History of Lambda-calculus and Combinatory Logic

Felice Cardone * J. Roger Hindley †

2006,
from
Swansea University Mathematics Department Research Report
No. MRRS-05-06. ‡

Contents

1 Introduction **1**

2 Pre-history **2**

3 1920s: Birth of Combinatory Logic **3**

4 1930s: Birth of λ and Youth of CL **6**
4.1 Early λ-calculus 6
4.2 CL in the 1930s 9

5 1940s and 1950s: Consolidation **12**
5.1 Simple type theory 12
5.2 Abstract reduction theory 13
5.3 Reductions in CL and λ 13
5.4 Illative systems 14

6 Programming languages **16**
6.1 John McCarthy and LISP 16
6.2 Peter Landin 16
6.3 Corrado Böhm: λ-calculus as a programming language 17

7 Syntactical developments **18**
7.1 Contributions from the programming side 19
7.2 Theory of reductions 21

8 Types **23**
8.1 The general development of type theories 23
8.1.1 Types as grammatical categories 24
8.1.2 Types as sets 25
8.1.3 Types as objects 26
8.1.4 Types as propositions 30
8.2 Early normalization proofs 35
8.3 Higher-order type theories 37

*Università di Milano-Bicocca, Dipartimento di Informatica, Sistemistica e Comunicazione, Milano, Italy. E-mail: cardone@disco.unimib.it

†Mathematics Department, Swansea University, Swansea SA2 8PP, U.K. E-mail: j.r.hindley@swansea.ac.uk

‡To be published in Volume 5 of **Handbook of the History of Logic,** Editors Dov M. Gabbay and John Woods, Elsevier Co.

8.4 Intersection types and recursive types 40
8.5 Algorithms for simple types 42

9 Models for λ **44**
9.1 Scott's D_∞ model 44
9.2 Computational and denotational properties 45
9.3 Other models 48

10 Domain theory **49**
10.1 Classical domain theory 49
10.2 Effective domains and Synthetic Domain Theory 53
10.3 Game semantics 54

1 Introduction

The formal systems that are nowadays called λ-calculus and combinatory logic were both invented in the 1920s, and their aim was to describe the most basic properties of function-abstraction, application and substitution in a very general setting. In λ-calculus the concept of abstraction was taken as primitive, but in combinatory logic it was defined in terms of certain primitive operators called *basic combinators*.

The present article will sketch the history of these two topics through the twentieth century.

We shall assume the reader is familiar with at least one of the many versions of these systems in the current literature. A few key technical details will be given as footnotes, however.[1] Often "combinatory logic" will be abbreviated to "CL" and "λ-calculus" to "λ". We shall distinguish between "pure" versions of λ or CL (theories of conversion or reduction with nothing more) and "applied" versions (containing extra concepts such as logical constants, types or numbers).

To understand the early history it is worth remembering that the situation in logic and the foundations of mathematics was much more fluid in the early 1900s than it is today; Russell's paradox was relatively recent, Gödel's theorems were not yet known, and a significant strand of work in logic was the building of systems intended to be consistent foundations for the whole of mathematical analysis. Some of these were based on a concept of set, others on one of function, and there was no general consensus as to which basis was better. In this context λ and CL were originally developed, not as autonomous systems but as parts of more elaborate foundational systems based on a concept of function.

Today, λ and CL are used extensively in higher-order logic and computing. Rather like the chassis of a bus, which supports the vehicle but is unseen by its users, versions of λ or CL underpin several important logical systems and programming languages. Further, λ and CL gain most of their purpose at second hand from such systems, just as an isolated chassis has little purpose in itself. Therefore, to give a balanced picture the present article should really include the whole history of function-based higher-order logic. However, it would then be too diffuse, so we shall take a more restricted approach. The reader should always keep the wider context in mind, however.

Seen in outline, the history of λ and CL splits into three main periods: first, several years of intensive and very fruitful study in the 1920s and '30s; next, a middle period of nearly 30 years of relative quiet; then in the late 1960s an upsurge of activity stimulated by developments in higher-order function theory, by connections with programming languages, and by new technical discoveries. The fruits of the first period included the first-ever proof that predicate logic is undecidable. The results of the second attracted very little non-specialist interest, but included completeness, cut-elimination and standardization theorems (for example) that found many uses later. The achievements of the third, from the 1960s onward, included constructions and analyses of models, development of polymorphic type systems, deep analyses of the reduction process, and many others probably well known to the reader. The high level of activity of this period continues today.

The present article will describe earlier work in chronological order, but will classify later developments by topic, insofar as overlaps allow. Each later topic will be discussed in some breadth, to show something of the contexts in which λ and

[1] A short introduction to λ-calculus is included in [Seldin, 2007] in the present volume. Others can be found in many textbooks on computer science. There are longer introductions in [Hankin, 1994], [Hindley and Seldin, 1986], [Stenlund, 1972]; also in [Krivine, 1990] (in French and English), [Takahashi, 1991] (in Japanese), and [Wolfengagen, 2004] (in Russian). A deeper account is in [Barendregt, 1981]. For combinatory logic there are introductions in [Hindley and Seldin, 1986, Ch.2], [Stenlund, 1972], and [Barendregt, 1981, Ch.7]; more detailed accounts are in [Curry and Feys, 1958, Chs.5–9] and [Curry *et al.*, 1972].

CL are being used. However, due to lack of space and the richness of the field we shall not be able to be as comprehensive as we would like, and some important subtopics will unfortunately have to be omitted; for this we ask the reader's forgiveness. Although we shall try to keep a balance, our selection will inevitably reflect our own experience. For example Curry's work will be given more space than Church's simply because we know it better, not because we think it more important.

A large bibliography will be included for the reader's convenience in tracing sources. When detailed evidence of a source is needed, the relevant precise section or pages will be given. Sometimes the date we shall give for an idea will significantly precede its date of publication; this will be based on evidence in the source paper such as the date of the manuscript. By the way, mention of a date and an author for an idea should not be interpreted as a claim of priority for that author; important ideas may be invented independently almost simultaneously, and key papers may be circulated informally for some years before being published, so questions of priority, although very interesting, are beyond our powers to decide.

Acknowledgements

The present account owes its beginning to a suggestion by Dirk van Dalen. For useful information and advice during its preparation the authors are very grateful to many helpful correspondents, in particular John Addison, Peter Andrews, Corrado Böhm, Martin Bunder, Pierre-Louis Curien, Steven Givant, Ivor Grattan-Guinness, Peter Hancock, Gérard Huet, Reinhard Kahle, Alexander Kuzichev, William Lawvere, Bruce Lercher, Giuseppe Longo, Per Martin-Löf, Eugenio Moggi, John Reynolds, Jonathan Seldin, John Shepherdson, William Tait, Christian Thiel, Anne Troelstra, Pawel Urzyczyn and Philip Wadler.

This account has also been much helped by material from other historical articles, especially [Barendregt, 1997], [Crossley, 1975], [Curry and Feys, 1958, §§0D, 1S, 2S, 3S, etc.], [Gandy, 1988], [Heijenoort, 1967], [Hodges, 1983], [Kalman, 1983], [Kamareddine *et al.*, 2002], [Kleene, 1981], [Laan, 1997], [Manzano, 1997], [Nederpelt and Geuvers, 1994], [Rosser, 1984], [Seldin, 1980a; Seldin, 1980b], and [Sieg, 1997].

However, any errors in what follows are our own responsibility.

On the financial side, we express our gratitude to the Dipartimento di Informatica of the University of Turin, and the British Council, for their very generous support and facilities which made a crucial consultation visit possible in May 2000.

2 Pre-history

Notations for function-abstraction and substitution go back at least as far as 1889. In Giuseppe Peano's book on axioms for arithmetic [Peano, 1889, §VI], for any term α containing a variable x, the function of x determined by α was called $\alpha[x]$, and for $\phi = \alpha[x]$ the equation $\phi\, x' = \alpha[x]x'$ was given, and its right-hand side was explicitly stated to mean the result of substituting x' for x in α. Later, Peano used other function-abstraction notations instead of $\alpha[x]$; these included $\alpha\,\overline{x}$ and $\alpha|x$, in [Peano, 1958, p.277] and [Peano, 1895, Vol.3 §11] respectively.

In 1891 Gottlob Frege discussed the general concept of function and introduced the notion of a function as a graph (*Wertverlauf* in [Frege, 1891]). Two years later, notations for abstraction and application appeared in his [Frege, 1893, Vol.1 §9], where the graph of a function $\Phi(\ )$ was denoted by $\dot{\varepsilon}\,\Phi(\varepsilon)$, and the result of applying the function to an argument Δ was denoted by $\Delta \cap \dot{\varepsilon}\,\Phi(\varepsilon)$. In the same work Frege also proved the equation $a \cap \dot{\varepsilon}\, f(\varepsilon) = f(a)$, [Frege, 1893, Vol.1 p.73, §54].

Notations for class-abstraction too appeared at around the same time. The set of all x satisfying condition α was denoted by $[x\,\varepsilon]\,\alpha$ in [Peano, 1889, §V], and by $\overline{x\,\varepsilon}\,\alpha$ in Peano's work from 1890 onward, see [Peano, 1958, Vol.2 p.110]. The latter notation was also used by his colleague Burali-Forti [Burali-Forti, 1894, p.71 §3]. Another class-abstraction notation, $\hat{z}(\phi\, z)$, appeared in [Russell and Whitehead, 1913, Ch.1, p.25]. The latter two authors also used $\varphi\,\hat{x}$ for the unary propositional function determined by $\varphi\, x$.

A function-abstraction notation was used by Bertrand Russell in 1903–5 in unpublished notes, see [Klement, 2003].

However, none of these authors gave formal definitions of substitution and conversion, which are characteristic of CL and λ proper. These did not appear until the 1920s, and, although λ might seem to express the concepts involved more directly than CL, it was actually CL that was invented first.

3 1920s: Birth of Combinatory Logic

Combinatory logic was invented by Moses Ilyich Schönfinkel. Schönfinkel was born in 1887 or '89 in Dniepropetrovsk in the Ukraine, and studied under the Odessa mathematician Samuel Shatunovsky – a brilliant pupil, according to [Yanovskaya, 1948, p. 31]. From 1914 to 1924 he belonged to what was probably the top mathematics research group of the time, that led by David Hilbert in Göttingen, Germany. He introduced combinators to this group in a talk on 7 December 1920, which was published in [Schönfinkel, 1924].

Schönfinkel's main aim was to eliminate bound variables, which he saw as merely auxiliary syntactic concepts, see [Schönfinkel, 1924, end of §1]. To do this, he introduced five operators Z, T, I, C and S which would nowadays be called *basic combinators* **B**, **C**, **I**, **K** and **S** respectively.[2]

Schönfinkel proved that **K** and **S** are adequate to define the other three basic combinators, and stated a form of combinatory completeness result (with a sketch proof): namely that from **K** and **S**, together with a logical operator equivalent to a universally quantified Sheffer stroke or *Nand*, one can generate all formulas of predicate logic without the use of bound variables. But he treated equality and logical equivalence only informally, as was common practice in mathematics at that time; he did not introduce formal conversion rules. Nor was an explicit abstraction algorithm stated in his paper, although from what was written it seems extremely likely that he knew one.

Along the way, Schönfinkel pointed out that multi-variable applications such as $F(x, y)$ could be replaced by successive single applications $(f(x))(y)$, where f was a function whose output-value $f(x)$ was also a function, see [Schönfinkel, 1924, §2]. This replacement-process is now known in the computer-science community as "currying", from its use in the work of Haskell Curry; although Curry many times attributed it to Schönfinkel, for example in [Curry, 1930, p.512] and [Curry and Feys, 1958, pp. 8, 11, 30, 106], and it was used even earlier by Frege, [Frege, 1893, Vol.1 §4].

After his 1924 paper, Schönfinkel published nothing more on combinators. In fact only one other paper bears his name: [Bernays and Schönfinkel, 1928], on certain special cases of the *Entscheidungsproblem*. Both papers were prepared for

[2] In today's notation (following [Curry and Feys, 1958, Ch.5]) their axiom-schemes are $\mathbf{B}XYZ = X(YZ)$, $\mathbf{C}XYZ = XZY$, $\mathbf{I}X = X$, $\mathbf{K}XY = X$, $\mathbf{S}XYZ = XZ(YZ)$, and a *combinator* is any applicative combination of basic combinators. A set $\mathcal{S}$ of combinators is called *combinatorially complete* when, for every sequence $x_1, ..., x_n$ of variables and every combination X of all or some of these (possibly with some occurring more than once), there exists a combination A of members of $\mathcal{S}$ such that $Ax_1...x_n = X$. An *abstraction algorithm* is any algorithm for constructing a suitable A from X. In what follows, the notation "$[x_1, ..., x_n].X$" will denote any suitable A.

publication largely by helpful colleagues: the earlier one by Heinrich Behmann and the later by Paul Bernays. By 1927, he was said to be mentally ill and in a sanatorium [Curry, 1927, p.3], [Kline, 1951, p.47]. Alexander Kuzichev in Moscow relates that Schönfinkel's later years were spent in Moscow in hardship and poverty, helped by a few friends, and he died in a hospital there in 1942 following some years of illness. After his death, wartime conditions forced his neighbours to burn his manuscripts for heating.[3]

However, in the meantime the combinator idea had appeared again, though in a very modified form. In 1925, John von Neumann published his doctoral thesis on axiomatizing set theory, [Neumann, 1925]. Despite its being a theory of sets, his formal system was function-based not set-based, and one group of his axioms postulated certain combinator-like operators which gave combinatory completeness for all expressions built from variables and constants by pairing and function-application, [Neumann, 1925, p.225].[4]

(We do not know whether von Neumann's idea came from Schönfinkel's. Von Neumann visited Göttingen well before 1925 and collaborated with the logic group there, and it is hard to imagine that he did not learn of Schönfinkel's work there. But the form in which the concept appeared in [Neumann, 1925] was very modified, and his [Neumann, 1925] and [Neumann, 1928] did not mention Schönfinkel.)

