the relations $\sim_\alpha$; from these the sets $[\![\alpha]\!]$ can be defined, of course, as $\{d: d\sim_\alpha d\}$. Mathematically this is neater than the view given here, but I hope that the view here may be easier for someone meeting the concepts for the first time to assimilate. After (or before!) reading this version, the reader should go through the account in [17, Section 7, pp. 560 ff.] or [18, end of Section 5].

Soundness Theorem (Holds for $\lambda\beta$, $\lambda\beta\eta$, $c\beta\eta$). *If $\mathscr{B}\vdash\alpha X$, then $\mathscr{B}\models\alpha X$ in the quotient-set semantics.*

Proof. By induction on '$\vdash$'. To make the induction work, we must prove also that if δy is any statement in $\mathscr{B}$, then for all $\mathscr{D}$, ρ, V,

$$e\sim_\delta f\Rightarrow[\![X]\!](\rho^e_y)\sim_\alpha[\![X]\!](\rho^f_y),\tag{4}$$

where ρ^e_y is ρ with $\rho(y)$ changed to e, and I have written $[\![X]\!](\rho)$ for $[\![X]\!]_\rho$.

Basis: If $\alpha X\in\mathscr{B}$, then $\mathscr{B}\models\alpha X$ by definition. Also, X is a variable x, so we must prove (4) when $y=x$ and $\delta=\alpha$:

$$e\sim_\alpha f\Rightarrow[\![x]\!](\rho^e_x)=e\sim_\alpha f=[\![x]\!](\rho^f_x).$$

Induction step: The cases of (eq) and $(\to e)$ are easy. For $(\to i)$, suppose

$$\alpha=\beta\to\gamma,\qquad X=\lambda x.M,\qquad \mathscr{B},\beta x\vdash\gamma M.$$

And let (4) hold for M. Take any $\mathscr{D}$, ρ, V satisfying $\mathscr{B}$. We must show that $[\![\lambda x.M]\!]_\rho \in [\![\beta \to \gamma]\!]$, and for this we must show that it maps $[\![\beta]\!]$ into $[\![\gamma]\!]$, and that it respects $\sim_\beta$.

Note that $\mathscr{D}$ satisfies (at least) all $\lambda\beta$-equations, so for all $e \in [\![\beta]\!]$ we have

$$[\![\lambda x.M]\!](\rho) \bullet e = [\![(\lambda x.M)x]\!](\rho^e_x) = [\![M]\!](\rho^e_x). \tag{5}$$

Also, $\mathscr{D}$, ρ^e_x, V satisfy $\mathscr{B}$ and βx. So by the induction hypothesis

$$[\![M]\!](\rho^e_x) \in [\![\gamma]\!].$$

Hence $[\![\lambda x.M]\!]_\rho$ maps $[\![\beta]\!]$ into $[\![\gamma]\!]$.

To show that $[\![\lambda x.M]\!]_\rho$ respects $\sim_\beta$, let $e \sim_\beta f$ and apply (4) for M, with $\delta = \beta$ and $y = x$.

Finally, we must prove (4) for $\lambda x.M$. Let $\delta y \in \mathscr{B}$ and let $e \sim_\delta f$. By change of bound variables we may assume $y \neq x$. We must show

$$[\![\lambda x.M]\!](\rho^e_y) \sim_{\beta\to\gamma} [\![\lambda x.M]\!](\rho^f_y).$$

By (5) and the definition of $\sim_{\beta\to\gamma}$, this holds iff

$$[\![M]\!](\rho^{eg}_{yx}) \sim_\gamma [\![M]\!](\rho^{fh}_{yx}) \quad (\forall g \sim_\beta h).$$

But the latter is true, by (4) for M. This proves soundness. $\square$

Completeness Theorem (Holds for $\lambda\beta$, $\lambda\beta\eta$, $c\beta\eta$). *If $\mathscr{B} \models \alpha X$ in the quotient-set semantics, then $\mathscr{B} \vdash \alpha X$.*

Proof. (Due to Mario Coppo, replacing an earlier longer proof.) The simple semantics is a special case of the quotient-set semantics.

