Formal Aspects of Computing (1992) 4: 470–486
© 1992 BCS

Formal Aspects of Computing

Types with Intersection: An Introduction

J. Roger Hindley
Mathematics Department, University College of Swansea, Swansea, UK

Keywords: Combinator; Intersection; Lambda-calculus; Types

Abstract. This paper is an informal introduction to the theory of types which use a connective " $\wedge$ " for the intersection of two types and a constant "ω" for a universal type, besides the usual connective "$\rightarrow$" for function-types. This theory was first devised in about 1977 by Coppo, Dezani and Sallé in the context of λ-calculus and its main development has been by Coppo and Dezani and their collaborators in Turin. With suitable axioms and rules to assign types to λ-calculus terms, they obtained a system in which (i) the set of types given to a term does not change under λ-conversion, (ii) some interesting sets of terms, for example the solvable terms and the terms with normal form, can be characterised exactly by the types of their members, and (iii) the type-apparatus is not so complex as polymorphic systems with quantifier-containing types and therefore probably not so expensive to implement mechanically as these systems.

There are in fact several variant systems with different detailed properties. This paper defines and motivates the simplest one from which the others are derived, and describes its most basic properties. No proofs are given but the motivation is shown by examples. A comprehensive bibliography is included.

1. Introduction

1.1. Motivation

Types were first introduced into logic by Bertrand Russell in the early 1900s, and into a λ-calculus-based system by Alonzo Church [Chu40]. In Church's system the only types are atomic constants (to denote particular sets) and types of form $(\sigma \rightarrow \tau)$, where σ and τ are any already-built types. Each λ-term has a unique built-

Correspondence and offprint requests to; J. R. Hindley, Mathematics Department, University College of Swansea, Swansea SA2 8PP, Wales, UK.

in type. Typed terms denote functions, and a type $(\sigma \to \tau)$ is intended to denote a set of functions f such that

$$\text{Domain}(f) = \sigma, \quad \text{Range}(f) \subseteq \tau \tag{1}$$

That is, f has type $(\sigma \to \tau)$ when f accepts inputs in σ (and only inputs in σ) and for each x in σ the corresponding output fx is in τ. The Russell–Church approach to types is summarised in [HiS86], Chap. 13.

Another approach to types was pioneered by Haskell Curry, beginning in the mid 1930s with systems based on combinatory logic. In it, types can contain variables as well as constants, and $(\sigma \to \tau)$ is interpreted as a set of functions or operators f such that

$$(\forall x)[(x \in \sigma \text{ and } fx \text{ defined}) \Rightarrow fx \in \tau] \tag{2}$$

In Curry's interpretation a statement that f has type $(\sigma \to \tau)$ makes no commitment as to what inputs f will accept; all it says is that if x is accepted (i.e. if fx is defined), then fx is in τ. In particular Curry's interpretation allows the possibility that the domain of f is larger than σ; thus one operator can in principle have many types, and in fact the system in [Cur69] gave each combinator an infinite number of types. In Curry's systems the types are not built into the formal terms, but assigned to untyped terms by axioms and rules. Those terms that receive types are called *stratified* or *typable*. For more on the basic Curry system see [HiS86], Chap. 14 (for combinatory logic) and Chap. 15 (for λ-calculus).

Curry's system focuses attention on the typable terms. The set of all typable terms is recursively decidable, closed under substitution, contains the basic combinators, and every member has a normal form. It can be argued that finding an easy-to-describe set with these properties is the main purpose of a type-theory.

But there is another point of view. Consider the famous fixed-point combinator

$$\mathsf{Y} \equiv \lambda x \cdot (\lambda y \cdot x(yy))(\lambda y \cdot x(yy))$$

for example. It is not typable, because it has no normal form. But Y has considerable practical importance (for example, see [HiS86], Chap. 3, Sect. 3B), and a theory that simply excludes it seems rather over-restrictive. Can we find a more generous type-theory, one that assigns a type to Y? This is the first question that motivated Mario Coppo and Mariangiola Dezani and, independently, Patrick Sallé, in the late 1970s.

Of course, in one sense the answer is "yes": simply introduce a new type-constant ω with axioms saying that all terms have type ω. But before we call this trick trivial and push it aside, we must state more clearly what we expect a type-assignment system to do. Coppo and Dezani took as their aim, that a statement "M has type τ" should describe the computational behaviour of the term M as an operator in some significant way. This is a reversal of attitude from Russell, Church and the basic system of Curry; we are no longer going to retreat to a small safe world of typable terms, shocked at the difficult behaviour of such a term as Y, but we shall bravely accept each term for what it is, and try to assign to the term a set of types that describes its behaviour.

Another defect of the Curry system is that the set of types assignable to a term may vary under λ-conversion. Can this defect be removed? Of course the set of a term's types could easily be made invariant under conversion by introducing a rule

$$\textit{if } \mathrm{M} \textit{ has type } \tau \textit{ and } \mathrm{M} \textit{ converts to } \mathrm{N}, \textit{ then give } \mathrm{N} \textit{ the type } \tau \tag{Eq}$$

But this rule is not decidable, because the convertibility relation is not decidable, so this answer is not very satisfactory; the real problem is to find decidable rules that give the same effect (and are reasonably neat).

Coppo, Dezani and Sallé answered both the above questions at once by enlarging the type-language; they introduced a "universal" type-constant ω and a new type-building operation $(\sigma \wedge \tau)$ to denote the intersection of two sets. They dealt with the second question by giving very natural and neat type-assignment rules which are decidable and yet make a term's set of types invariant under conversion. And as an answer to the first question, they proved that two computationally interesting sets of terms, not all of which are typable in Curry's sense, can be exactly characterised by the types of their members.

1.1.1. The Literature

Most of the development of the field has been by the group led by Mariangiola Dezani in Turin; see the papers by Barendregt, Coppo, Dezani, Hindley, Honsell, Longo, Margaria, Ronchi della Rocca, Venneri and Zacchi ([BCD83], [Cop80], [CoD78], [CoD80], [CDH78], [CDS79], [CDV80], [CDV81], [CDZ87], [DeH92], [DeM84], [DeM86], [Hin82], [HoR91], [Ron82], [RoV84]). Their work started around 1977. For the newcomer, [CDV81] is a good reference to the main ideas and includes motivation, but its notation is no longer used nowadays. [BCD83] contains a clear but condensed summary of the syntax and main results and proofs for the simplest system. The papers before 1983 used a more primitive formalism than the smoother treatment introduced in [BCD83]; the precise connection between the two formalisms was examined in [Hin82].

Intersection types have recently been used by John Reynolds and introduced into a programming language, Forsythe; see [Rey88] and [Rey89].

Besides the above mainstream there have been two other contributions to the field. First, Patrick Sallé invented intersection-types independently at around the same time as Coppo and Dezani; see [Sal78] and [CDS79].