The von Neumann axioms eventually evolved, with many changes, including from a function base to a sets-and-classes base, into the nowadays well known Von-Neumann-Bernays-Gödel system NBG. In that system the analogue of von Neumann's combinator axioms was a finite list of class-existence axioms, and the analogue of his combinatory completeness theorem was a general class-existence theorem. That system gave a finite axiomatization of set theory, in contrast to the Zermelo-Fraenkel system known nowadays as ZF. It did not, however, eliminate bound variables completely from the logic; unlike Schönfinkel, von Neumann did not have that as an aim.

The next major step in CL was independent of both Schönfinkel and von Neumann: the combinator concept was re-invented in the U.S.A. by Haskell Curry.

Curry was born in 1900. His parents ran a school of public speaking near Boston, U.S.A. (now called Curry College), and from them he derived an interest in linguistics which coloured much of his later work. He began his student career at Harvard University intending to work in medicine but quickly changed to mathematics, and a strong underlying inclination towards foundational questions drew him eventually to logic.[5]

In propositional logic the rule of substitution is significantly more complex than *modus ponens* and the other deduction-rules, and at the time Curry began his studies the substitution operation was largely unanalyzed.[6] In particular, an explicit

[3]Little more is known about him. We are indebted mainly to [Thiel, 1995], [Yanovskaya, 1948, pp.31–34], [Kline, 1951], and correspondence from A. S. Kuzichev, 2005. His date of birth was 4th Sept. 1889 according to [Thiel, 1995], but 1887 according to Kuzichev.

[4]The relevant axioms appeared in [Neumann, 1925, §3, axiom-group II], see comments in [Curry and Feys, 1958, pp.10–11]. We show them here in a notation chosen to emphasize their similarity to CL. Von Neumann used two atomic predicates, "x is a function" and "x is an argument" (not mutually exclusive), and two term-building operations: pairing, which we call "$\langle\,,\,\rangle$", and application "$(\ )$". (For pairing, he postulated two projection-functions; we omit them here.)

$(\exists\, fn\ \mathsf{I})(\forall\, args\ x)$	$(\mathsf{I}x) = x$;
$(\forall\, args\ x)(\exists\, fn\ \mathsf{K}_x)(\forall\, args\ y)$	$(\mathsf{K}_x y) = x$;
$(\exists\, fn\ \mathbf{App})(\forall\, args\ f, x)$	f *is a* $fn\ \supset\ (\mathbf{App}\,\langle f, x\rangle) = (fx)$;
$(\forall\, fns\ f, g)(\exists\, fns\ \mathsf{S}_{f,g}, \mathsf{B}_{f,g})(\forall\, args\ x)$	$(\mathsf{S}_{f,g}\,x) = \langle (fx), (gx)\rangle\ \wedge\ (\mathsf{B}_{f,g}\,x) = (f(gx))$.

[5]For information on Curry we are indebted to [Curry and Feys, 1958, §§1S,6S,8S], [Curry, 1980], and [Seldin, 1980b; Seldin, 1980a], also to discussions with Jonathan Seldin, whose [Seldin, 2007] gives a fuller account of Curry and Church than the present one. Curry died in 1982.

[6]The substitution rule in propositional logic states that from a formula Q we may deduce

statement of the substitution rule was missing from *Principia Mathematica* [Russell and Whitehead, 1913], as was admitted later by Russell himself, see [Russell, 1919, Ch.14, p.151]. Also the corresponding rule for predicate logic was stated incorrectly in the first edition of a leading textbook [Hilbert and Ackermann, 1928, Ch.3 §5 p.53], see remarks in [Church, 1956, §49 pp. 290–291].

Around 1926–27, Curry began to look for a way to break the substitution process down into simpler steps, and to do this, he introduced a concept of combinator essentially the same as Schönfinkel's. Then in late 1927 he discovered [Schönfinkel, 1924], and a note in his files [Curry, 1927, p.1] admits "This paper anticipates much of what I have done". He did not abandon his own work, however, but travelled to Göttingen and in 1930 completed a doctoral thesis under the guidance of Bernays, [Curry, 1930]. In it he set out a formal system of combinators with a proof of the combinatory completeness of $\{\mathsf{B},\mathsf{C},\mathsf{K},\mathsf{W}\}$ [Curry, 1930, p.528 Hauptsatz II].[7]

Crucial to every such completeness proof is an abstraction algorithm. The one in Curry's thesis was not only the first to appear in print, but was also much more efficient than many that followed it. He saw the problem of producing an A such that $Ax_1...x_n$ converted to X, from the point of view of building X from the sequence $x_1,\ldots,x_n$. First, A must remove from $x_1,\ldots,x_n$ all variables which do not occur in X; this can be done using K and B. Then A must repeat variables as often as they occur in X; this can be done using W and B. Next, A must re-arrange the variables into the order in which they occur in X; this can be done using C and B. Finally, A must insert the parentheses that occur in X, and this can be done using B (together with I or CKK).[8]

But the details of Curry's algorithm were rather complex, and it was later replaced by ones that were much simpler to describe. These algorithms were multi-sweep; i.e. they built $[x_1,\ldots,x_n].X$ by building first $[x_n].X$, then $[x_{n-1}].[x_n].X$, etc., and their key was to build $[x].X$ for all x and X by a direct and very simple induction on the number of symbols in X.

The first to appear was in [Church, 1935, §3 p.278]. Church used just two basic combinators I and J, where $\mathsf{J}UXYZ = UX(UZY)$; the latter had been proposed by Rosser in 1933 [Rosser, 1935, p.128].[9]

But the simple algorithm based on S and K that is used in textbooks today is due to Paul Rosenbloom [Rosenbloom, 1950, p.117]. He seems to have been the first person, at least in print, to have realised that by using S in the induction step for $[x].X$ the proof of its main property $([x].X)Y = [Y/x]X$ was made trivial.[10] This algorithm, with some variants, became generally accepted as standard.[11]

$[P_1/p_1,\ldots,P_n/p_n]Q$, where the latter is obtained from Q by substituting formulas $P_1,\ldots,P_n$ for propositional variables $p_1,\ldots,p_n$ which may occur in Q.

[7] W has the axiom-scheme $\mathsf{W}XY = XYY$.

[8] See [Curry, 1930, Part II, §§C–E] for details, or [Curry and Feys, 1958, §6S2] for an outline.

[9] The basis $\{\mathsf{I},\mathsf{J}\}$ allows $[x].X$ to be built when x occurs in X but not when x is absent. This corresponds to the λI-calculus, see p.7 below. Church's algorithm is also in [Church, 1941, p.44].

[10] No he wasn't. [Seldin, 2007] points out that Curry realized this neat use of S while reading [Rosser, 1942], though, due to war work, he did not publish it until [Curry, 1949, p.393]. Thus the algorithm's direct inductive form is due to Church and its neat use of S to Curry and Rosenbloom independently.

[11] Some $[x].X$-algorithms were compared in [Curry and Feys, 1958, §§6A2–3]. Curry assumed the basic combinators were some of B, C, I, K, S, and $[x].X$ was built by induction on X using some or all of the following clauses (in which U does not contain x, but V may):

(a)	$[x].U$	$\equiv \mathsf{K}U,$	(d)	$[x].UV$	$\equiv \mathsf{B}U([x].V),$
(b)	$[x].x$	$\equiv \mathsf{I},$	(e)	$[x].VU$	$\equiv \mathsf{C}([x].V)U,$
(c)	$[x].Ux$	$\equiv U,$	(f)	$[x].X_1X_2$	$\equiv \mathsf{S}([x].X_1)([x].X_2).$

For example, let the basis be $\{\mathsf{I},\mathsf{K},\mathsf{S}\}$. Then algorithm (fab) builds $[x].X$ by first applying (f) whenever possible, then (a), then (b) when the other two do not apply. Alternatively, (abcf) first tries to apply (a), then (b), then (c), and (f) only when the other three do not apply; it usually produces shorter terms $[x].X$. The Curry-Rosenbloom algorithm was (fab).

If the basis is $\{\mathsf{K},\mathsf{S}\}$ one can first define $\mathsf{I} \equiv \mathsf{SKK}$ and then use (fab) or (abcf). If the basis is

However, although neater to describe, the direct inductive algorithms were not so efficient to use: they produced considerably longer outputs than Curry's first one, and in the 1970s when the desire arose to use combinators in practical programming languages, and attention began to focus on efficiency, the algorithms that were invented then were more like Curry's original, see for example [Kearns, 1973; Abdali, 1976; Turner, 1979; Piperno, 1989].

Besides the first abstraction algorithm, Curry's thesis contained the first formal definition of conversion. Further, this definition included a finite set of axioms from which he proved the admissibility of rule (ζ):[12]

(ζ) *if* $Ux = Vx$ *and* x *does not occur in* UV, *then* $U = V$.

Thus the discovery that (ζ) could be finitely axiomatized was made very soon after the birth of the theory. The theory of equality was also proved consistent in a certain sense, see [Curry, 1930, p.528 Hauptsatz I], but without using a concept of strong reduction or a confluence theorem.

The system in [Curry, 1930] also contained some logical constants. But not all their usual rules were included, negation was omitted, and logic was not actually developed beyond the theory of conversion. In fact, unlike von Neumann, Curry aimed, not at quickly building a system strong enough to deduce a significant part of mathematics, but at making a thorough study of basic logical notions in as general a setting as possible.

A consequence of this interest in generality was that for Curry, as for Schönfinkel, every combinator was allowed to be applied to every other combinator and even to itself, in a way not commonly accepted for set-theoretic functions. This freedom was later criticized on semantic grounds by several other leading logicians, e.g. Willard V. Quine in [Quine, 1936a, p.88]. In fact Curry's use of combinators was connected very closely to his philosophy of mathematics: for him, a formal system was not a description of some pre-existing objects, but simply a structure built by induction from some primitives using certain rules, and he did not demand that a semantics for such a system be a total function. In his view a system could be used fruitfully in applications of mathematics without necessarily every part of it having a meaning in some set-theory-based semantics, and usefulness was more important than semantic totality.[13]

In [Curry, 1964] he pointed out that the combinators constructed by his original abstraction algorithm were much easier to interpret in a set-theoretic semantics than those built by simpler algorithms using K and S. (But, as mentioned above, he himself was perfectly happy to use the simpler algorithms.)

However, semantic questions apart, by the end of the 1920s the combinator concept had provided two useful formal techniques: a computationally efficient way of avoiding bound variables, and a finite axiomatization of set theory.

4 1930s: Birth of λ and Youth of CL

4.1 Early λ-calculus

The λ-calculus was invented in about 1928 by Alonzo Church, and was first published in [Church, 1932]. Church was born in 1903 in Washington D.C. and

$\{\mathsf{B},\mathsf{C},\mathsf{K},\mathsf{W}\}$ one can define $\mathsf{S} \equiv \mathsf{B}(\mathsf{B}(\mathsf{BW})\mathsf{C})(\mathsf{BB})$ and $\mathsf{I} \equiv \mathsf{CKK}$ or WK, and then use (abcdef).

[12] See [Curry, 1930, p.832 Satz 4] or [Curry, 1932, p.558 Theorem 5]. Rule (ζ) makes Curry's equality equivalent to what is nowadays called $\beta\eta$-equality or *extensional equality*.

[13] See [Curry, 1951], [Curry, 1963, Chs. 2–3] or [Seldin, 1980a]. In the 1920s the general concept of formal system was still in the process of clarification and Curry was one of the pioneers in this.

studied at Princeton University. He made his career at Princeton until 1967, though in 1928–29 he visited Göttingen and Amsterdam.[14]

Around 1928 he began to build a formal system with the aim of providing a foundation for logic which would be more natural than Russell's type theory or Zermelo's set theory, and would not contain free variables (for reasons he explained in [Church, 1932, pp. 346–347]). He chose to base it on the concept of function rather than set, and his primitives included abstraction $\lambda x[M]$ and application $\{F\}(X)$, which we shall call here "$\lambda x.M$" and "(FX)".

(By the way, why did Church choose the notation "λ"? In [Church, 1964, §2] he stated clearly that it came from the notation "$\hat{x}$" used for class-abstraction by Whitehead and Russell, by first modifying "$\hat{x}$" to "$\wedge x$" to distinguish function-abstraction from class-abstraction, and then changing "$\wedge$" to "λ" for ease of printing. This origin was also reported in [Rosser, 1984, p.338]. On the other hand, in his later years Church told two enquirers that the choice was more accidental: a symbol was needed and "λ" just happened to be chosen.)

As mentioned earlier, Church was not the first to introduce an explicit notation for function-abstraction. But he was the first to state explicit formal conversion rules for the notation, and to analyse their consequences in depth. (He did not read Frege's work until 1935, according to a letter cited in [Scott, 1980a, p.260], and we assume he did not see Peano's 1889 notation either.)

Church's system as it appeared in [Church, 1932] was a type-free logic with unrestricted quantification but without the law of excluded middle. Explicit formal rules of λ-conversion were included. However, almost immediately after publication a contradiction was found in it. The system was revised a year later, [Church, 1933], and in the revised paper Church stated a hope that Gödel's recent incompleteness theorems for the system of [Russell and Whitehead, 1913] did not extend to his own revised system, and that a finitary proof of the consistency of this system might be found, [Church, 1933, pp. 842–843]. At that time the full power of Gödel's theorems was only slowly becoming generally understood.