More precisely, let $\mathscr{D}$ be any model of $\lambda\beta$ or $\lambda\beta\eta$ or $c\beta\eta$, and let V be any interpretation of type-variables in the simple semantics. Then there is a V^* in the quotient-set semantics such that for all ρ, α, X, αX is satisfied by $\mathscr{D}$, ρ, V^* in the quotient-set semantics iff αX is satisfied by $\mathscr{D}$, ρ, V in the simple semantics.

V^* is defined as follows: for all type-variables b,

$$V^*(b) = V(b),$$

$$d_1 \sim_b d_2 \Leftrightarrow d_1, d_2 \in V(b).$$

(That is, $\sim_b$ makes all members of $V(b)$ equivalent.) It is easy to deduce that for all α,

$$[\![\alpha]\!]_{V^*} = [\![\alpha]\!]_V,$$

$$d_1 \sim_\alpha d_2 \Leftrightarrow d_1, d_2 \in [\![\alpha]\!]_V.$$

Hence for all X, α, ρ,

$$[\![X]\!]_\rho \in [\![\alpha]\!]_{V^*} \Leftrightarrow [\![X]\!]_\rho \in [\![\alpha]\!]_V.$$

Finally, if $\mathcal{B} \models \alpha X$ in the quotient-set semantics, then by above, $\mathcal{B} \models \alpha X$ in the simple semantics; and hence $\mathcal{B} \vdash \alpha X$ by completeness for the simple semantics. □

References

[1] P.B. Andrews, Resolution in type theory, *J. Symbolic Logic* **36** (1971) 414–432.

[2] H. Barendregt, M. Coppo and M. Dezani, A filter lambda model and the completeness of type-assignment *J. Symbolic Logic*, to appear.

[3] H. Barendregt, The type free lambda calculus, in: J. Barwise, Ed., *Handbook of Mathematical Logic* (North-Holland, Amsterdam, 1977).

[4] C.-B. Ben-Yelles, Type-assignment in the lambda-calculus: syntax and semantics, Doctoral thesis, University College of Swansea (1979). Author's address: Institut de Mathematiques, B. P. 9, Dar el Beida, Algiers, Algeria.

[5] A. Church, *Calculi of Lambda Conversions* (Princeton University Press, Princeton, NJ, 1941).

[6] M. Coppo, M. Dezani and B. Venneri, Principal type-schemes and λ-calculus semantics, in: *To H.B. Curry* (Academic Press, New York, 1980).

[7] H.B. Curry, Modified basic functionality in combinatory logic, *Dialectica* **23** (1969) 83–92.

[8] H.B. Curry and R. Feys, *Combinatory Logic, I* (North-Holland, Amsterdam, 1958).

[9] H.B. Curry, J.P. Seldin and J.R. Hindley, *Combinatory Logic, II* (North-Holland, Amsterdam, 1972).

[10] R.O. Gandy, An early proof of normalisation by A.M. Turing, in: *To H.B. Curry* (Academic Press, New York, 1980).

[11] R. Hindley, The principal type-scheme of an object in combinatory logic, *Trans. Amer. Math. Soc.* **146** (1969) 29–60.

[12] R. Hindley and G. Longo, Lambda-calculus models and extensionality, *Z. Math. Logik* **26** (1980) 289–310.

[13] R. Hindley, The simple semantics for Coppo–Dezani–Sallé types, *Proc. 5th Symposium on Programming*, Lecture Notes in Computer Science **137** (Springer, Berlin, 1982) 212–226.

[14] R. Milner, A theory of type polymorphism in programming, *J. Comput. System Sci.* **17** (1978) 348–374.

[15] J.C. Reynolds, Towards a theory of type structure, Lecture Notes in Computer Science **19** (Springer, Berlin, 1974) 408–425.

[16] L.E. Sanchis, Functionals defined by recursion, *Notre Dame J. Formal Logic* **8** (1967) 161–174.

[17] D. Scott, Data types as lattices, *SIAM J. Comput.* **5** (1976) 522–587.

[18] D. Scott, Lambda-calculus: some models, some philosophy, in: J. Barwise et al., Eds., *The Kleene Symposium* (North-Holland, Amsterdam, 1980).

[19] Various authors, Open problems, Lecture Notes in Computer Science **37** (Springer, Berlin, 1975) 367–370.