Also, Garrel Pottinger has given an independent account of intersection from a propositional-logic point of view, in which $\sigma \wedge \tau$ is seen as meaning that σ and τ are both true and both have the same proof in some sense, see [Pot80]; this view has been developed further in [Lop85a], [Lop85b] and [Min89]. As Pottinger has pointed out, the main problem with this view is in making "some sense" precise.

1.1.2. This Paper

The present paper is an outline exposition of the most basic of the systems developed by Coppo and Dezani, with proofs omitted but with examples included to show the motivation. It refers to [HiS86] for background λ-calculus and type-theory.

2. Basics

2.1. Background Notation

Terms here are λ-terms. The same notation is used here for them as in [HiS86], Chap. 1. Variables in terms are called *term-variables* here to distinguish them from type-variables. *Types* will be defined below.

Combinators are λ-terms without free variables. The following will be used in examples later:

$$\mathsf{I} \equiv \lambda x \cdot x, \quad \mathsf{K} \equiv \lambda xy \cdot x, \quad \mathsf{S} \equiv \lambda xyz \cdot xz(yz)$$

For λ-terms M and N, the relations *M is identical to N, M β-reduces to N, M β-converts to N* are denoted, respectively, by

$$M \equiv N, \quad M \rhd_\beta N, \quad M =_\beta N$$

Type-assignment (TA-) Statements are expressions $M:\sigma$, where M is any λ-term and σ any type. (Older notations for $M:\sigma$ were "σM" in papers by Curry, Sallé and the Coppo-Dezani group and "$M \varepsilon \sigma$" in [HiS86].)

A *basis* $\mathscr{B}$ is any finite set of TA-statements of form $\{x_1:\sigma_1, x_2:\sigma_2, \ldots\}$. Note that $x_1, x_2, \ldots$ need not be distinct, so a basis $\mathscr{B}$ can assign a number of types to one term-variable. This number is finite, since $\mathscr{B}$ is finite. (In contrast, in basic Curry-style type theory the number is always assumed to be 1.)

2.2. Definitions

2.2.1. Types

We assume given an infinity of *type-variables*, a, b, c, ..., and one *type-constant*, ω, called the *universal type.*

(i). The type-variables and ω are types (called *atoms*).
(ii). If σ, τ are types, then so are $(\sigma \to \tau)$, $(\sigma \wedge \tau)$.

Notation. Greek letters ρ, σ, τ, μ denote types. *Parentheses* are often omitted from types, and should be restored by *association to the right*, for example

$$\rho \wedge \sigma \wedge \tau \wedge \mu \equiv (\rho \wedge (\sigma \wedge (\tau \wedge \mu)))$$
$$\rho \to \sigma \to \tau \to \mu \equiv (\rho \to (\sigma \to (\tau \to \mu)))$$

(Contrast term-notation, which uses association to the left!)

Informal Interpretation. Each type is intended to denote a set: $\sigma \wedge \tau$ denotes the intersection of σ and τ; ω denotes the universe of all objects; and $\sigma \to \tau$ is intended to denote a set of partial operators which, when applied to an input in σ, produce an output in τ, if they produce any output at all. (If an operator has type $\sigma \to \tau$, it may also accept inputs that are not in σ, and for such inputs its output need not be in τ.)

Notation. Types that do not contain ω or ∧ will be called *arrow-types.*

Warning. The reader who has met systems of typed terms before should never think of the type-assignment systems below as mere notational variants of typed systems. They will have many common features but they will have a much more flexible syntax. Further, and perhaps more importantly, they will differ profoundly in semantics. The core of this difference lies in the way $\sigma \to \tau$ is interpreted, as seen in the contrast between (1) and (2) above. In (1), which may be called the *Church interpretation* of $\sigma \to \tau$, the members of $\sigma \to \tau$ are functions f whose domain of acceptable inputs is exactly σ;

$$\text{Domain}(f) = \sigma, \quad \text{Range}(f) \subseteq \tau$$

In (2), the *Curry interpretation*, the members of $\sigma \to \tau$ are partial operators f whose domain need not be the whole of σ, and need not be restricted to σ; all that the Curry interpretation demands is

$$(\forall x)\,[(x \in \sigma \text{ and } fx \text{ defined}) \Rightarrow fx \in \tau]$$

2.2.2. *The Coppo–Dezani (CD) Type-Assignment System* $\mathrm{TA}_\lambda(\wedge, \omega)$

This has an infinite set of axioms, one for each term M, summarised in one axiom-scheme

$$M : \omega \qquad (\omega)$$

It has six rules (in the style of Gerhard Gentzen's "Natural Deduction" with elimination and introduction rules for each of $\rightarrow$ and $\wedge$, see the notes below):

$$\frac{M : (\sigma \rightarrow \tau) \quad N : \sigma}{(MN) : \tau} \qquad (\rightarrow E)$$

$$\frac{\begin{matrix}[x : \sigma] \\ \vdots \\ \vdots \\ M : \tau\end{matrix}}{(\lambda x \cdot M) : (\sigma \rightarrow \tau)} \qquad (\rightarrow I)$$

$$\frac{M : (\sigma_1 \wedge \sigma_2)}{M : \sigma_1} \quad \frac{M : (\sigma_1 \wedge \sigma_2)}{M : \sigma_2} \qquad (\wedge E)$$

$$\frac{M : \sigma_1 \quad M : \sigma_2}{M : (\sigma_1 \wedge \sigma_2)} \qquad (\wedge I)$$

$$\frac{(\lambda x \cdot Mx) : \sigma}{M : \sigma} \quad (\text{if x is not free in M}) \qquad (\eta_1)$$

Using Rule ($\rightarrow$ I).

(a). When we use ($\rightarrow$ I), we must *cancel* or *discharge* (enclose in brackets) each occurrence of $x : \sigma$ as an assumption above $M : \tau$ that has not been previously cancelled.

(b). Rule ($\rightarrow$ I) is allowed to be used when there are no occurrences of $x : \sigma$. (This is called *vacuous cancellation.*)

(c). Rule ($\rightarrow$ I) must not be used if an uncancelled non-axiom assumption with form $x : \sigma'$ occurs above $M : \tau$ with $\sigma' \not\equiv \sigma$.

(d). If $\sigma \equiv \omega$, then for each occurrence of $x : \omega$ as an assumption in the deduction of $M : \tau$ we have a choice; to view it as an assumption and cancel it, or to view it as an ω-axiom and leave it uncancelled.

(e). For more details on the use of ($\rightarrow$ I) and ($\rightarrow$ E) see [HiS86], Sects 15.1–15.6.

Notation. Deductions are built, by the above rules, in the form of trees with assumptions at the tops of branches and the conclusion at the bottom of the tree. Let $\mathscr{B}$ be any basis $\{x_1 : \sigma_1, x_2 : \sigma_2, \ldots\}$ and $M : \tau$ be any TA-statement. Iff there is a deduction of $M : \tau$ with all non-axiom assumptions cancelled except those in the basis $\mathscr{B}$, we say

$$\mathscr{B} \vdash M : \tau$$

Iff $\mathscr{B}$ is empty, we call the deduction a *proof* and say M *has type* τ, or

$$\vdash \mathrm{M}:\tau$$

The Curry-Style Systems TA_λ, $\mathrm{TA}_{\lambda=}$. As a contrast to the CD system, let us look in passing at two systems based on arrow-types.