At the end of his 1933 paper, Church introduced the representation of the positive integers by the λ-terms now known as the *Church numerals* [Church, 1933, §9]:

$$\mathbf{1} \equiv \lambda xy.\, xy, \quad \mathbf{Succ} \equiv \lambda xyz.\, y(xyz), \quad \mathbf{n} =_\beta \lambda xy.\, \underbrace{x(\cdots(x}_{n \text{ times}}\, y)\cdots).$$

He defined the set of all positive integers in essentially the same way as the set of all finite cardinal numbers in [Russell, 1903, §123], except that Russell's set included zero. The natural λ-term to represent zero would have been $\lambda xy.y$, but although Church included this term in his language, he forbade the reduction $(\lambda xy.y)FY \triangleright Y$ and avoided zero.

(Church's name is often associated with the λI-*calculus*, the version of λ-calculus in which $\lambda x.M$ is only counted as a term when x occurs free in M. But he did not limit himself so strictly to this version as is often thought. In 1932 and '33 he allowed non-λI-terms $\lambda x.M$ to exist but not to be "active", i.e. he did not allow a term $(\lambda x.M)N$ to be contracted when x did not occur free in M, see [Church, 1932, pp. 352, 355]. In his 1940 paper on type theory, where consistency would have been less in doubt, although probably not yet actually proved, he was happy for non-λI-terms to be active [Church, 1940, pp. 57, 60]. Only in his 1941 book did he forbid their very existence [Church, 1941, p. 8].)[15]

[14] For more on Church's life, see [Manzano, 1997, §2] and the obituary [Enderton, 1995]. For more on his work, see [Manzano, 1997], [Anderson, 1998], [Enderton, 1998], [Sieg, 1997], and (on the impact of λ-calculus) [Barendregt, 1997]. He died in 1995.

[15] The unrestricted λ-calculus is often called the λK-*calculus*.

From 1931 to 1934 in Princeton University, Church received the help of two outstanding graduate students, Stephen Kleene and Barkley Rosser, and in a remarkable four years of collaboration this group made a series of major discoveries about both Church's 1933 logic and the underlying pure λ-calculus. Unfortunately for the 1933 logic, one of these discoveries was its inconsistency, as a variant of the Richard paradox was proved in the system, [Kleene and Rosser, 1935].

In contrast, the pure λ-calculus, which had at first seemed shallow, turned out to be surprisingly rich. (Well, Kleene was surprised, anyway, [Kleene, 1981, p.54].) The study of its reduction-theory was begun using the method of residuals, which was applied to prove the Church-Rosser confluence theorem [Church and Rosser, 1936, p.479], and this ensured the consistency of the pure system. The system's relation to combinatory logic was also clarified, and a simple proof of the confluence of combinatory weak reduction was given; see [Rosser, 1935, esp. pp. 145–146].[16]

Further, the λ-definable numerical functions were found to form a much more extensive class than expected, and this class was eventually proved equivalent to two definitions whose purpose was to formalize the informal concept of effectively calculable function: the Herbrand-Gödel recursive functions, see [Kleene, 1936] and [Church, 1936b], and the Turing computable functions, [Turing, 1937a]. In view of this, Church conjectured that λ-definability exactly captured the informal concept of effective calculability (Church's Thesis).[17] The high point of this definability work was Church's negative solution of Hilbert's long-standing *Entscheidungsproblem* for first-order logic: first he proved in [Church, 1936b] that the convertibility problem for pure λ-calculus was recursively undecidable, then he deduced that no recursive decision procedure existed for validity in first-order predicate logic, see [Church, 1936a].

But this result was also proved independently almost immediately afterwards by Alan Turing in England. In Cambridge in the 1930s the topologist Max Newman had maintained a side-interest in the foundations, and in early 1935 he taught a course on this topic which Turing attended. Later that year Turing wrote his now famous paper on computability, [Turing, 1936]. He then visited Princeton and took a doctorate under Church (in 1936–38).

Incidentally, in the course of his doctoral work Turing gave the first published fixed-point combinator, [Turing, 1937b, term Θ]. This was seen as only having minor interest at that time, but in view of the later importance given to such combinators (see §8.1.2 below), we digress here to discuss them.

A *fixed-point combinator* is any closed term $\mathbf{Y}$ such that $\mathbf{Y}x$ converts to $x(\mathbf{Y}x)$. Turing's was

$$(\lambda xy.y(xxy))(\lambda xy.y(xxy)).$$

The next one to appear was in [Rosenbloom, 1950, pp.130–131, Exs. 3e, 5f]. It was $\lambda x.\,\mathbf{W}(\mathbf{B}x)(\mathbf{W}(\mathbf{B}x))$, which is convertible to $\lambda x.\,(\lambda y.x(yy))(\lambda y.x(yy))$. The latter has often been called Curry's $\mathbf{Y}$; in fact Curry gave no explicit fixed-point combinator before [Curry and Feys, 1958, §5G], but the accounts of Russell's paradox in [Church, 1932, p.347] and [Curry, 1934a, §5] both mentioned $(\lambda y.N(yy))(\lambda y.N(yy))$, where N represented negation, and Curry's use of the latter term dated back to a

[16] For an introduction to reduction-theory in general, see [Klop, 1992] or (less general) [Barendregt, 1981, §3.1], or (in λ) [Hindley and Seldin, 1986, §1C & Appendices 1 & 2] or [Barendregt, 1981, Chs. 11–15]. Some notation: a reducibility relation $\triangleright$ is *confluent*, or has the *Church-Rosser property*, iff $(U \triangleright X \;\&\; U \triangleright Y) \Longrightarrow (\exists Z)\ (X \triangleright Z \;\&\; Y \triangleright Z)$. It is *locally confluent* iff $(U \triangleright_1 X \;\&\; U \triangleright_1 Y) \Longrightarrow (\exists Z)\ (X \triangleright Z \;\&\; Y \triangleright Z)$, where "$\triangleright_1$" means one-step reducibility. A one-step reduction is called a *contraction*. A contraction $X \triangleright_1 Y$ is made by replacing an occurrence in X of a term called a *redex* by another term called its *contractum*. In λ, a (β-) redex is any term $(\lambda x.M)N$ and its contractum is $[N/x]M$. A term containing no redexes is called a *normal form*; it cannot be reduced.

[17] Comments on the history of Church's Thesis are in: [Kleene, 1981], [Kleene, 1952, §62], [Rosser, 1984], [Davis, 1982], [Gandy, 1988] and [Sieg, 1997].

letter to Hilbert in 1929.[18] Note that although fixed-point combinators are often used in λ-defining recursion, they are not really necessary for recursion on the natural numbers; none was used in the λ-representation of the recursive functions in [Kleene, 1936].

Returning to the Princeton group: their successes in pure λ-calculus were not really matched on the applied side, but the results were by no means completely negative. After the discovery of the Kleene-Rosser inconsistency, Church replaced his 1933 system by a free-variable logic based on the $\lambda\delta$-calculus, which he proved consistent by extending the Church-Rosser theorem. (In essence δ was a discriminator between normal forms; see [Church, 1935] or [Church, 1941, §20], and the review [Curry, 1937].) But the new system was too weak for much mathematics and seems to have played no role in Church's future work. In fact in the later 1930s Church retreated from the task of designing a general type-free logic to the much less ambitious one of re-formulating simple type theory on a λ-calculus base. In this he succeeded, publishing a smooth and natural system in [Church, 1940] that has been the foundation for much type-theoretic work since (see §5.1 and §8 below).

4.2 CL in the 1930s

The work of Curry through the 1930s continued his earlier study of the most basic properties of abstraction, universal quantification and implication in as general a setting as possible. He analysed the Kleene-Rosser proof of the inconsistency of Church's system, and by 1942 he had found in a logical system of his own a very simple inconsistency now called the *Curry paradox* [Curry, 1942b]. This showed combinatory completeness to be incompatible with unrestricted use of certain simple properties of implication in a very general setting.[19]

Curry also began to develop a *type theory*. A note on types dated 1928 occurs in his files, and he gave a talk on this topic in 1930 and wrote it up in [Curry, 1934a] and [Curry, 1936].[20] He took a different approach from Russell and from the one that Church later took: he added to his combinator-based logic a *functionality constant* F, with the intention that an expression such as $\mathsf{F}abf$ should mean approximately $(\forall x)(x \in a \supset fx \in b)$.[21] This approach differed from Church's in that Curry's $\mathsf{F}abf$ did not imply that the domain of f was exactly a, only that it included a. In Curry's functionality-theory a term could have an infinite number of types. Indeed his axioms for F included axioms assigning an infinity of type-expressions to each atomic combinator B, C, K, W; for example (from [Curry, 1934a, p.586] or [Curry, 1936, p.378])

$$\text{Ax. (FB):} \quad (\forall x, y, z)\ \Big(\big(\mathsf{F}(\mathsf{F}xy)(\mathsf{F}(\mathsf{F}zx)(\mathsf{F}zy))\big)\ \mathsf{B}\Big).$$

[18] Stated in [Curry and Feys, 1958, §5S]. By the way, Turing's fixed-point combinator satisfied $\mathbf{Y}x \triangleright x(\mathbf{Y}x)$, but Rosenbloom's and Curry's only satisfied "=" not "$\triangleright$". For more on fixed-point combinators, see [Barendregt, 1981, §§6.1, 6.5], [Hindley and Seldin, 1986, §3B], or [Curry *et al.*, 1972, §11F7].

[19] A general form of Curry's paradox runs thus, cf. [Hindley and Seldin, 1986, pp.268–269]. Suppose a system $\mathcal{S}$ of logic has a class $\mathcal{P}$ of expressions such that $X, Y \in \mathcal{P} \Longrightarrow (X \supset Y) \in \mathcal{P}$, and has an axiom-scheme $(X \supset (X \supset Y)) \supset (X \supset Y)$ (for all X, $Y \in \mathcal{P}$), and rules (Eq): X, $X{=}Y \vdash Y$ and (MP): X, $X \supset Y \vdash Y$.

For every $Z \in \mathcal{P}$, let Z' be $\lambda y.((yy) \supset ((yy) \supset Z))$, where y does not occur in Z, and let Z^* be $Z'Z'$. It is easy to prove that $Z^* = Z^* \supset (Z^* \supset Z) = (Z^* \supset (Z^* \supset Z)) \supset (Z^* \supset Z)$.

Suppose $Z \in \mathcal{P} \Longrightarrow Z^* \in \mathcal{P}$, for all Z. Then every Z in $\mathcal{P}$ is provable in $\mathcal{S}$. Because, the axiom-scheme gives $\vdash (Z^* \supset (Z^* \supset Z)) \supset (Z^* \supset Z)$, so by rule (Eq) twice, $\vdash Z^* \supset (Z^* \supset Z)$ and $\vdash Z^*$; hence by (MP) twice, $\vdash Z$. (Curry's treatment of the Russell paradox in [Curry, 1934a, pp.588–589] was interestingly similar to this argument.)

[20] We are very grateful to J. P. Seldin for showing us Curry's 1928 note (T 28.12.13.A, reproduced in [Seldin, 2002]).

[21] But only approximately; see the discussions of Π and F in [Seldin, 2007].

Curry's approach was what we would now call *polymorphic*. In contrast, in a Church-style type-theory, for a term $f^{\alpha\to\beta}$, its domain would be exactly α and its type would be uniquely $\alpha \to \beta$.

Curry's use of types differed from Russell and Church in another way. Russell had introduced type-expressions to help him restrict his language to avoid certain troublesome expressions, and Church later had the same aim. But Curry allowed into his language all combinators, troublesome or placid, and simply assigned labels to some of them to tell us how they would behave.

Another early user of a polymorphic approach to types was Quine. In [Quine, 1937, pp. 78–79] he proposed restricting the comprehension axiom-scheme in set theory to formulae he called *stratified*, but which we would now call *typable*, and on p.79 he made a very clear argument in favour of a polymorphic approach. His proposed system is nowadays called NF for "New Foundations". Its history is beyond our scope, but the equivalence between stratification and typability was proved in [Newman, 1943, Thm. 4], and a general study of NF is in [Forster, 1995].

The *propositions-as-types* correspondence was noticed by Curry in the 1930s. In its simplest form this is an isomorphism between the type-expressions assignable to combinators and the implicational formulas provable in intuitionistic logic. The first hint of this, at least for atomic combinators, occurs in a note dated 1930 in Curry's files, and appeared in print in his papers [Curry, 1934a] and [Curry, 1934b].[22] In [Curry, 1934a, p.588] and [Curry, 1934b, p.850] he gave his axioms for implication the names of combinators, for example

$$\text{Ax. (PB):} \quad (\forall x, y, z)\ \big(\mathbf{P}(\mathbf{P}xy)(\mathbf{P}(\mathbf{P}zx)(\mathbf{P}zy))\big),$$

where $\mathbf{P}xy$ was his notation for $x \supset y$. His name "(PB)" was clearly motivated by the axiom's similarity to Ax. (FB), although he did not state the **P**-to-**F** translation explicitly in print until later, [Curry, 1942a, p.60, footnote 28]. In [Curry, 1936, pp.391–394] he proved that if **F** was defined as

$$\mathbf{F} \quad \equiv \quad [a, b, f].\ (\forall x)(ax \supset b(fx))$$

in a system with combinators and suitable rules for $\forall$, then each **F**-axiom would be deducible from the corresponding **P**-axiom.