First, TA_λ: its types are arrow-types and its rules are ($\rightarrow$ I) and ($\rightarrow$ E). We shall not need its details here; they are in [HiS86], Sect. 15.6. (There was an α-invariance rule in its definition there, but when bases of assumptions contain no composite terms, as here, it can be shown to be redundant.) TA_λ is essentially equivalent to the combinatory logic system in [Cur69] with no type- and term-constants.

Next, $\mathrm{TA}_{\lambda=}$: it is defined by rules ($\rightarrow$ I), ($\rightarrow$ E) and the following rule:

$$\frac{\mathrm{M}:\tau \quad \mathrm{M} =_\beta \mathrm{N}}{\mathrm{N}:\tau} \qquad (\mathrm{Eq}\beta)$$

Iff M$:\tau$ is provable or deducible from some basis in TA_λ (or ($\mathrm{TA}_{\lambda=}$), we say M is *typable* or *stratified* in TA_λ (or $\mathrm{TA}_{\lambda=}$).

Remark. The roles of $\wedge$ and ω in $\mathrm{TA}_\lambda(\wedge,\omega)$ will now be clarified by some examples. The role of rule (η_1) will be explained in Section 4.

2.3. Examples

2.3.1. The Role of $\wedge$

Consider the term $\lambda x\cdot xx$. In the Curry-style systems this is well known to be untypable. But in the CD system it is given a type by the proof in Fig. 1.

$$\dfrac{\dfrac{\dfrac{[x:(a \wedge (a\rightarrow b))]}{x:a\rightarrow b}\,(\wedge\,\mathrm{E}) \qquad \dfrac{[x:(a \wedge (a\rightarrow b))]}{x:a}\,(\wedge\,\mathrm{E})}{xx:b}\,(\rightarrow \mathrm{E})}{(\lambda x\cdot xx):(a \wedge (a\rightarrow b))\rightarrow b}\,(\rightarrow \mathrm{I})\text{ cancel } x:(a \wedge (a\rightarrow b))$$

Fig. 1.

Informally, this proof says that, although we may not like a universe in which an object x can be both function and argument at once, if there did exist such objects then self-application would have the type shown in Fig. 1. This information about $\lambda x\cdot xx$ is non-trivial, and is certainly more than the Curry-style systems give us by just dismissing $\lambda x\cdot xx$ as untypable.

2.3.2. Vacuous Cancellation and the Role of ω

In the Curry-style systems, cancellation allows us to prove such statements as

$$\mathsf{K}:a\rightarrow(b\rightarrow a)$$

In the CD system it also combines with the ω-axioms to give new types to well-known terms such as I, as in the proof in Fig. 2.

The statement $\mathsf{I}:(\omega\rightarrow\omega)\rightarrow\omega$ says informally that the identity-operation takes members of $\omega\rightarrow\omega$ and changes them into members of ω. Since the identity

$$\cfrac{\begin{matrix}(\omega\text{-axiom})\\ \mathrm{x}:\omega\end{matrix}}{\mathsf{I}:(\omega\to\omega)\to\omega.}\;(\to\mathrm{I})\text{ cancel vacuous }\mathrm{x}:\omega\to\omega$$

Fig. 2.

operation actually changes nothing, this says that members of $\omega\to\omega$ are already members of ω, i.e. that $\omega\to\omega$ is a subset of ω.

By the way, in Fig. 2 a vacuous statement containing x was cancelled, even though x occurred free in an axiom. This is allowed, and the same device will be used again in the next example.

2.3.3. The Fixed-Point Combinator and the Role of ω

Recall the fixed-point combinator

$$\mathrm{Y}\equiv\lambda\mathrm{x}\cdot\mathrm{ZZ},\quad\text{where}\quad \mathrm{Z}\equiv\lambda\mathrm{y}\cdot\mathrm{x(yy)}$$

The proof in Fig. 3 will give a type to Y which describes Y's operational behaviour in a non-trivial way.

But first, let us look informally at what this type could be. The characteristic property of Y is

$$\mathrm{F(YF)}=_\beta\mathrm{YF}\quad\text{(for all terms F)}$$

Informally speaking, if F represents an operator, let its range be τ. This means that an arbitrary input x gives rise to an output Fx in τ if there is any output at all, and in type-language this says that the type of F is $\omega\to\tau$. Since YF = F(YF), YF is in the range of F, so YF should have type τ. Hence Y should have type

$$(\omega\to\tau)\to\tau$$

for each type τ. And in fact Y does get these types in the CD system, as Fig. 3 shows.

$$\cfrac{\cfrac{\cfrac{[\mathrm{x}:\omega\to\tau]\qquad \begin{matrix}(\omega\text{-axiom})\\ \mathrm{yy}:\omega\end{matrix}}{\mathrm{x(yy)}:\tau}\,(\to\mathrm{E})}{\mathrm{Z}:(\omega\to\tau)\to\tau}\,(\to\mathrm{I})\begin{cases}\text{cancel}\\ \text{vacuous }\mathrm{y}:\omega\to\tau\end{cases}\qquad \cfrac{\cfrac{[\mathrm{x}:\omega\to\tau]\qquad \begin{matrix}(\omega\text{-axiom})\\ \mathrm{yy}:\omega\end{matrix}}{\mathrm{x(yy)}:\tau}\,(\to\mathrm{E})}{\mathrm{Z}:\omega\to\tau}\,(\to\mathrm{I})\begin{cases}\text{cancel}\\ \text{vacuous }\mathrm{y}:\omega\end{cases}}{\cfrac{\mathrm{ZZ}:\tau}{\mathrm{Y}:((\omega\to\tau)\to\tau).}\,(\to\mathrm{I}),\text{ cancel }\mathrm{x}:\omega\to\tau}\,(\to\mathrm{E})$$

Fig. 3.

3. Main Syntactic Results

Theorem 3.1 (*β-invariance*). If $\mathrm{M}=_\beta\mathrm{N}$, then in $\mathrm{TA}_\lambda(\wedge,\omega)$ we have for any basis $\mathscr{B}$

$$\mathscr{B}\vdash\mathrm{M}:\tau\Leftrightarrow\mathscr{B}\vdash\mathrm{N}:\tau$$

Proof. By the method of [CDV81], Sect. 2 Theorem 1, using Theorem 5.1 below and Note 5.1. Here the roles of $\wedge$ and ω in the proof will just be sketched briefly.

The Role of $\wedge$ *for* β*-Invariance.* Suppose we have a β-contraction

$$(\lambda x\cdot M)\,N \rhd_{\beta} [N/x]\,M$$

and there is a proof of $[N/x]\,M:\tau$ in which two substituted occurrences of N have two different types σ_1 and σ_2. We wish to build a proof of

$$((\lambda x\cdot M)\,N):\tau$$

We first remove substituted N's from the proof of $[N/x]\,M:\tau$, to get a deduction

$$x:\sigma_1, x:\sigma_2 \vdash M:\tau$$

We then put $(\wedge E)$ at the top of this deduction, to make a deduction

$$x:(\sigma_1 \wedge \sigma_2) \vdash M:\tau$$

Then we apply $(\to I)$ and $(\wedge I)$ and $(\to E)$ to get $((\lambda x\cdot M)\,N):\tau$.

Here are two examples of the above process in action; they are standard examples of different ways in which the Curry-style system fails to be β-invariant.

Example 3.1. Consider the β-contraction

$$(\lambda x\cdot xx)\,\mathsf{I} \rhd_{\beta} \mathsf{I}\mathsf{I}$$

In the Curry-style system TA_{λ}, one side is stratified but the other is not. In fact $(\lambda x\cdot xx)\,\mathsf{I}$ has no type because $\lambda x\cdot xx$ has none; but II has type $a\to a$ by the proof in Fig. 4

$$\dfrac{\dfrac{[x:a\to a]}{\mathsf{I}:(a\to a)\to(a\to a)}\,(\to I)\text{ cancel } x:a\to a \qquad \dfrac{[x:a]}{\mathsf{I}:a\to a}\,(\to I)\text{ cancel } x:a}{\mathsf{II}:a\to a.}\,(\to E)$$

Fig. 4.

Note that the type of II is obtained by giving the two I's two different types. In $(\lambda x\cdot xx)$ these I's are replaced by two x's, and these must both have the same type by the restriction on rule $(\to I)$; thus we cannot imitate the proof of $\mathsf{II}:a\to a$ to get a type for $\lambda x\cdot xx$ in the Curry system.

The introduction of $\wedge$ allows us to overcome this problem similarly to the example in section 2.3.1 above (see Fig. 5: Let $\sigma \equiv a\to a$ in that figure).

$$\dfrac{\dfrac{\dfrac{\dfrac{[x:(\sigma\wedge(\sigma\to\sigma))]}{x:\sigma\to\sigma}\,(\wedge E) \qquad \dfrac{[x:(\sigma\wedge(\sigma\to\sigma))]}{x:\sigma}\,(\wedge E)}{xx:\sigma}\,(\to E)}{(\lambda x\cdot xx):(\sigma\wedge(\sigma\to\sigma))\to\sigma}\,(\to I)\text{ cancel } x \qquad \dfrac{\dfrac{[x:a]}{\mathsf{I}:a\to a}\,(\to I) \qquad \dfrac{[x:\sigma]}{\mathsf{I}:\sigma\to\sigma}\,(\to I)}{\mathsf{I}:\sigma\wedge(\sigma\to\sigma)}\,(\wedge I)}{(\lambda x\cdot xx)\mathsf{I}:\sigma.}\,(\to E)$$

Fig. 5.

Example 3.2. Consider the β-contraction

$$(\lambda x\cdot \mathsf{K}x(\mathsf{S}x))\,\mathsf{I} \rhd_{\beta} \mathsf{KI}(\mathsf{SI})$$

In the Curry-style system TA_{λ}, both sides are stratified but they have different sets of types. In fact, it is not hard to see that the most general types that TA_λ can give these terms are

$$(\lambda x\cdot Kx(Sx))\,I:(a\to b)\to(a\to b),\qquad KI(SI):a\to a.$$

Thus $KI(SI)$ has a type $a\to a$ that $(\lambda x\cdot Kx(Sx))\,I$ does not. Just as in the previous example, the problem is that two I's in $KI(SI)$ with different types are replaced by x's in $Kx(Sx)$, and these x's must have the same type before $(\to I)$ can be used.

Figure 6 shows a proof of $KI(SI):a\to a$. (Let

$$\sigma\equiv((b\to c)\to b)\to((b\to c)\to c).)$$

(The proofs of types for I, K, S are omitted.)

$$\dfrac{\dfrac{K:(a\to a)\to(\sigma\to(a\to a))\quad I:a\to a}{KI:\sigma\to(a\to a)}\,(\to E)\qquad \dfrac{S:((b\to c)\to(b\to c))\to\sigma\quad I:(b\to c)\to(b\to c)}{SI:\sigma}\,(\to E)}{KI(SI):a\to a.}\,(\to E)$$

Fig. 6.

Introducing $\wedge$ overcomes the problem and gives us Fig. 7, which is a proof of $(\lambda x\cdot Kx(Sx))\,I:a\to a$ in $TA_\lambda(\wedge,\omega)$. (Let

$$\sigma\equiv((b\to c)\to b)\to((b\to c)\to c),\ \tau\equiv(b\to c)\to(b\to c).)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{K:(a\to a)\to(\sigma\to(a\to a))\quad \dfrac{x:(a\to a)\wedge\tau}{x:a\to a}\,(\wedge E)}{Kx:\sigma\to(a\to a)}\,(\to E)\qquad \dfrac{S:\tau\to\sigma\quad \dfrac{x:(a\to a)\wedge\tau}{x:\tau}\,(\wedge E)}{Sx:\sigma}\,(\to E)}{Kx(Sx):a\to a}\,(\to E)}{(\lambda x\cdot Kx(Sx)):((a\to a)\wedge\tau)\to(a\to a))}\,(\to I)\text{ cancel }x\qquad \dfrac{I:a\to a\quad I:\tau}{I:(a\to a)\wedge\tau}\,(\wedge I)}{(\lambda x\cdot Kx(Sx))I:a\to a.}\,(\to E)$$

Fig. 7.

The Role of ω for β-Invariance. Suppose we have a β-contraction

$$(\lambda x\cdot M)\,N\rhd_\beta[N/x]\,M$$

and x does not occur in M. Then $[N/x]\,M\equiv M$. Suppose there is a proof of $[N/x]\,M:\tau$, that is, of $M:\tau$. We build a proof of

$$((\lambda x\cdot M)\,N):\tau$$

thus: apply $(\to I)$ to the proof of $M:\tau$, cancelling $x:\omega$ vacuously, to get

$$(\lambda x\cdot M):\omega\to\tau$$

then apply $(\to E)$ to this statement and the ω-axiom $N:\omega$.

Theorem 3.2. (*Conservativity*). $TA_\lambda(\wedge,\omega)$ is a conservative extension of the Curry-style system $TA_{\lambda=}$. That is, if τ and the types in a basis $\mathscr{B}$ do not contain $\wedge$, ω, then

$$\mathscr{B}\vdash M:\tau\quad\text{in }TA_\lambda(\wedge,\omega)\qquad\Leftrightarrow\qquad \mathscr{B}\vdash M:\tau\quad\text{in }TA_{\lambda=}$$

Proof. For "$\Rightarrow$": [BCD83], Corollary 4.10, using Theorem 5.1 below and Note 5.1. For "$\Leftarrow$": the only rule of $TA_{\lambda=}$ that is not in the definition of $TA_\lambda(\wedge,\omega)$ is

(Eqβ), and this is admissible in $TA_\lambda(\wedge, \omega)$ by the β-invariance theorem (Theorem 3.1). □

Remark 3.1. The next two theorems will show that two sets of terms that are interesting from the point of view of their computational behaviour can be characterised exactly by the form of the types of their members. The first set is the set of all terms with normal forms (in the $\lambda\beta$- or $\lambda\beta\eta$-calculus, it does not matter, as a term has a β-normal form iff it has an $\beta\eta$-normal form, by [HiS86], Theorem 7.13). The second set is the set of all solvable terms; it will be defined after the characterisation theorem for the first set.