For non-atomic combinators the propositions-as-types correspondence was not mentioned by Curry until many years later; in fact the first discussion that included composite combinators was in [Curry and Feys, 1958, §9E]. There it was proved explicitly that when **F** was replaced by **P**, the set of all type-expressions assignable to the combinators became exactly the set of all formulas provable in intuitionistic implicational logic, provided type-assignment was made in a certain restricted system. This system had no $\forall$, had axiom-schemes instead of the previous **F**-axioms for the atomic combinators, and its main rule was:

$$\text{Rule } \mathbf{F}: \qquad \mathbf{F}ABF,\ \ AX \ \vdash\ B(FX)$$

(which would become the rule of *modus ponens* if **F** was replaced by **P** and F, X, FX were deleted). It had only one other rule, to ensure that interconvertible terms received the same types, [Curry and Feys, 1958, §9A3, Rule Eq′]; this rule did not affect the total set of assignable types, and was dropped in later studies of propositions-as-types. After dropping that rule, it became clear that there was also a correspondence between type-assignment deductions (or typed terms) and propositional deductions. This fact, and its analogue for λ, and some possible extensions

[22] We thank J. P. Seldin for a copy of Curry's 1930 note (T 30.07.15.B, reproduced in [Seldin, 2002]). The paper [Curry, 1934b] was based on a talk given in 1932.

of the correspondence, were pointed out by William Howard in a manuscript circulated in 1969 and published in [Howard, 1980]. The correspondence is often called *formulae-as-types*, following [Howard, 1980][23], or the *Curry-Howard isomorphism*.

The propositions-as-types correspondence was also discovered independently by several other workers in the 1960s; this period will be discussed further in §8.1.4 below.

Partly as a consequence of propositions-as-types, the authors of [Curry and Feys, 1958] were led to apply Gentzen's methods to type-theory, an idea which is standard now but was new then. This gave them the first published proof of the normalization theorem for typable terms, see §5.4 below.

In the 1930s another logician who made use of combinators was Frederic Fitch in Yale University. Fitch's work began with his doctoral thesis [Fitch, 1936], which described a type-free combinator-based logic that avoided Russell's paradox by forbidding certain cases of abstraction $[x_1, ..., x_n].X$ where X contained repeated variables. He then developed over several decades a system CΔ, very different from the one in his thesis but still type-free and combinator-based, with unrestricted quantification and abstraction; see [Fitch, 1963]. The set of theorems of CΔ was not recursively enumerable, [Fitch, 1963, p.87], but the system was adequate for a considerable part of mathematical analysis and had a (non-finitary) consistency proof. He presented a further modified system in his student-level textbook on CL, [Fitch, 1974].

Incidentally, combinators are not the only way of eliminating bound variables: an alternative technique is *algebraic logic*. Very roughly speaking, while CL algebraizes abstraction the latter algebraizes quantification. In 1936 Quine sketched how this could be done in principle by means of an algebra of relations, [Quine, 1936b]. (Such algebras had been published earlier in [Peirce, 1883] and [Schröder, 1905, Vol.III].) In 1938 Leopold Löwenheim proposed to Alfred Tarski that all of mathematics could be expressed in a Peirce-Schröder algebra, see [Löwenheim, 1940], and in 1941 Tarski developed an abstract theory of binary relations, called the theory of *relation algebras*, and used it to prove that set theory could be formalized without variables, see [Tarski, 1941] and [Tarski, 1953]. A detailed formalization of set theory without variables was carried out in the book [Tarski and Givant, 1987], completed just before Tarski died in 1983.

However, for a formalization of first-order logic in general, Tarski's theory was inadequate, and to remedy this he and his co-workers developed the theory of *cylindric algebras* from the late 1940s onward. A similar algebraic treatment of bound variables was also given by the *polyadic algebras* of Paul Halmos in 1956, see [Halmos, 1962]. Other significant contributions to the algebraic approach were made by [Bernays, 1958], and [Craig, 1974]. William Craig had previously contributed to CL, see [Curry and Feys, 1958, §§5H, 7E].

For more on the history of algebraic logic, see the introductions to [Tarski and Givant, 1987] and [Givant and Andréka, 2002]. A good comparison of algebraic logic with type-free CL is in [Quine, 1972]. From the viewpoint of a semantics based on a standard set theory such as ZF or NBG, an advantage of algebraic logic is that it retains the restrictions of first-order logic, and therefore its semantic simplicity; as noted earlier, unrestricted type-free combinators are much harder to interpret in standard set theories.

A different algebraic approach was taken by the mathematician Karl Menger.

[23] Although this name was later criticised by Howard himself who said "a type should be regarded as an abstract object whereas a formula is the name of a type", [Howard, 1980, p.479].

By the way, in [Frege, 1879, §13] the axioms for implication were the principal types of **K**, **S** and **C**. These might be seen, via propositions-as-types, as a partial anticipation of the concept of combinator basis, at least for the typable terms. But this is probably far-fetched; in particular, [Frege, 1879] had no explicit concept of completeness.

Menger's method involved an algebraization of a limited form of function-composition, and was expounded by him in a series of papers from the 1940s to the '60s, for example [Menger, 1944] and [Menger, 1964]. But his system is more restricted than Tarski's, and less mathematics has been developed in it; see the review [Lercher, 1966].

5 1940s and 1950s: Consolidation

For many years after the mid-1930s, neither λ nor CL attracted much interest among logicians in general. Both systems would later play a natural role in studies of higher-order logic, but the formal metatheory of that topic had hardly begun to be developed at that early period. The λ-formalism had been designed to describe substitution and application in detail and seemed off-puttingly heavy for other purposes. It had given the first proof of the unsolvability of the *Entscheidungsproblem*, but Turing's proof via his machine model of computation was much more transparent, and even Kleene preferred other systems for his subsequent development of computability theory.

However, on the expository side, an advance in opening up λ to non-specialists was made by Church with his readable introductory textbook [Church, 1941]. And Paul Rosenbloom included a section on CL in his general logic textbook [Rosenbloom, 1950, Ch.III, §4]. Advances were made in both subjects on the technical side too, and these helped to lay a foundation for the expansion that came later. They can be split roughly into four main themes: simple type theory, abstract reduction theory, reductions in CL and λ, and illative systems.

5.1 Simple type theory

Church's simple type theory was a function-based system, stemming from ideas of Frank Ramsey and Leon Chwistek in the 1920s, for simplifying the type theory of [Russell and Whitehead, 1913].[24] Church lectured on his system in Princeton in 1937–38 before publishing it in [Church, 1940], and his lectures were attended by Turing, who later made some technical contributions. These included the first proof of the system's weak normalization theorem, although this lay unpublished for nearly 40 years, see [Turing, 1980] and §8.2 below.

Church's system was analysed and extended in a series of Princeton Ph.D. theses from the 1940s onward, of which perhaps the best known are Leon Henkin's in 1947, published in [Henkin, 1950], and Peter Andrews', published in [Andrews, 1965]. Henkin gave two definitions of model of typed λ (*standard* and *general* models),[25] and proved the completeness of simple type theory with respect to general models. Andrews extended Church's system to make a smooth theory of tranfinite types.

Church's own interests underwent some evolution after his 1941 textbook appeared. Much of his time and energy was spent on the business of the Association for Symbolic Logic, of which he had been a co-founder in 1936 and in which he was still a driving force, and on editing the Journal of Symbolic Logic, especially its very comprehensive Reviews and Bibliography sections. On the research side he became interested in intensional logic and the logic of relevant implication. For the former he made axiomatizations in [Church, 1951a] and [Church, 1973], both based on modifications of his type theory. For relevant implication he gave a formulation, in [Church, 1951b], which was essentially equivalent, under the propositions-as-types correspondence, to the simple type theory of pure λI-terms.

[24] See [Gandy, 1977] for a sketch of the origins of Church's system (and of Turing's contributions).
[25] See §8.1.2 below for the difference between these; details are in [Henkin, 1950, pp.83–85].

5.2 Abstract reduction theory

Probably the first publication on abstract reduction-theory was a paper by the mathematician Axel Thue on rewriting finite trees [Thue, 1910].[26] Thue's paper even included a lemma that local confluence implies confluence if every contraction strictly decreases a finite measure [Thue, 1910, §4 Satz 3], very like Newman's lemma of 30 years later that local confluence implies confluence if all reduction paths are finite [Newman, 1942, Thm. 3].

But the main inspiration for abstract work in this field was [Newman, 1942]. In the 1920s, Newman proved a crucial confluence conjecture in combinatorial topology, and a decade later his interest in logic (and probably his link with Turing) led him to study the confluence theorem for λ and the method of residuals that Church and Rosser had used in its proof [Church and Rosser, 1936].[27] The first part of [Newman, 1942] was a study of confluence in a very general setting; in the second, Newman gave a set of abstract axioms about the concept of residual, from which he deduced confluence. Unfortunately, due to an error involving variables, his axioms failed to cover λ as a special case. This was first corrected about 20 years later by David Schroer in his Ph.D. thesis [Schroer, 1965].

Newman's analysis led in the 1970s to general methods for proving confluence and other key theorems, such as standardization, for systems of λ augmented by extra operators, for example a primitive recursion operator. (A summary of these results is in [Hindley, 1978b, pp.269–270].) Two influential abstract studies of confluence were [Rosen, 1973] and [Huet, 1980].

In the 1980s the abstract theory of reductions was re-organised completely and developed by Jan-Willem Klop, beginning with his penetrating thesis [Klop, 1980], and from this work the important abstract theories of *combinatory reduction systems* and *term rewriting systems* evolved. These are being actively pursued today, but go far beyond the boundaries of λ and CL and are the subjects of good surveys in [Klop, 1992] and [Klop *et al.*, 1993] and a comprehensive textbook [Terese, 2003], so we shall not give details of them here.

5.3 Reductions in CL and λ

Besides abstract reduction theory, there was also progress in analysing the two most important concrete reductions, weak reduction in CL and β-reduction in λ. As mentioned earlier, Church and Rosser had begun the study of these in the paper containing their confluence proof, [Church and Rosser, 1936].

After 1950 their programme was continued by Curry. His research had been suspended completely during the Second World War, like that of many other academics, and did not recover momentum until quite late in the 1940s. Then for about 20 years he put his main effort into compiling two large monographs containing much new work, [Curry and Feys, 1958] and [Curry *et al.*, 1972]. Part of this effort consisted of a detailed study of $\lambda\beta$-reduction, including the first proof of its standardization theorem, [Curry and Feys, 1958, §4E1]. He also made the first study of $\lambda\beta\eta$-reduction, including its confluence and η-postponement theorems, [Curry and Feys, 1958, §4D]. (But his proof of the latter contained a gap,

[26] See [Steinby and Thomas, 2000], which contains an English summary of [Thue, 1910].

[27] If a term X contains redex-occurrences $R_1, ..., R_n$, and $X \triangleright_1 Y$ by contracting R_1, certain parts of Y can be considered to be descendants of $R_2, ..., R_n$ in a sense; they are called *residuals* of $R_2, ..., R_n$; for details in λ, see [Curry and Feys, 1958, §§4B1–2] or [Hindley and Seldin, 1986, Def. A1.3], or [Church and Rosser, 1936, pp.473–475]. If we reduce Y by contracting one of these residuals, and then a residual of these residuals, etc., the reduction $X \triangleright_1 Y \triangleright$ ... is called a *development* of $R_1, ..., R_n$. If it is finite and its last term contains no residuals of $R_1, ..., R_n$, it is called *complete*.

which was noticed and remedied fifteen years later by Robert Nederpelt, [Nederpelt, 1973, Thm. 7.28].)

In 1965 the thesis of Schroer gave the first statement and proof of the finiteness-of-developments theorem for $\lambda\beta$, [Schroer, 1965, Part I, Thm. 6.20]. But Schroer's work was never published and this theorem was not re-discovered until about 8 years later, see §7.2 below.

Weak reduction in CL is much simpler than β-reduction in λ, and was given a straightforward systematic analysis in [Curry *et al.*, 1972, §11B].

Curry also defined a *strong reduction* for CL analogous to $\lambda\beta\eta$-reduction, [Curry and Feys, 1958, §6F].[28] His purpose was to make later discussions of logic applicable equally to languages based on CL and on λ. Strong reduction indeed allowed this to be done to a significant extent, but its metatheory turned out to be complicated, and although simplifications were made by Roger Hindley and Bruce Lercher in the 1960s which produced some improvement, it eventually seemed that wherever strong reduction could be used, it would probably be easier to use λ instead of CL. Strong reduction was therefore more or less abandoned after 1972. However, in the 1980s a β-strong reduction was defined and studied by Mohamed Mezghiche, see [Mezghiche, 1984], and the idea of working with a combinatory reduction which is preserved by abstraction but avoids the complexity of bound variables is still tantalisingly attractive.

5.4 Illative systems

Turning from pure to applied CL and λ: although type-theory made important advances in the 1950s, some logicians felt that type-restrictions were stronger than necessary, and that type-free higher-order logic was still worth further study. Most systems of such logic contain analogues of combinators in some form, but here we shall only mention those in which CL or λ were more prominent.

Curry's work on applied systems (which he called *illative*, from the Latin "illatum" for "inferred") was published in the books [Curry and Feys, 1958] and [Curry *et al.*, 1972]. He never gave up his interest in type-free logic, although one of his main themes of study at this time was the relation between types and terms.