Theorem 3.3. (*Characterisation of terms with normal forms*). A λ-term M has a normal form iff $\mathscr{B} \vdash M:\tau$ for some basis $\mathscr{B}$ and type τ, neither of which contains ω.

Proof. [CDV81] or [BCD83], Theorem 4.13(ii), plus Theorem 5.1 and Note 5.1 below. □

Definition 3.1. (*solvable terms*). A λ-term M with no free variables is called *solvable* iff there exist m and terms $Q_1, \ldots, Q_m$ such that $MQ_1 \ldots Q_m$ β-converts to a normal form. A λ-term M with free variables $x_1, \ldots, x_n$ is called *solvable* iff $\lambda x_1 \ldots x_n \cdot M$ is solvable.

Example: The fixed-point combinator Y is solvable (although it is known to have no normal form), because

$$Y(KI) =_\beta KI(Y(KI)) =_\beta I$$

Note 3.1. The following characterisation theorem uses a relation $\sim$ between types that will be defined in Definition 5.2. Roughly speaking, $\sigma \sim \tau$ holds iff σ and τ both denote the same set in a certain fairly natural sense.

Theorem 3.4. (*Characterisation of solvable terms*). A λ-term M is solvable iff $\mathscr{B} \vdash M:\tau$ for some basis $\mathscr{B}$ and some $\tau \not\sim \omega$.

Proof. [CDV81] or [BCD83], Theorem 4.13(i), plus Theorem 5.1 and Note 5.1 below. □

4. Rule (η_1) and Completeness

4.1. Rule (η_1)

This rule is part of the definition of $TA_\lambda(\wedge, \omega)$ (Section 2.2.2). It says that if x does not occur free in M, then

$$(\lambda x \cdot Mx):\sigma \vdash M:\sigma$$

The informal intuition behind it is that every λ-term should represent a function or operator in some sense, and that if M represents an operator then $\lambda x \cdot Mx$ should represent the same operator and hence should have exactly the same types as M.

4.1.1. The Role of Rule (η_1)

Rule (η_1), or rather another rule equivalent to it, first appeared in [BCD83] and [Hin82]. It was not part of the original intuitions about $\wedge$ and ω, and was not in the early formal systems. Moreover it is not necessary to add it, if all we want is a system of types that are invariant under β-conversion; for example [CDV79] proves β-invariance without it and [Sal78] and [Pot80] do not use it either.

But around 1980 the original intuitions about $\wedge$ and ω were re-examined and developed into a precise definition of what it means for a statement $M:\tau$ to be satisfied in a given model of λ-calculus. (The *simple semantics*, see [Hin82], Sect. 2 or [BCD83]; in summary: given a closed term M, a λ-model $\mathscr{D}$ and an interpretation $\mathbb{V}$ of type-variables as subsets of $\mathscr{D}$, we say $M:\tau$ is *satisfied* by $\mathbb{V}$ in $\mathscr{D}$ iff the interpretation of M in $\mathscr{D}$ is a member of the interpretation of τ generated by $\mathbb{V}$; we say $M:\tau$ is *valid* in $\mathscr{D}$ iff it is satisfied by every $\mathbb{V}$.)

It was then found that there were statements $M:\tau$ that were valid in all λ-models but were not provable by the earlier rules for $\wedge$ and ω; that is, the early formal systems were *incomplete* with respect to the simple semantics.

The purpose of including rule (η_1) in $TA_\lambda(\wedge, \omega)$ is to make the system complete. The following examples will help to show how it does this.

Example 4.1. Using rule (η_1) we have a proof of

$$M:\omega\to\omega$$

for all λ-terms M, see Fig. 8. Informally speaking, $M:\omega\to\omega$ says that M denotes a function or operator.

$$\begin{array}{cl}
(\omega\text{-axiom}) & \\
Mx:\omega & \\
\hline
(\lambda x\cdot Mx):\omega\to\omega & (\to I)\text{ cancel vacuous } x:\omega \\
\hline
M:\omega\to\omega & (\eta_1)
\end{array}$$

Fig. 8.

Example 4.2. Let $I \equiv \lambda x\cdot x$. Then using rule (η_1) we have a proof of

$$I:\omega\to(\omega\to\omega)$$

(see Fig. 9). Informally, this statement says that I changes members of ω into members of $\omega\to\omega$. But I is the identity operator and changes nothing, so $I:\omega\to(\omega\to\omega)$ says that ω is a subset of $\omega\to\omega$, i.e. that every object in the universe is a function or operator. (This is the converse of the example in Section 2.3.2 which was provable without rule (η_1) and made the much weaker statement that every function is an object, $I:(\omega\to\omega)\to\omega$.)

In the simple semantics the statement $I:\omega\to(\omega\to\omega)$ turns out to be valid in all λ-models. Hence in a complete system this statement must be provable, so if the proof in Fig. 9 were not possible, $TA_\lambda(\wedge, \omega)$ could not claim to be complete.

It can be shown that if $TA_\lambda(\wedge, \omega)$ did not contain rule (η_1) then $I:\omega\to(\omega\to\omega)$ would not be provable.

$$\begin{array}{cl}
(\omega\text{-axiom}) & \\
xz:\omega & \\
\hline
(\lambda z\cdot xz):\omega\to\omega & (\to I)\text{ cancel vacuous } z:\omega \\
\hline
x:\omega\to\omega & (\eta_1) \\
\hline
(\lambda x\cdot x):\omega\to(\omega\to\omega) & (\to I)\text{ cancel vacuous } x:\omega
\end{array}$$

Fig. 9.

Example 4.3. Using rule (η_1) we have a proof of

$$\mathsf{I}:((a\to b)\wedge(a\to c))\to(a\to(b\wedge c))$$

(see Fig. 10). Informally, this statement says that if a, b, c denote sets, then the set $(a\to b)\wedge(a\to c)$ is a subset of $a\to(b\wedge c)$. This can easily be seen to agree with the intended meaning of "$\to$" and "$\wedge$", see (A 7) in Definition 5.1. Moreover, the above statement can be shown to be valid in all λ-models. Hence if the proof in Fig. 10 were not possible, $TA_\lambda(\wedge,\omega)$ could not claim to be complete.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x:(a\to b)\wedge(a\to c)]}{x:a\to b}\,(\wedge E)\qquad [y:a]}{xy:b}\,(\to E)\qquad \dfrac{\dfrac{[x:(a\to b)\wedge(a\to c)]}{x:a\to c}\,(\wedge E)\qquad [y:a]}{xy:c}\,(\to E)}{xy:b\wedge c}\,(\wedge I)}{(\lambda y\cdot xy):a\to(b\wedge c)}\,(\to I)\ \text{cancel } y:a}{x:a\to(b\wedge c)}\,(\eta_1)}{(\lambda x\cdot x):((a\to b)\wedge(a\to c))\to(a\to(b\wedge c))}\,(\to I)\ \text{cancel } x$$

Fig. 10.