In his 1958 book with Robert Feys, perhaps the authors' most important contribution in the illative sections was to introduce Gentzen's techniques into type theory. As we mentioned in §4.2, this approach was not as obvious in 1958 as it is now. Feys had written on combinatory logic and on Natural Deduction in [Feys, 1946a; Feys, 1946b], and Curry had emphasized the importance of Gentzen's methods in a series of lectures on first-order logic at the University of Notre Dame in 1948. In [Curry and Feys, 1958, §9F] they presented a sequent-based system of type-assignment and a proof of a cut-elimination theorem for this system, from which they deduced the weak normalization theorem for simple type theory [Curry and Feys, 1958, §9F6, Cor. 9F9.2]; this was the first published proof of this theorem. (Other proofs will be discussed in §8.2.) Further type-theoretic properties described in that book included the propositions-as-types correspondence in [Curry and Feys, 1958, §9E], see §4.2 above and §8.1.4 below, and the link with grammatical categories in [Curry and Feys, 1958, §8S2], see §8.1.1 below.

In [Curry *et al.*, 1972], Curry's main aim was to make a thorough analysis of basic logical concepts in the settings of λ and CL, with as few restrictions as possible and an emphasis on constructive methods.[29] The book contained a detailed comparison of logical systems based on three different concepts: (i) **F**, the functionality

[28] In CL, $\beta\eta$-strong reduction $\succ_{\beta\eta}$ is defined by adding to the rules defining weak reduction the extra rule (ξ): *if* $X \succ_{\beta\eta} Y$ *and* x *is any variable, then* $[x]_{abcf}.X \succ_{\beta\eta} [x]_{abcf}.Y$, where $[x]_{abcf}$ is evaluated by abstraction-algorithm (abcf), see §3 above.

[29] For further comments on Curry's programme see [Seldin, 1980a, pp.22–27].

constant described in §4.2 above, (ii) Ξ, where ΞAB was approximately equivalent to $(\forall x)(Ax \supset Bx)$, and (iii) $\forall$ and $\supset$. The formulation of these systems benefitted greatly from results and simplifications introduced by Jonathan Seldin in his thesis [Seldin, 1968], but many variants of each system were discussed and the resulting treatment was rather complex.

A proposal for a generalized functionality concept **G** was made by Curry in [Curry and Feys, 1958, Appendix A] and [Curry *et al.*, 1972, §§15A1, 15A8]. In terms of $\forall$ and $\supset$, $\mathbf{G}ABF$ was intended to mean approximately $(\forall x)(\, Ax \supset (Bx)(Fx)\,)$.[30] This proposal was developed into a fully-fledged higher-order type-theory by Seldin in 1975, see [Seldin, 1979] or [Hindley and Seldin, 1986, 1st edn., Ch. 17]. Seldin later became interested in Thierry Coquand's calculus of constructions, which was stronger, see [Seldin, 1997], and thus his work on Curry-style illative systems gradually merged with the work of others on higher-order systems. These will be discussed in §8.3 below.

Returning to the 1950s: at that time the main contributor to the system-building approach to type-free higher-order logic was Fitch. As described in §4.2, his work on this continued into the 1970s. Some of his technical ideas were appreciated and used by other workers, for example Corrado Böhm, see §6.3, and Solomon Feferman, see [Feferman, 1984, p.106]. But his formal systems themselves seem to have been little used.

The system-building tradition was continued by others, however. In 1974 in his thesis for Doctorat d' Etat, André Chauvin produced a thoughtful and interesting analysis of type-free higher-order logic and the role of combinators, and described a formal system based on the notion of *partial* operation or function. Unfortunately only a little of this thesis was published, [Chauvin, 1979].

The concept of partial function was also behind several substantial papers by Feferman from 1975 through the 1980s, in which he examined the possibilities for type-free systems, particularly as foundations for constructive mathematics and for category theory, and proposed several actual systems in which "partial" combinators played a role, see for example [Feferman, 1975a; Feferman, 1975b; Feferman, 1977; Feferman, 1984] and [Aczel and Feferman, 1980].

An analysis of the foundations based closely on λ was made by Peter Aczel in an influential paper [Aczel, 1980].

From 1967 onward, work on type-free illative systems was undertaken by Martin Bunder. He made a careful study of the inconsistencies that had arisen in past systems, and proposed several new systems that avoided these, including some in which all of ZF set theory can be deduced; see, for example, [Bunder, 1983a] and [Bunder, 1983c], depending on [Bunder, 1983b]. A good short overview is in [Barendregt *et al.*, 1993].

With Wil Dekkers and Herman Geuvers, Bunder and Barendregt published useful comparisons between their type-free illative systems and other higher-order formal systems such as Pure Type Systems: [Dekkers *et al.*, 1998] and [Bunder *et al.*, 2003].

In 1990, Randall Holmes formulated systems of λ and CL in which abstraction was restricted by stratification rules inspired by the stratification concept in Quine's "New Foundations" system [Quine, 1937]: see [Holmes, 1991], or [Holmes, 1995] for a short overview.

In Moscow, independently of all the above, a series of type-free combinator-based systems was proposed by Alexander Kuzichev from the 1970s onward; see, for example, [Kuzichev, 1980; Kuzichev, 1983; Kuzichev, 1999] and the references therein.

[30] In a **G**-based system the type of the output-value of a function may depend on the value of its argument. Type systems with this property are called *dependent*. Other systems of dependent types are mentioned in §8.3.

6 Programming languages

Returning to the 1960s: at about this time, λ and CL began to attract the interest of a new group outside the community of logicians, namely computer scientists involved in the theory and practice of programming languages. Perhaps the first published suggestion that CL or λ be used directly as a programming language came from the logician Fitch in 1957, although for only a restricted class of problems: in his paper [Fitch, 1958] he showed how sequential logic-circuits could be represented in CL.[31] But the main characters concerned were computer scientists: we shall look at three of these.

6.1 John McCarthy and LISP

From 1956 to '60, John McCarthy in the U.S.A. developed the computer language LISP, which was a list-processing language with a function-abstraction facility. McCarthy's goal was to apply LISP eventually to problems in non-numerical computation, especially in the newborn field of *artificial intelligence*, and to encourage the style of program-organization that is nowadays called *functional programming*.[32]

LISP was not directly based on λ-calculus, although it owed something to λ (and its abstraction notation was even called "LAMBDA"). LISP's substitution procedure was not defined exactly the same as in λ, in particular identifiers were handled according to so-called "dynamic binding" rules. This choice simplified LISP's implementation by means of interpreters, but it greatly complicated the use of bound variables by the programmer.[33]

McCarthy's emphasis on a functional approach to programming also had influence on the theoretical side, where his contributions included two important papers on a mathematical basis for the theory of computation, [McCarthy, 1963b; McCarthy, 1963a]. There we find a systematic use of conditional expressions within a functional formalism for recursively defining functions over symbolic expressions like the S-expressions of LISP, together with a thorough study of their formal properties. Equalities among recursive functions expressed in that formalism were proved by means of *recursion induction*, to be mentioned again in §8.1.2 in connection with its generalization as fixed-point induction by Scott [Scott, 1969e]. Along with recursive definitions of functions, [McCarthy, 1963a, §2.6] also discussed recursive definitions of *sets*, like the definition of the set of S-expressions itself as a solution of the equation $S \cong A + (S \times S)$.

6.2 Peter Landin

In the early 1960s in England, Peter Landin proposed the use of λ-terms to code constructs of the programming language Algol 60, see [Landin, 1965].[34] While for LISP a precise correspondence with λ was hindered by the use of dynamic binding, for Algol its block structure perfectly matched the way names were handled in λ. In fact, Landin's work made it possible to look at λ itself as a programming language, and one especially suited for theoretical purposes. Taking this standpoint, and in parallel with his translation of Algol, Landin described in 1963 an abstract machine

[31] We have recently learned that actually Fitch was not the first; Curry suggested using combinators in programming, in a talk in 1952, [Curry, 1954]; see [Seldin, 2007].

[32] This name dates back at least to a 1963 tutorial on LISP by P. M. Woodward [Fox, 1966, p.41].

[33] See [Moses, 1970] for comments. For the ideas behind LISP see [McCarthy, 1960], [McCarthy, 1963a]; for its history to 1996 see [McCarthy, 1981], [Steele and Gabriel, 1996]. One notable descendant of LISP with the variable-binding problem fixed is Scheme, see [Dybyg, 1996], dating from about 1975.

[34] More precisely, λ-terms extended with some primitive operations for dealing with jumps.

for reducing λ-terms considered as programs, the *SECD-machine* of [Landin, 1964, pp. 316–318] and [Landin, 1966a]. This consisted of a transition system whose states were made of four components:

- a *stack* S for storing intermediate results of evaluation,
- an *environment* E, that associates values to free identifiers,
- a *control* C consisting of a list of expressions that drives the evaluation process,
- a *dump* D that records a complete state of the machine, of the form (S', E', C', D').

The transition rules of the machine implemented the order of evaluation now known as *call by value*, so their results did not agree exactly with those of leftmost-first reduction of λ-terms. While the design of the SECD-machine was influenced by similar earlier devices for the interpretation of programming languages, notably the LISP interpreter devised in [Gilmore, 1963],[35] it was however the first to be presented in a formal way, and may fairly be said to have opened the way to the study of the *formal* operational semantics of λ-calculus considered as a programming language, that would find a systematic exposition a decade later starting with [Plotkin, 1975].

6.3 Corrado Böhm: λ-calculus as a programming language

In the above developments λ and CL began to contribute ideas to the theory of programming languages, and their influence is still active; but in the opposite direction, programming experience had only a limited impact on the "pure" theory of λ and CL at that time. The exception was the work of Corrado Böhm from the early 1960s onward.[36]

Böhm gained his Ph.D. in 1951 at the ETH in Zurich, with a thesis which included the first-ever description of a complete compiler in its own language, [Böhm, 1954]. While working on his thesis he came in contact with Bernays, who introduced him to the classical formalisms for computable functions. These included Turing machines and Post and Thue systems, but the variety of these models of computation led Böhm in the following years to search for an inclusive formalism. This aim was the origin of his interest in λ-calculus, with which he became acquainted around 1954 from Church's textbook [Church, 1941], but whose importance he realized only at the end of the 1950s after noticing that functional abstraction was used in LISP. A colleague, Wolf Gross, drew his attention to [Curry and Feys, 1958] and to the third chapter of [Rosenbloom, 1950], and in the years 1960-61 Böhm and Gross together began to study combinators, λ-calculus and recursive function theory. By 1964 they had realized that λ-terms and combinators could be used directly as the basis of "a very flexible description language for machines, programs and algorithms", [Böhm and Gross, 1966, p.36]. This was the starting point of the CUCH language (a system using both CUrry's combinators and CHurch's λ-terms), which was developed independently of the ideas that were emerging at about the same time from the work of Strachey and Landin, but was influenced partly by the use of λ-abstraction notation in [McCarthy, 1963a, §3] (and by the use of β-reduction in Paul Gilmore's interpreter [Gilmore, 1963, §3]), and especially by the use of combinators to describe sequential circuits in [Fitch, 1958]. The latter gave in fact one of the first applications of CUCH (described in [Böhm and Giovannucci,

[35] See [Landin, 1966a, §20] for a discussion of related work. A textbook treatment of the SECD-machine is contained in [Burge, 1978].

[36] Our reconstruction of Böhm's work has been helped by the scientific biography in [Dezani *et al.*, 1993, pp. 1–8], and a personal communication by Böhm, April 11th, 2001.

1964] together with the representation of analogue circuits), as well as the encoding of the operations of Iverson's programming language APL in unpublished work of Marisa Venturini Zilli in 1962-64.[37]

An abstract CUCH-machine was described in [Böhm and Dezani, 1972], and had an actual implementation in the 1990s, in unpublished work by Stefano Guerrini. The CUCH machine is of interest for the history of abstract interpreters for the λ-calculus, for it used a leftmost-first reduction strategy instead of the call-by-value order realized by the more widely known SECD-machine of Landin.

But the main influence of Böhm and his students on λ and CL was through discoveries about the pure systems. The presentation of CUCH in [Böhm, 1966] and [Böhm and Gross, 1966] set up a uniform language for formulating technical problems about the syntax of untyped λ and CL, and the solution of such problems dominated much of Böhm's later research activity. We shall discuss these in §7.1.

7 Syntactical developments

The revival of interest in λ and CL that began in the 1960s brought with it some new advances in the study of their syntax.

Work by proof-theorists on Gödel's *Dialectica* paper [Gödel, 1958], stimulated particularly by Georg Kreisel in Stanford University, led to normalization proofs for various typed extensions of λ and CL from 1963 onward. These will be discussed in §8.2 below.

In 1963 Dana Scott gave lectures on λ to an autumn seminar on foundations at Stanford [Scott, 1963]. These were partly motivated by "a certain lack of satisfaction with systems for functions suggested in papers of John McCarthy", with the hope "that the underlying philosophy will ... lead to the development of a general programming language for use on computers" [Scott, 1963, p. 1.2]. Scott's syntactical innovations included a new representation of the natural numbers for which the predecessor combinator was much simpler than those known for Church's numerals,[38] and the following neat and general new undecidability theorem for λ: if two non-empty sets of terms are closed under expansion (i.e. reversed reduction), then the pair cannot be recursively separated, i.e. there is no total recursive function which outputs 1 on the members of one set and 0 on the other; see [Scott, 1963, §9] or [Barendregt, 1981, §6.6] or [Curry *et al.*, 1972, §13B2]. Incidentally, Curry independently proved a similar undecidability theorem, though his was slightly less general than Scott's and was made a few years later (but before seeing Scott's theorem), see [Curry, 1969b, p.10] or [Curry *et al.*, 1972, p.251, footnote 7].

Scott's next major contribution to the theory of λ was on the semantic side in 1969 and radically changed both the subject itself and its more general context, the study of higher-order functions; see §9.1 below.

In 1971 appeared the influential thesis of Henk Barendregt, [Barendregt, 1971]. Part of its motivation was semantical, so we shall discuss it in §9.2 p.46, but its results and methods were syntactical and stimulated much work on syntactical properties of pure λ in the 1970s. Some of this work will be described in §7.2, and many of its pre-1981 results were incorporated into the monograph [Barendregt, 1981], which became the standard reference for the core properties of λ.

[37] Almost 20 years later, [Böhm, 1982] described a combinatory representation of Backus' systems of functional programming, [Backus, 1978], that are in many respects similar to APL.

[38] Scott's numerals were $\mathbf{0} \equiv \mathsf{K}$, $\mathbf{succ} \equiv \lambda uxy.\, yu$, see [Scott, 1963, §3] or [Curry *et al.*, 1972, pp. 260–261]. For them, $\mathbf{pred} \equiv \lambda x.\, x\mathbf{0I}$. Some predecessors known for the Church numerals are given in [Curry *et al.*, 1972, pp. 218, 226]. Other numeral systems have been proposed and used, for example in [Barendregt, 1976, §4.1], in [Wadsworth, 1980], and by Böhm, see below. Numeral systems in general have been discussed, with examples, in [Curry *et al.*, 1972, §§13A1, 13C1] and [Barendregt, 1981, §§6.4, 6.8].

7.1 Contributions from the programming side

In Italy, the work of Böhm on the λ-based language CUCH led him to ask a series of technical questions about the syntax of λ which came to have an important influence on the development of the subject, both through the questions themselves and through the later activities of the students he attracted to his work. The first of these questions, at the end of a talk given in 1964 [Böhm, 1966, p.194], was whether it was possible to discriminate between any two distinct closed normal forms F, G by constructing a λ-term $\Delta_{F,G}$ such that

$$\Delta_{F,G}F =_{\beta\eta} \lambda xy.x, \qquad \Delta_{F,G}G =_{\beta\eta} \lambda xy.y.$$

A few years later the answer "yes" was proved for $\beta\eta$-normal forms by Böhm himself, [Böhm, 1968]; this result is now known as *Böhm's theorem*.[39] Böhm's proof of his theorem was analysed by Reiji Nakajima and by Barendregt independently, see [Nakajima, 1975] and [Barendregt, 1977, §6], and from these analyses came the now well-known concept of *Böhm tree*, as well as the use of this concept in the study of models of λ (see §9.2 below).[40] In [Barendregt, 1981, Ch. 10] there is a careful account of the ideas behind the proof (including the name *Böhming out* for the technique of extracting subtrees of a Böhm tree by means of contexts).

Another conjecture of Böhm's, made in [Böhm, 1968, p.4], was what is now called the *Range theorem*: the *range* of a combinator F is the set of all $\beta\eta$-convertibility classes of closed λ-terms of the form FM, and the theorem states that this set is either infinite or a singleton. This theorem was proved within a year by Myhill and Barendregt independently. A constructive version of the theorem, prompted by a remark of Dirk van Dalen, together with some historical remarks on the various proofs, is in [Barendregt, 1993].

Böhm's aim of using CUCH as a real programming language led him to study various ways of coding data structures as λ-terms. He coded natural numbers as the Church numerals [Böhm, 1966, Appendix II], although he also considered alternative representations, for example

$$\mathbf{n} \quad \equiv \quad \lambda x.\langle \underbrace{x, \ldots, x}_{n \text{ times}} \rangle$$

as described in [Böhm, 1966, Appendix VI]. Vectors $\langle X_1, \ldots, X_n \rangle$ were represented as terms of the form $\lambda x.xX_1 \ldots X_n$ with projections U_i^n defined by $\lambda x_1 \ldots x_n.x_i$, [Böhm, 1966, Appendix IV]. Böhm saw both data and programs as higher-order functions that should be represented by terms in normal form (or by strongly normalizing terms). This led him to the problem of structuring the set of all normal forms algebraically, which became especially important in his work. He wished to find an algebraic structure of normal forms suitable for representing data structures as initial algebras, and over which iterative, primitive recursive and even total recursive functions could be represented by terms in normal form. He explored this representation from 1983 onwards, and reported the results in a long series of works: [Böhm and Berarducci, 1985], [Böhm, 1986; Böhm, 1988b; Böhm, 1988a; Böhm, 1989]. The first of these, with Alessandro Berarducci in Rome, investigated the representation of data structures in second order λ-calculus and, as we shall discuss in §8.3, was very influential in investigations of polymorphic λ-calculi.

The quest for algebraic structure in combinatory logic led also to the study of the monoid structure of combinators (modulo η-conversion), extending an early idea

[39] A well organised proof of Böhm's theorem is in [Krivine, 1990, pp. 68–74]. A fuller proof which has been mechanised in the language CAML is in [Huet, 1993]; it deals with problems about bound variables which are usually ignored but become serious when infinite trees are involved.

[40] Although for terms in normal form the Böhm tree concept had already appeared in [Böhm and Dezani, 1974].

of Church [Church, 1937], and even to combinatory groups through the study of invertible terms. In 1976 his co-worker and former student Mariangiola Dezani[41] in Turin made a characterization of the invertible normalizable terms of $\lambda\beta\eta$-calculus, [Dezani, 1976], and her result was extended to all terms by Klop and Jan Bergstra, [Bergstra and Klop, 1980]. With Francoise Ermine, Dezani then gave a description of maximal monoids of normal forms, using an early version of intersection types, [Dezani and Ermine, 1982]. Monoids of combinators were studied further in [Böhm and Dezani, 1989]. (A detailed account of the results in this area until 1980 is in [Barendregt, 1981, Ch. 21].)

Taking a similar algebraic approach, around 1977 Böhm proposed a simple reinterpretation of combinatory logic by regarding combinators as generalized numerals [Böhm, 1979; Böhm, 1980]; this was suggested by the fact that exponentiation $\mathbf{m^n}$ of Church numerals $\mathbf{m}, \mathbf{n}$ is just $\mathbf{nm}$. He regarded combinators as a structure with sum, product and exponentiation with the usual arithmetical axioms, plus a non-arithmetical constant for pairing.

James H. Morris at the Massachusetts Institute of Technology was another influential contributor who, like Böhm, was motivated by an interest in programming languages; his thesis [Morris, 1968] acknowledged an intellectual debt to Landin, although most of the results in it concerned the syntax of untyped and simply typed λ-calculus.

That thesis contained the first formulation of "pure" simply typed λ in isolation without the extra logical axioms and rules that the systems of Church and Curry had contained. Morris' formulation was what is nowadays called "Curry-style"; types were assigned to untyped pure λ-terms. In [Morris, 1968, p.107] he proved weak normalization by a method essentially the same as in Turing's unpublished note, see §8.2 below, and in [Morris, 1968, pp.100–105] he described and verified a test for typability, see §8.5 below.

Morris' thesis also contained perhaps the first application of Böhm's Theorem, in a result which, although syntactical, later played a significant role in the model theory of λ. Morris suggested looking at a λ-term M as a partial function consisting of those pairs $\langle\langle M_1, \ldots, M_n\rangle, N\rangle$ such that reduction starting at $MM_1 \ldots M_n$ halts with the normal form N. The problem with this idea was that $\beta\eta$-conversion does not coincide with equality of partial functions, so Morris introduced the following preorder on terms [Morris, 1968, p.50]:

> Given λ-terms M, N: M *extends* N if, for all contexts $C[\ ]$, if $C[M]$ has a normal form then $C[N]$ has the same normal form.

The equivalence relation on terms generated by the extension preorder came later to be called *contextual equivalence*. Morris proved it to be, essentially, the largest congruence on (pure) λ-terms that non-trivially extends $\beta\eta$-convertibility, by applying the newly-published Böhm Theorem [Morris, 1968, p.53, Cor. 5].

A point worth noting in Morris' definition of the extension preorder was the use of *contexts*: these became an ubiquitous tool for the definition of equivalences on both untyped and typed terms (see §9.2 and §10 later). Morris also proved that the fixed-point combinator $\mathbf{Y}$ commonly attributed to Curry yields the least fixed point under the extension preorder [Morris, 1968, Ch. III, §C, Cor. 7(a)], by a complicated analysis of reductions. This result is used to justify, by way of example, a form of McCarthy's principle of Recursion Induction.

Near the end of his thesis, Morris argued for the usefulness of allowing circular type-expressions in simply typed λ-calculus, like the solution of the type equation $\alpha = \alpha \to \beta$, remarking that "we have only a limited intuition about what kind of an object a circular type expression might denote (possibly, a set of functions, some

[41] Married name Mariangiola Dezani Ciancaglini.

of which have themselves as arguments and values)", [Morris, 1968, p.123]. The answer to this puzzle was to be found less than one year later by Dana Scott with his set-theoretic model for the λ-calculus, see §9.1 below.

7.2 Theory of reductions

Ever since the original proof of the confluence of $\lambda\beta$-reduction in [Church and Rosser, 1936], a general feeling had persisted in the logic community that a shorter proof ought to exist. The work on abstract confluence proofs described in §5.2 did not help, as it was aimed mainly at generality, not at a short proof for $\lambda\beta$ in particular.

For CL, in contrast, the first confluence proof was accepted as reasonably simple; its key idea was to count the simultaneous contraction of a set of non-overlapping redexes as a single unit step, and confluence of sequences of these unit steps was easy to prove, [Rosser, 1935, p.144, Thm. T12].

Then in 1965 William Tait presented a short confluence proof for CL to a seminar on λ organized by Scott and McCarthy at Stanford. Its key was a very neat definition of a unit-step reduction by induction on term-structure. Tait's units were later seen to be essentially the same as Rosser's, but his inductive definition was much more direct. Further, it could be adapted to $\lambda\beta$. (This possibility was noted at the seminar in 1965, see [Tait, 2003, p.755 footnote]). Tait did not publish his method directly, but in the autumn of 1968 he showed his CL proof to Per Martin-Löf, who then adapted it to $\lambda\beta$ in the course of his work on type theory and included the $\lambda\beta$ proof in his manuscript [Martin-Löf, 1971b, pp.8–11, §2.5].

Martin-Löf's $\lambda\beta$-adaptation of Tait's proof was quickly appreciated by other workers in the subject, and appeared in [Barendregt, 1971, Appendix II], [Stenlund, 1972, Ch. 2] and [Hindley *et al.*, 1972, Appendix 1], as well as in a report by Martin-Löf himself, [Martin-Löf, 1972b, §2.4.3].[42]

In λ, each unit step defined by Tait's structural-induction method turned out to be a minimal-first development of a set of redexes (not necessarily disjoint). Curry had introduced such developments in [Curry and Feys, 1958, p.126], but had used them only indirectly; Hindley had used them extensively in his thesis, [Hindley, 1969a, p.547,"MCD"], but only in a very abstract setting. They are now usually called *parallel reductions*, following Masako Takahashi. In [Takahashi, 1989] the Tait-Martin-Löf proof was further refined, and the method of dividing reductions into these unit steps was also applied to simplify proofs of other main theorems on reductions in λ.

Tait's structural-induction method is now the standard way to prove confluence in λ and CL. However, some other proofs give extra insights into reductions that this method does not, see for example the analysis in [Barendregt, 1981, Chs. 3, 11–12].

Besides confluence, major themes in the study of $\lambda\beta$-reductions have been given by theorems on *finiteness of developments*, *standardisation* and the *equivalence of complete developments*, all depending either directly or indirectly on the concept of *residual*. The first two of these theorems were proved for λI-terms in [Church and Rosser, 1936, Lemma 1], and the second was proved for CL in [Rosser, 1935, p.142, Thm. T9]; but their analogues for the full λK-calculus are more subtle and did not come until much later.

The finiteness-of-developments theorem for λK was first proved by Schroer in his unpublished thesis [Schroer, 1965, Part I, Thm. 6.20]. It was re-proved later using different methods: by Martin Hyland in 1973 [Hyland, 1975, Thm. 3.6], by Hindley

[42]The earlier manuscript [Martin-Löf, 1971b] was not published, as the system described in it was inconsistent from the standpoint of the propositions-as-types correspondence, see §8.1.4 below; but its confluence proof for $\lambda\beta$ was not faulty.

in 1976 [Hindley, 1978a, Thm. 1], and by Barendregt, Bergstra, Klop and Volken in 1976 [Barendregt *et al.*, 1976, pp.14–17]. The latter proof used a particularly neat labelling technique and a strong normalization theorem for labelled reductions, see [Barendregt, 1981, §11.2].

Labelling techniques to follow components of terms down through reductions were first used in [Hyland, 1976, §2] and [Wadsworth, 1978, §4], in proofs of the limit theorem for approximants, see §9.2 below. In [Wadsworth, 1976, p.508] the idea was attributed by Wadsworth to a 1972 communication from Hyland. It was extended by Jean-Jacques Lévy in 1974 [Lévy, 1976, pp. 106–114] and in [Lévy, 1978, Ch. II], and applied to prove the completeness of "inside-out" reductions and the other main theorems on reductions. A good general account of labelling is in [Barendregt, 1981, Ch. 14].

The standardization theorem for λK first appeared in [Curry and Feys, 1958, §4E1], though with a proof that depended on the finiteness of leftmost-first developments. In 1975 Curry's proof was much simplified by Gerd Mitschke, [Mitschke, 1979]. A consequence of standardization is the cofinality of leftmost reductions, i.e. that the leftmost reduction of a term is finite if and only if the term has a normal form. In the early 1970s Barendregt pointed out that the reductions most often used in practice were quasi-leftmost, not leftmost, and proved the corresponding cofinality theorem for these reductions too [Barendregt, 1981, Thm. 13.2.6].[43] In his 1976 report with Bergstra, Klop and Volken the more general concept of *reduction strategy* was introduced and studied, see [Barendregt *et al.*, 1976, pp.33–43] or [Barendregt, 1981, Ch. 13].