Theorem 4.1. (*Completeness*). For any term M containing no free variables, the statement $M:\tau$ is provable in $TA_\lambda(\wedge,\omega)$ iff $M:\tau$ is valid in every λ-model (in the sense of the simple semantics).

Proof. [BCD83], Theorem 3.10, or [Hin82], Sect. 3, plus Theorem 5.1 and Note 5.1 below. □

5. The $\leqslant$-Relation and Rule $(\leqslant)$

In most formulations of $TA_\lambda(\wedge,\omega)$ in the literature, rule (η_1) is replaced by another rule called $(\leqslant)$ which is easier to modify to produce variant systems. But rule $(\leqslant)$ is more complicated to state, which is why it was not used in Section 2.2.2. To state it we must first define a formal relation $\sigma\leqslant\tau$ between types by means of axioms and rules. This relation was introduced in [BCD83], Definition 2.3 to imitate the informal subset relation.

Definition 5.1. (*The $\leqslant$-relation*).

The relation $\sigma\leqslant\tau$ is defined inductively by the following axiom-schemes and rules.

Axiom-schemes:

(A1) $\sigma\leqslant\sigma$,

(A2) $\sigma\leqslant\omega$,

(A3) $\omega\leqslant\omega\to\omega$,

(A4) $\sigma\leqslant\sigma\wedge\sigma$,

(A5) $\sigma\wedge\tau\leqslant\sigma$,

(A6) $\sigma \wedge \tau \leqslant \tau$,
(A7) $(\sigma \to \tau_1) \wedge (\sigma \to \tau_2) \leqslant \sigma \to (\tau_1 \wedge \tau_2)$.

Rules:
(R1) $\sigma \leqslant \sigma', \tau \leqslant \tau' \;\Rightarrow\; \sigma \wedge \tau \leqslant \sigma' \wedge \tau'$,
(R2) $\sigma \leqslant \sigma', \tau \leqslant \tau' \;\Rightarrow\; \sigma' \to \tau \leqslant \sigma \to \tau'$ {*not* $\sigma \to \tau \leqslant \sigma' \to \tau'$},
(R3) $\tau_1 \leqslant \tau_2, \tau_2 \leqslant \tau_3 \;\Rightarrow\; \tau_1 \leqslant \tau_3$.

Definition 5.2. $\sigma \sim \tau \Leftrightarrow \sigma \leqslant \tau$ and $\tau \leqslant \sigma$.

Informal justification of (A1)–(R3). Think of types as sets and "$\sigma \leqslant \tau$" as meaning that σ is a subset of τ, and think of a universe in which every object is an operator or function. Then all of (A1)–(A7) and (R1)–(R3) are valid. The only ones whose validity is not immediately obvious are (A3), (A7) and (R2). Here they are in detail.

For (A3): "$\omega \leqslant \omega \to \omega$" says that every object is an operator; compare the motivation for rule (η_1) in Section 4.1.

For (A7): If f is an operator in $(\sigma \to \tau_1) \wedge (\sigma \to \tau_2)$, then for each input x in σ, the output fx is in τ_1 (if there is an output at all), and fx is also in τ_2. Hence fx is in $\tau_1 \wedge \tau_2$ for each x in σ which produces an output; thus f is in $\sigma \to (\tau_1 \wedge \tau_2)$.

For (R2): Let $\sigma \subseteq \sigma'$ and $\tau \subseteq \tau'$, and let f be in $\sigma' \to \tau$. For each x in σ, x is also in σ', so fx must be in τ (if fx exists). But $\tau \subseteq \tau'$, so fx is in τ'. Hence f is in $\sigma \to \tau'$.

Lemma 5.1. Decidability. The $\leqslant$-relation is recursive; i.e. there is an algorithm which tests, for each pair of types σ, τ, whether $\sigma \leqslant \tau$ or not.

Proof. One algorithm is given in [Hin82], Sect. 4, Lemma 3(v) and the Corollary to Lemma 5; it starts by translating σ and τ into a more limited type-language and then decides the relation $\leqslant$ in this restricted language. (This language is essentially the one in which intersection types were first introduced, see [CDV81] for example.

Definition 5.3. Rule $(\leqslant)$ states:

$$\frac{M:\sigma \quad \sigma \leqslant \tau}{\mathrm{M}:\tau} \qquad (\leqslant)$$

Theorem 5.1. In $\mathrm{TA}_\lambda(\wedge, \omega)$, rule (η_1) can be replaced by rule $(\leqslant)$ without affecting the set of provable statements or the relation $\mathscr{B} \vdash \mathrm{M}:\tau$.

Proof. Let "$\vdash_{\eta_1}$" denote provability in $\mathrm{TA}_\lambda(\wedge, \omega)$ and "$\vdash_{\leqslant}$" provability in the system resulting from replacing rule (η_1) by rule $(\leqslant)$. We must show that

$$\mathscr{B} \vdash_{\eta_1} \mathrm{M}:\tau \;\Leftrightarrow\; \mathscr{B} \vdash_{\leqslant} \mathrm{M}:\tau \qquad (3)$$

Proof of (3) "$\Rightarrow$". By the subject-reduction theorem for $\beta\eta$-reduction ([BCD83] Remark 2.10) or the η-lemma in [Hin82], Sect. 5 p. 225, if $\mathscr{B} \vdash_{\leqslant} (\lambda \mathrm{x} \cdot \mathrm{Mx}):\sigma$ and x is not free in M then $\mathscr{B} \vdash_{\leqslant} \mathrm{M}:\sigma$. Thus rule (η_1) is admissible in the system of $\vdash_{\leqslant}$.

Proof of (3) "$\Leftarrow$", It is enough to show that if $\sigma \leqslant \tau$, then

$$\mathscr{B} \vdash_{\eta_1} \mathrm{M}:\sigma \;\Rightarrow\; \mathscr{B} \vdash_{\eta_1} \mathrm{M}:\tau \qquad (4)$$

This is done by induction on the clauses of Definition 5.1. As examples we shall do (A7) and (R2) here; the other clauses are easier, in particular (A3) is like Example 4.1.