The basic equivalence theorem for complete developments in λK was first stated and proved in [Curry and Feys, 1958, pp.113–130]. In 1978 it was improved by Lévy to a stronger equivalence, see [Lévy, 1978, pp. 37, 65] or [Barendregt, 1981, §§12.2.1, 12.2.5]. Lévy's equivalence theorem led to a new view of β-reductions, embodied in the thesis [Klop, 1980]. The transformations involved in proofs of confluence and standardization were seen as conversions of *reduction diagrams*, and the reasonably tidy underlying structure that had been sought for the theory of β-reduction for the previous forty years was at last achieved. (An account is in [Barendregt, 1981, Chs. 11–14].)

For $\lambda\beta\eta$, confluence and the postponement of η-steps were known since Curry's work in the 1950s, as was mentioned in §5.3, but no standardization theorem had been proved analogous to that for β, nor had any cofinality theorem been proved for leftmost reductions, despite attempts. The first successful proofs of these theorems were made in [Klop, 1980, Ch. IV].

Returning to $\lambda\beta$: problems involved in making reduction more efficient began to attract attention in the 1970s, stimulated by programming needs. One such problem was that of finding a reduction of a term to its normal form that was optimal in some reasonable sense. For CL this problem was studied in 1972 by John Staples, though his work was not published until 9 years later, [Staples, 1981]. In 1974 for a system of recursive programs, Jean Vuillemin formulated a "delay rule" for reducing first-order terms with function symbols defined by a recursive program, [Vuillemin, 1974a, §3]. This rule in Vuillemin's system could be proved optimal and easily implementable, but the problem of optimality turned out to be more difficult in the case of λ-calculus. However, Lévy extended Vuillemin's idea to λ in his thesis [Lévy, 1978], by an in-depth analysis of the lattice of β-reductions starting at a term, and by exploiting his labelling technique to define certain families of redexes that can be shared through reductions.[44] Lévy also proved the completeness of *inside-out* β-

[43] A reduction is quasi-leftmost iff, for every non-leftmost step, some later step is leftmost.

[44] A suitable data structure for implementing the sharing required for optimal reductions was described in 1990, independently in [Lamping, 1990] and [Kathail, 1990]. Then in [Gonthier *et al.*, 1992] and [Asperti, 1995] the technique was related to Girard's *geometry of interaction* [Girard,

reductions (almost the opposite of leftmost reductions), which had been conjectured in [Welch, 1975], see [Lévy, 1976].

Substitution was another topic which aroused interest from a computing point of view. Its correct definition had been settled years ago, but its efficient implementation now became an important problem; for example in LISP in the 1950s some of the details had been over-simplified and had had to be corrected, see §6.1 above. With the implementation problem in mind, in 1978 Gyorgy Révész proposed an interesting new alternative to β-reduction. It generated the usual equivalence relation $=_\beta$, was confluent, had the same normal forms as the usual β-reduction, and broke substitutions into smaller steps; see [Révész, 1978] or [Révész, 1985].

Another approach to the problem of implementing substitution efficiently was to extend the λ-language by introducing an *explicit substitution* operator with suitable reduction rules. This idea, which had occurred independently to Mitschke and Barry Rosen for use in confluence proofs for $\lambda\beta$-reduction ([Mitschke, 1973, pp. 150–151], [Rosen, 1973, §9]), led in the 1990s to several formal systems which are still being developed; a few examples of papers on these are [Komori, 1990], [Abadi *et al.*, 1991], [Kamareddine and Nederpelt, 1993], [Curien *et al.*, 1996], [Benaissa *et al.*, 1996], [Kamareddine and Rios, 1997], [Bloo and Geuvers, 1999], and [Kamareddine and Rios, 2000].

A special case of substitution was the problem of changing bound variables efficiently. In 1972 de Bruijn described a formalism which avoided this problem completely, by numbering all λs in a term and replacing bound variables by number-indices to show which λ bound them, [Bruijn, 1972]. De Bruijn's formalism was used in the compiler of the programming language ML in the 1980s, and has been used in many explicit substitution systems.

8 Types

8.1 The general development of type theories

In the type theory of Russell and Whitehead's *Principia Mathematica* and its simplifications by Chwistek and Ramsey [Russell and Whitehead, 1913; Chwistek, 1922; Ramsey, 1926], there was little agreement as to what kind of entities types should be. Already in [Russell, 1903, §497] we find at least two ways in which types can be intended:

(1) *Types as ranges of significance of propositional functions:* given a propositional function $\varphi(x)$, there is a class of objects, the *type* of x, such that φ has a value whenever it is applied to a member of this type.

(2) *Types as sets:* individuals form the lowest *type* of objects, then there are sets of individuals, sets of sets of individuals and so on, ad infinitum.

The first interpretation of types is related to the theory of *grammatical categories*. The view of types as *sets* underlies, of course, the early developments in set theory, cf. [Quine, 1963, Ch. XI], but most notably from a λ-viewpoint it underlies the definition of the hierarchy of simple types in [Church, 1940] and much of the research on set-theoretical models of type theories.

The later grafting of the type-theoretic tradition onto λ and CL, and the mathematical study of the models of the resulting theories, contributed to the discovery of new ways in which types could be interpreted. A new unifying paradigm that became (and still is) very influential in research in type theory was introduced by Lawvere in 1969 and Lambek in the 1970s [Lawvere, 1969a; Lawvere, 1969b; Lambek, 1974; Lambek, 1980b; Lambek, 1980a]:

1989a].

(3) *Types as objects in a category*, especially a *cartesian closed category*, whose definition provides a general framework for a formulation of the set-theoretical bijection between functions of several arguments and unary functions with functions as values.

In addition, research in proof-theory and the semantics of intuitionistic mathematics showed that it was in many cases fruitful to view

(4) *Types as propositions:* in either λ or CL, terms of a given type play the role of codes for proofs of (the proposition coded by) that type.

We shall now follow the historical development of each of these threads in turn.[45]

8.1.1 Types as grammatical categories

The theory of categorial grammars began in about 1900 with the attempt of Edmund Husserl to develop a theory of pure, *a priori* laws that underlie the composition of meanings, [Husserl, 1922, Fourth Investigation]. Through the teaching of Kazimierz Twardowski, Husserl's ideas influenced the Lwów school of logicians, and in particular the design of the logical systems of Leśniewski [Leśniewski, 1929], where they replaced the logical types of Russell. Ajdukiewicz was the first to give a self-contained formalization of a system of categorical grammar, [Ajdukiewicz, 1935]. There, it is assumed that to each word of a language can be assigned one or more *indices*, that are symbolic representations of the grammatical categories to which the word belongs. Starting from indices n for noun phrases and s for propositions, one can obtain infinitely many new indices α/β for phrases that yield a phrase of index α whenever prefixed to a phrase of index β. The concatenation of two words with respective indices α and β has index $\alpha\beta$, and one is allowed to replace each subsequence of the form $(\alpha/\beta)\beta$ by α. This basic system can be extended by forming the index $\alpha\backslash\beta$ for phrases that yield a phrase of index β whenever suffixed to a phrase of index α, with a rule that allows a subsequence of the form $\alpha(\alpha\backslash\beta)$ to be replaced by β, [Bar-Hillel, 1953; Bar-Hillel, 1959; Lambek, 1958; Lambek, 1961]. Writing $x \to y$ to mean that every expression of category x has also category y, one can prove theorems like, for example, that $xy \to z$ implies $y \to x\backslash z$. Lambek introduced a Gentzen-like calculus for deriving theorems of this form, and proved its decidability through a cut-elimination theorem, [Lambek, 1958].

Curry, because of his interest in the formal study of natural languages, knew about the theory of categorial grammar,[46] and noticed that his theory of functionality (see §4.2) had a grammatical interpretation in which types could be regarded as grammatical categories. This was in harmony with his early ideas on the theory of functionality, which was intended to be a tool for drawing categorial distinctions between the terms of systems of illative combinatory logic, with the idea that "a theory of types is essentially a device for saying that certain combinations are not propositions", see [Curry, 1980, §9]. The grammatical interpretation of functionality was described in [Curry, 1961];[47] it regarded the combination $\mathbf{F}\alpha\beta$ as the category of those functors that take an argument of category α and give a result of category β, in the context of a formal system whose only operation is application. (For example, if nouns and noun-phrases have category N, then adjectives have category $\mathbf{F}NN$, adverbs (as modifiers of adjectives) have category $\mathbf{F}(\mathbf{F}NN)(\mathbf{F}NN)$, and the suffix "-ly" (which changes adjectives to adverbs) has category $\mathbf{F}(\mathbf{F}NN)(\mathbf{F}(\mathbf{F}NN)(\mathbf{F}NN))$.)

The possibility of describing grammatical categories by means of types has also played a role in formalizing the syntax of logical theories within typed λ-calculi. This

[45] We shall not try to survey all of type-theory, but only the parts that are most relevant to λ and CL. In particular, we shall not discuss the type-concept implicit in ZF set theory.

[46] In particular, about [Ajdukiewicz, 1935], see the remark in [Curry and Feys, 1958, §8S1].

[47] Based on [Curry, 1948], summarized briefly in [Curry and Feys, 1958, §8S2, pp.274–275].

has happened, for example, with the theory of expressions that underlies Martin-Löf's intuitionistic theory of types (described for the first time by Martin-Löf in his lecture at the Brouwer Centenary in 1981; see [Nordström *et al.*, 1990, p.13]) and the more recent Edinburgh Logical Framework [Harper *et al.*, 1993]. In a broader perspective, the same connection has been exploited in mathematical linguistics in the area known as *Montague grammar*, [Montague, 1974].

8.1.2 Types as sets

There was a clear set-theoretic intuition behind the hierarchy of simple types in [Church, 1940]: Church used a type-expression $\beta\alpha$ to denote a set of functions from the set (denoted by) α to the set (denoted by) β. This intuition can readily be extended to yield a notion of model for the simply typed λ-calculus as a set-theoretic structure $\mathfrak{D}$ consisting of a family of sets $\{D^\alpha\}_{\alpha\in\mathbb{T}}$, where $\mathbb{T}$ is the set of all type-expressions of the system, such that $D^{\beta\alpha}$ is the set of all functions from D^α to D^β, with a family of application-functions $\cdot_{\alpha,\beta} : (D^{\beta\alpha} \times D^\alpha) \to D^\beta$, for $\alpha, \beta \in \mathbb{T}$, that correspond to the application of a function of type $\beta\alpha$ to an argument of type α with result of type β. This notion is essentially that of *standard model* introduced by Leon Henkin in [Henkin, 1950, pp.83–85], who also considered *general models* where $D^{\beta\alpha}$ is only assumed to be *some* class of functions from D^α to D^β.[48]

The interpretation of types as sets led to developments that became important also for later extensions of this rather limited framework. One of these was the notion of *logical relation* between models $\mathfrak{D}$ and $\mathfrak{E}$, defined as a family of relations $R^\alpha \subseteq D^\alpha \times E^\alpha$ indexed over types, such that[49]

$$R^{\alpha\to\beta}(f,g) \iff \big((\forall x \in D^\alpha, y \in E^\alpha)\ R^\alpha(x,y) \Rightarrow R^\beta(f \cdot_{\alpha,\beta} x,\ g \cdot_{\alpha,\beta} y)\big)\,.$$

An early instance of the use of logical relations was the proof of completeness of typed $\beta\eta$-conversion with respect to any standard model (over an infinite set of individuals) of simply typed λ-calculus, given by Harvey Friedman in 1970 and published in [Friedman, 1975]. This result exploited the properties of *partial homomorphisms* of models, where a partial homomorphism from $\mathfrak{D}$ to $\mathfrak{E}$ is defined as a family of partial surjections $f^\alpha : D^\alpha \rightharpoonup E^\alpha$ from $\mathfrak{D}$ to $\mathfrak{E}$ for each type α, with the property that $f^\beta(x \cdot_{\alpha,\beta} y) \simeq f^{\alpha\to\beta}(x) \cdot_{\alpha,\beta} f^\alpha(y)$. The graph of a partial homomorphism is then a logical relation from $\mathfrak{D}$ to $\mathfrak{E}$.

Later research showed that logical relations afford a convenient framework for formulating and proving the main properties of simply typed λ-calculi. On the syntactic side, the various forms of the notion of "computability" used in Tait-style normalization proofs can be described as logical relations, as well as some other properties of reduction like confluence and η-postponement, [Statman, 1985]. On the semantic side, the extensional collapse of an applicative structure uses logical partial equivalence relations (a result from 1971 of Jeff Zucker, reported in [Troelstra, 1973, §2.4.5]; see also [Statman, 1983]). Special classes of logical relations were used in the characterization of definability in D_∞ models of the type-free λ-calculus and in standard models of the simply-typed λ-calculus by [Plotkin, 1973] (where the name 'logical relation', attributed to Mike Gordon, seems to have first appeared). See also [Plotkin, 1980].

The research on higher-order notions of computability in the 1950s produced a host of type structures or even models for simply typed λ in which function types are interpreted by some set of *computable* function(al)s.[50] Especially important from our standpoint is the search for a suitable notion of *partial* computable functional,

[48] Church's system had an extensionality axiom; for systems without this, the definition of model would be more complicated than in [Henkin, 1950]; see, for example, [Barendregt, 1981, 2nd edn., Ch.5] or [Hindley and Seldin, 1986, §11A].

[49] From now on we write the nowadays-common notation $\alpha \to \beta$ instead of Church's $\beta\alpha$.

[50] For a thorough historical account of this field, highlighting its many intersections with our

leading to the partial *monotone* functionals studied in Richard Platek's Stanford PhD thesis [Platek, 1966] (see [Longley, 2001, §4.1.1] and [Moldestad, 1977]).