For (A7): Let $\mathscr{B}\vdash_{\eta_1} \mathrm{M}:(\sigma\to\tau_1)\wedge(\sigma\to\tau_2)$; we want $\mathscr{B}\vdash_{\eta_1}\mathrm{M}:\sigma\to(\tau_1\wedge\tau_2)$. This is obtained by the deduction in Fig. 11 (similar to Fig. 10). Note that y is any term-variable not occurring in M or $\mathscr{B}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{M}:(\sigma\to\tau_1)\wedge(\sigma\to\tau_2)}{\mathrm{M}:\sigma\to\tau_1}(\wedge\,\mathrm{E})\qquad [\mathrm{y}:\sigma]}{\mathrm{My}:\tau_1}(\to\mathrm{E})\qquad \dfrac{\dfrac{\mathrm{M}:(\sigma\to\tau_1)\wedge(\sigma\to\tau_2)}{\mathrm{M}:\sigma\to\tau_2}(\wedge\,\mathrm{E})\qquad [\mathrm{y}:\sigma]}{\mathrm{My}:\tau_2}(\to\mathrm{E})}{\mathrm{My}:\tau_1\wedge\tau_2}(\wedge\,\mathrm{I})}{(\lambda\mathrm{y}\cdot\mathrm{My}):\sigma\to(\tau_1\wedge\tau_2)}(\to\mathrm{I})\ \text{cancel } \mathrm{y}:\sigma}{\mathrm{M}:\sigma\to(\tau_1\wedge\tau_2)}(\eta_1)$$

Fig. 11.

For (R2): Let $\mathscr{B}\vdash_{\eta_1}\mathrm{M}:\sigma'\to\tau$ and $\sigma\leqslant\sigma'$ and $\tau\leqslant\tau'$; we want

$$\mathscr{B}\vdash_{\eta_1}\mathrm{M}:\sigma\to\tau'$$

Choose a term-variable y not in M or $\mathscr{B}$. Now trivially

$$\mathscr{B},\mathrm{y}:\sigma\vdash_{\eta_1}\mathrm{y}:\sigma$$

and $\sigma\leqslant\sigma'$ so the induction hypothesis can be applied to give us

$$\mathscr{B},\mathrm{y}:\sigma\vdash_{\eta_1}\mathrm{y}:\sigma'$$

From this and rule $(\to\mathrm{E})$ it follows that

$$\mathscr{B},\mathrm{y}:\sigma\vdash_{\eta_1}\mathrm{My}:\tau$$

and then the induction hypothesis applied to the relationship $\tau\leqslant\tau'$ gives us

$$\mathscr{B},\mathrm{y}:\sigma\vdash_{\eta_1}\mathrm{My}:\tau'$$

Hence by rule $(\to\mathrm{I})$,

$$\mathscr{B}\vdash_{\eta_1}(\lambda\mathrm{y}\cdot\mathrm{My}):\sigma\to\tau'$$

and then rule (η_1) gives $\mathrm{M}:\sigma\to\tau'$ as required. □

Note 5.1. The equivalence of rules (η_1) and $(\leqslant)$ was known to the Turin group as far back as 1980, I believe. In fact, logically speaking, Theorem 5.1 should be at the start of the present paper, as the proofs of the other main theorems (β-invariance and conservativity etc.) are all written in the literature for the system based on $(\leqslant)$ not (η_1).

Note 5.2. (The $\leqslant$ and subset relations). The relation $\leqslant$ was originally introduced with the intention of formalising some key properties of the informal subset relation. In [BCD83], corollary 3.9, this intention is shown to succeed in the sense that the following holds, using the definition of the simple semantics in [Hin82], Sect. 2 or [BCD83]:

$\sigma\leqslant\tau\Leftrightarrow$ for every λ-model $\mathscr{D}$ and every interpretation of the type-variables in σ and τ as subsets of $\mathscr{D}$, the corresponding interpretation of σ is a subset of that of τ. (5)

Informally speaking, an alternative way of saying that σ is a subset of τ is to say that the identity operator maps σ into τ. Formally, the following can be proved:

$$\sigma \leqslant \tau \Leftrightarrow \text{the statement } \mathsf{I}:\sigma \to \tau \text{ is provable in } \mathrm{TA}_\lambda(\wedge, \omega). \qquad (6)$$

(Proof of (6): "$\Rightarrow$" is easy. For "$\Leftarrow$"; if $\mathsf{I}:\sigma \to \tau$ is provable then it is valid in every λ-model by the completeness theorem (theorem 4.1), and in the simple semantics this is easily seen to imply that the interpretation of σ is a subset of that of τ; hence $\sigma \leqslant \tau$ by (i).)

Note 5.3. (*Rule* ($\leqslant$) *and choice of semantics*). Rule ($\leqslant$) is normally used in the literature in preference to (η_1), although it takes longer to state. This is because the simple semantics is not the only interesting way of interpreting types in a λ-model and for other interpretations rule (η_1) is not valid. Of course if rule (η_1) is not valid, rule ($\leqslant$) will not be valid either, as the two rules are equivalent. But for most of the different interpretations considered so far in the literature, the axioms in the definition of $\leqslant$ can be changed to suit that particular interpretation and make the rule valid. Thus rule ($\leqslant$) has a flexibility that (η_1) does not have.

(For some other definitions of semantics and corresponding modifications of ($\leqslant$) see [CDH83], [CDZ87], [DeM84] and [DeM86]. Completeness theorems are proved in [CDH83] and [CDZ87].)

6. Conclusion

For more on types with intersection see the list of references below. This gives all publications on the topic to the end of 1991 (see also [Bar89], [BoR89] and [Rey91] in addition to those already cited), and some more recent papers (e.g. [Bak92]).

Acknowledgements

I wish to record thanks to the organisers and participants of the International Symposium on Mathematical Logic and its Applications, Nagoya 1988, the Workshop on Hardware Verification, Banff 1989, and the British Colloquium on Theoretical Computer Science, 1989, for sponsoring and suffering the talks from which this paper has been developed. Also to Franco Barbanera for some items in the reference list, and to the referees for some useful improvements to the main text.

References

[AlB91] Alessi, F. and Barbanera, F.: Strong Conjunction and intersection Types. In: *Mathematical Foundations of Computer Science* 1991, A. Tarlecki (ed.), *Lecture Notes in Computer Science* 520, Springer-Verlag, pp. 64–73 (1991).

[Bak92] van Bakel, S: Complete Restrictions of the Intersection Type Discipline, *Theoretical Computer Science* (to appear).

[BaD91] Barbanera, F. and Dezani, M.: Intersection and Union Types. In: *Theoretical Aspects of Computer Software, International Conference TACS* '91, *Sendai, Japan, September* 1991, *Proceedings*, T. Ito and A. R. Meyer (eds.), *Lecture Notes in Computer Science* 526, Springer-Verlag, pp. 651–674 (1991).

[Bar89] Barbanera, F.: Combining Term Rewriting and Type Assignment Systems, *Proc. Third Italian Conf. on Theoretical Computer Science, Mantova 1989*, A. Bertoni *et al.* (eds) World Scientific Press, pp. 71–84 (1989). (To appear in *Foundations of Computer Science.*)

[BCD83] Barendregt, H. P., Coppo, M. and Dezani, M.: A Filter Lambda Model and the Completeness of Type Assignment. *Journal of Symbolic Logic*, **48**, 931–940 (1983).

[BoR89] Bosio, E. and Ronchi della Rocca, S.: Type Synthesis for Intersection Type Discipline. In: *Proc. Third Italian Conference on Theoretical Computer Science Mantova 1989*, A. Bertoni *et al.* (eds.), World Scientific Press, pp. 109–122, (1989).

[Chu40] Church, A.: A Formalisation of the Simple Theory of Types. *Journal of Symbolic Logic*, **5**, 56–68 (1940).

[Cop80] Coppo, M.: An Extended Polymorphic Type System for Applicative Languages. In: Mathematical Foundations of Computer Science 1980, Proceedings of the 9th Symposium, Held in Rydzyna, Poland, P. Dembinski (ed.), *Lecture Notes in Computer Science* 88, Springer-Verlag, pp. 194–204 (1980).

[CoD78] Coppo, M. and Dezani, M.: A New Type-Assignment for Lambda Terms. *Archiv für Math. Logik*, **19**, 139–156 (1978).

[CoD80] Coppo, M. and Dezani, M.: An Extension of the Basic Functionality Theory for the λ-Calculus. *Notre Dame Journal of Formal Logic*, **21**, 685–693 (1980).

[CDH83] Coppo, M., Dezani, M., Honsell, F. and Longo, G.: Extended Type Structures and Filter Lambda Models. In: *Logic Colloquium '82*, G. Lolli *et al.*, (eds), North-Holland, pp. 241–262 (1983).

[CDS79] Coppo, M., Dezani, M. and Sallé, P.: Functional Characterization of Some Semantic Equalities Inside λ-Calculus. In: Automata, Languages and programming, Sixth Colloquium, Graz, July 1979, H. A. Maurer (ed.), *Lecture Notes in Computer Science* 71, Springer-Verlag, pp. 133–146 (1979).

[CDV80] Coppo, M., Dezani, M. and Venneri, B. Principal Type Schemes and λ-Calculus Semantics. In: *To H. B. Curry*, J. R. Hindley and J. P. Seldin (eds), Academic Press, pp. 535–560 (1980).

[CDV81] Coppo, M., Dezani, M. and Venneri, B.: Functional Characters of Solvable Terms. *Zeit. Math. Logik* **27**, 45–58 (1981).

[CDZ87] Coppo, M., Dezani, M. and Zacchi, M.: Type-Theories, Normal Forms, and D_∞-Lambda Models. *Information and Computation*, **72**, 85–116 (1987).

[Cur69] Curry, H. B.: Modified Basic Functionality in Combinatory Logic. *Dialectica* **23**, 83–92 (1969).

[DeH92] Dezani, M. and Hindley, J. R.: Intersection Types for Combinatory Logic, *Theoretical Computer Science* (to appear).

[DeM84] Dezani, M. and Margaria, I.: F-Semantics for Intersection Type Discipline. In: Semantics of Data Types, International Symposium, Sophia-Antipolis, France, June 1984, Proceedings, G. Kahn, D. MacQueen and G. Plotkin (eds.), *Lecture Notes in Computer Science*, 173, Springer-Verlag, 279–300 (1984).

[DeM86] Dezani, M. and Margaria, I.: A Characterization of F-complete Type Assignments. *Theoretical Computer Science*, **45**, 121–157 (1986).

[Hay91] Hayashi, S.: Singleton, union and intersection types for program extraction. In: Theoretical Aspects of Computer Software, international Conference TACS '91, Sendai, Japan, September 1991, Proceedings, T. Ito and A. R. Meyer (eds.), *Lecture Notes in Computer Science* 526, Springer-Verlag, pp. 701–730 (1991).

[Hin82] Hindley, J. R.: The Simple Semantics for Coppo–Dezani–Sallé Types. In: International Symposium on programming, 5th Colloquium, Turin, April 1982, M. Dezani and U. Montanari (eds.), *Lecture Notes in Computer Science* 137, Springer-Verlag, 212–226 (1982).

[HiS86] Hindley, J. R. and Seldin, J. P.: *Introduction to Combinators and λ-Calculus*, Cambridge University Press, 1986.

[HoR91] Honsell, F. and Ronchi della Rocca, S.: Reasoning About Interpretation in Qualitative Lambda Models. In: *IFIP TC2 Working Conf. on Programming Concepts and Methods 1990*, North-Holland, pp. 492–508, (1991).

[Lop85a] Lopez-Escobar, E. K. G.: Proof Functional Connectives. In: *Lecture Notes in Mathematics*. 1130, Springer-Verlag, 208–221 (1985).

[Lop85b] Lopez-Escobar, E. K. G. On Intuitionistic Sentential Connectives, *Rev. Colombiana de Math.*, **19**, 117–130 (1985).

[MaZ91] Margaria, I., Zacchi, M.: Principal Typing in a Polymorphic-intersection Type Discipline. MS, Dipartimento di Informatica, Corso Svizzera 185, Turin, Italy (1991).

[Min89] Mints, G.: The Completeness of Provable Realizability. *Notre Dame Journal of Formal Logic*, **30**, 420–441 (1989).

[Pie91] Pierce, B. C.: Programming with intersection types, union types and polymorphism. Technical Report CMU-CS-91-106, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA (1991).

[Pot80] Pottinger, G.: A Type Assignment for the Strongly Normalizable λ-Terms. In: *To H. B. Curry*, J. R. Hindley and J. P. Seldin (eds), Academic Press, pp. 561–577, (1980).

[Rey88] Reynolds, J. C.: *Preliminary Design of the Programming Language Forsythe.* Report CMU-CS-88-159. Computer Science Department, Carnegie-Mellon University, Pittsburgh, USA. (1988).

[Rey89] Reynolds, J. C.: Syntactic Control of Interference, Part 2. In: Automata, Languages and Programming, 16th International Colloquium, Stresa, Italy, July 1989, Proceedings, G. Ausiello, M. Dezani and S. Ronchi della Rocca (eds.), *Lecture Notes in Computer Science* 372, Springer-Verlag, pp. 704–722 (1989).

[Rey91] Reynolds, J. C.: The Coherence of Languages with Intersection Types. In: Theoretical Aspects of Computer Software, International Conference TACS '91, Sendai, Japan, September 1991, Proceedings, T. Ito and A. R. Meyer (eds.), *Lecture Notes in Computer Science* 526, Springer-Verlag, pp. 675–700 (1991).

[Ron82] Ronchi Della Rocca, S.: Characterization Theorems for a Filter Lambda Model. *Information and Control*, **54**, 201–216 (1982).

[RoV84] Ronchi Della Rocca, S. and Venneri, B.: Principal Type-Schemes for an Extended Type Theory. *Theoretical Computer Science*, **28**, 151–169 (1984).

[Sal78] Sallé, P.: Une Extension de la Théorie des Types. In: Automata, Languages and Programming, Fifth Colloquium, Udine, July 1978, G. Ausiello and C. Böhm (eds.), *Lecture Notes in Computer Science* 62, Springer-Verlag, pp. 398–410 (1978).

Received December 1990
Accepted in revised form 1991 by J. V. Tucker