The idea of an ordering on functionals, stressed by Platek, together with the mathematical theory of computation of McCarthy [McCarthy, 1963a], are at the basis of the theory of computable functions of higher type described in a privately circulated typescript written in October 1969 by Dana Scott [Scott, 1969e] (see also [Scott, 1969d]). Scott proposed there a typed alternative to the (type-free) programming languages introduced in [Böhm, 1966; Böhm and Gross, 1966] and [Landin, 1966b], which amounted to a simply typed λ-calculus with ground types ι for integers and o for truth values, and containing a functional $\mathbf{Y} : (\tau \to \tau) \to \tau$, for all types τ, for forming (least) fixed-points of functions of type $\tau \to \tau$ (suggested by Platek's thesis), and McCarthy's conditional operator. In addition, there were constants Ω_ι and Ω_o denoting the "undefined" integer and truth value, respectively, so that partial numerical functions could be described as continuous total functions that may take value Ω_ι at some argument. The intended model for this system was based on the interpretation of types as *domains*, i.e. partially ordered sets D with a least element $\bot_D$ and least upper bounds of increasing chains. Functions over domains were required to be *continuous*: these are the functions that preserve least upper bounds of chains. The judgements of the system exploited the semantical ordering between values: notably, there was a rule of induction of the form:

$$\frac{\Phi \ \vdash \ g(\Omega) \leq h(\Omega) \qquad \Phi, g(x) \leq h(x) \ \vdash \ g(f(x)) \leq h(f(x))}{\Phi \ \vdash \ g(\mathbf{Y}(f)) \leq h(\mathbf{Y}(f))}.$$

This rule was considered by Scott to be the main advantage of the system (Scott [Scott, 1969e, §3]), as compared e.g. to the principle of recursion induction arising from the work of McCarthy [McCarthy, 1963a, §8].

The resulting system was the basis of the LCF project (LCF stands for "Logic for Computable Functions", the name given to Scott's system by Robin Milner) developed from 1972 onward by a group at Stanford University led by Milner. The functional programming language ML in [Gordon *et al.*, 1979] was originally developed in order to check proofs in a version of LCF, see [Gordon, 2000]. Furthermore, the typed λ-calculus that constitutes its logic-free part became the language PCF ("Programming language for Computable Functions", [Plotkin, 1977]), that is widely used as the prototype functional programming language for many theoretical purposes, in particular the study of the important property of full abstraction (see later, §10).

8.1.3 Types as objects

The connection between type theory and category theory that underlies a large amount of current research shows up along the lines set by the works of Joachim Lambek and F. William Lawvere.

The latter realized in the early 1960s that certain basic mathematical transformations of a general nature in what he later called *categorical dynamics* [Lawvere, 1979] were in fact also basic structural ingredients of several other categories of foundational interest. In fact, these transformations were those used later to define the notion of cartesian closed category [Eilenberg and Kelly, 1966].

Though Lambek did not stress the category theoretic content of his results on the syntactic calculus in [Lambek, 1958; Lambek, 1961], we mention his ideas here because they represent an alternative route to the applications of cartesian closed categories to λ-calculus via deductive systems and a related notion of functional completeness, inspired by the abstraction algorithms of combinatory logic

present subject, we refer the reader to [Longley, 2001]. Longley (ibid., §3.3.1) also points out [Kreisel, 1959] as the source of many ideas that have played a key role in the later development of λ.

(§3 above). Furthermore, he later explicitly described formally the connection of the syntactic calculus with closed categories [Lambek, 1968; Lambek, 1988; Lambek, 1995], providing, with hindsight, the unifying framework for his work and that of Lawvere.

Lambek used the system of categorial grammar described in §8.1.1 as a formal tool in the construction of canonical mappings between functors of bimodules over associative rings.[51] There, the following correspondence between the constructors of syntactic types and operation on bimodules was exploited:

$$\begin{aligned} AB &=_{\text{def}} A \otimes_S B \\ C/B &=_{\text{def}} \text{Hom}_T(B, C) \\ A\backslash C &=_{\text{def}} \text{Hom}_R(A, C) \end{aligned}$$

for an R-S-bimodule A, an S-T-bimodule B and an R-T-bimodule C. There is an evident closed category structure in this example, in the form of what was called a *residuated category* in [Lambek, 1968]. In fact, already in his first published paper on the syntactic calculus, [Lambek, 1958, §10], Lambek noted the formal similarity between the operation of (right and left) division in the calculus of syntactic types and residuation in lattice theory, the operation normally used to interpret implication in algebraic logic.

The notion of cartesian closed category emerged gradually, basically through the work of Lawvere who in 1959 became aware of the central role of exponential objects B^A, characterized by the isomorphisms $C \times A \to B \;\cong\; C \to B^A$.[52] An early example of this transformation came from the description of the motion of a material body seen as a mapping $T \times M \to E$ that gives the position of each of its particles at each time instant, or equivalently, as its trajectory $T \to E^M$ through time, or as the path $M \to E^T$ followed by each of its particles, [Lawvere, 1976, p.123].

There are other instances of exponentiation that played an important role in the development of the notion of cartesian closed category. The introductory chapter of Lawvere's PhD thesis [Lawvere, 1963] contains the description of the category of categories, where the exponential of two objects (namely, categories) $\mathcal{A}$ and $\mathcal{B}$ is the category of all functors from $\mathcal{A}$ to $\mathcal{B}$.[53] Later, in the years 1963–64, Lawvere investigated an elementary[54] axiom system for the category of sets in order to develop a basic framework for analysis, [Lawvere, 1964]: the exponential B^A of two objects A and B is here the set of all functions with domain A and codomain B. The relation of exponentiation to λ-abstraction is mentioned for the first time in this work, but no formal detail is given. During his stay in Berkeley in 1961–62, Lawvere had noted that Heyting algebras are cartesian closed categories: more generally, a preordered set with finite products has a monoidal structure which is closed if and only if the preorder is residuated, that is, iff it also satisfies the equivalence:

$$p \wedge q \;\leq\; r \;\text{ if and only if }\; p \;\leq\; q \Rightarrow r,$$

[51] This application seems to have been one main mathematical motivation for the development of the syntactic calculus. It originated from work carried out by Lambek jointly with G. D. Findlay in 1955, described in the manuscript [Lambek and Findlay, 1955] (unpublished, but summarized in [Lambek, 1961, Appendix II]). According to [Lambek, 1988, p.297], the relation of this formalism to natural language was already clear to him at an early stage, but it was only later that, through a bibliographic search, he discovered the paper [Bar-Hillel, 1953] on categorial grammars.

[52] Intuitively, these may be thought of as the tranformation of a two-argument mapping $f : C \times A \to B$ into the one-argument mapping $\hat{f} : C \to B^A$ which outputs a function as value. See §3, currying.

[53] This was later published as [Lawvere, 1966].

[54] In this context, "elementary" refers to the fact that the axioms are sentences of a multi-sorted first-order language—for example, with one sort for maps and another for objects.

where $\Rightarrow$ is the residuation operation. (This example appears in [Eilenberg and Kelly, 1966, Chapter IV, §4]). This is one of the possible ways of arriving at what was later called the "propositions as types" correspondence (§8.1.4).

A general notion of category $\mathbb{C}$ with exponentiation, that took the name of *cartesian closed category* after discussions with Eilenberg and Kelly [Eilenberg and Kelly, 1966, Chapter IV, §2], was described by Lawvere by the following adjoint situations [Lawvere, 1969a]:

- a terminal object, given by a functor $1 : \mathbf{1} \to \mathbb{C}$ right adjoint to the unique functor from $\mathbb{C}$ to the category $\mathbf{1}$ with one object;
- a bifunctor $\times : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ (cartesian product) right adjoint to the diagonal functor $\Delta : \mathbb{C} \to \mathbb{C} \times \mathbb{C}$ that transforms an object C of $\mathbb{C}$ into the object $\langle C, C \rangle$ of $\mathbb{C} \times \mathbb{C}$; and finally
- a right adjoint $(\cdot)^A : \mathbb{C} \to \mathbb{C}$ to the functor $A \times (\cdot) : \mathbb{C} \to \mathbb{C}$, for each object A of $\mathbb{C}$. The counit of each adjunction $A \times (\cdot) \dashv (\cdot)^A$ is a family of morphisms indexed by objects B of $\mathbb{C}$, $\varepsilon_B : B^A \times A \to B$, called *evaluation*.

The explicit connection between cartesian closed categories and type theory was highlighted in Lawvere's unpublished manuscript "Category-valued higher-order logic", distributed in 1967 to the Los Angeles Set Theory Symposium, whose contents were later published as [Lawvere, 1969a; Lawvere, 1969b; Lawvere, 1970a]. The central categorical notion in these papers is that of a *hyperdoctrine*, and was intended to capture some of the examples that would later lead to the notion of *elementary topos* [Lawvere, 1970b]. A hyperdoctrine is defined by a cartesian closed category $\mathbb{T}$ of *types*, whose morphisms are regarded as *terms*. For each type X in $\mathbb{T}$, there is a cartesian closed category $P(X)$ of *attributes of type* X, whose morphisms are called *deductions over* X. The terminal object of $P(X)$ is regarded as the identically true attribute of type X, whereas cartesian product and exponentiation represent conjunction and implication of attributes, respectively, evaluation being *modus ponens*. For every term $f : X \to Y$ in $\mathbb{T}$, there are adjoint situations

$$\sum_f \quad \dashv \quad f \cdot (\quad) \quad \dashv \quad \prod_f$$

that correspond to existential quantification, substitution along f, and universal quantification, respectively.

Examples of hyperdoctrines include, of course, the syntactical hyperdoctrine whose types are the type symbols of a formal type theory and whose attributes of type X are the formulas whose free variables have types corresponding to X. More interesting from the present point of view (see also later, §8.1.4), is an example suggested by the work of H. Läuchli [Läuchli, 1970], in which a complete semantics for intuitionistic predicate calculus is described in terms of a set-theoretic notion of construction inspired by Kleene's notion of realizability and the theories of constructions of Goodman, Kreisel, Troelstra and Scott.[55] In this example $\mathbb{T}$ is the category **Set** of sets, and for any set X seen as a type, the category of attributes of type X is $\mathbf{Set}/X$, so that each attribute φ of type X may be regarded as an X-indexed family of sets, whose x-th coordinate is written as $x \cdot \varphi$. A noteworthy

[55] [Läuchli, 1970] appeared in the proceedings of a conference held at Buffalo in 1968, but an earlier manuscript was circulated already in the Spring of 1967 and was discussed by Lawvere and Dana Scott in San Francisco in the same year. See also [Lawvere, 1996], from which it appears that his manuscript "Category-valued higher-order logic" was discussed also at the symposium on Automatic Demonstration, where the two fundamental papers on the interpretation of proofs as typed λ-terms, [Scott, 1970a] and [Bruijn, 1970], were presented. We shall discuss later, in §8.1.4, the relation between the works of Lawvere and Scott.

feature of this hyperdoctrine is the interpretation of the logical connectives in the category of attributes of type X, for example [Lawvere, 1970a]:

- $\varphi \supset \psi$ is a family of sets whose x-th coordinate is $(x \cdot \psi)^{(x \cdot \varphi)}$. Unfolding the definitions, this means that a proof over X of $\varphi \supset \psi$ is a function which, for every $x \in X$, assigns a proof of $x \cdot \psi$ to a proof of $x \cdot \varphi$;
- $y \cdot \sum_f(\varphi) = \sum_{f(x)=y} x \cdot \varphi$. This means that a proof over Y of $\sum_f(\varphi)$ is a function assigning to every $y \in Y$ an ordered pair $\langle x, d\rangle$ where $f(x) = y$ and d is a proof of $x \cdot \varphi$.

In the meantime, in a series of papers on "deductive systems and categories", Lambek was exploiting the description of categories by means of equational theories over what he called *deductive systems* [Lambek, 1968; Lambek, 1969; Lambek, 1972]. These are essentially (oriented multi)graphs with an edge $1_A : A \to A$ for every node A and an edge $g \circ f : A \to C$ for every pair of edges $f : A \to B$ and $g : B \to C$. In the case of cartesian closed categories, the right kind of deductive system has the following additional structure [Lambek, 1974; Lambek, 1980a]:

- a node T with an edge $0_A : A \to T$ for each node A;
- a node $A \times B$ for every pair of nodes A, B, with edges $\pi_{A,B} : A \times B \to A$ and $\pi'_{A,B} : A \times B \to B$ and $\langle f, g\rangle : C \to A \times B$ for every pair of edges $f : C \to A$ and $g : C \to B$;
- a node B^A for every pair of nodes A, B, and edges $\varepsilon_{B,A} : B^A \times A \to B$ and $\hat{h} : C \to B^A$ for every edge $h : C \times A \to B$.

Observe that a deductive system as described here is neutral as to the interpretation of its nodes and edges as formulas and proofs of the positive intuitionistic propositional calculus or as objects and morphisms of a cartesian closed category, respectively. Therefore, equations between edges of a deductive system can be interpreted either as describing a notion of equivalence of proofs for intuitionistic sequents[56] or as the equational presentation of a cartesian closed category.

The equational presentation of cartesian closed categories allowed Lambek to give a smooth proof of their *functional completeness* [Lambek, 1972; Lambek, 1974], thus making fully explicit the connection with typed λ-calculus already hinted at by Lawvere. If $\mathcal{D}$ is a deductive system, then an assumption of a formula A may be regarded as an *indeterminate*, i.e. as a new edge $x : T \to A$. The edges $\varphi(x) : B \to C$ of a deductive system $\mathcal{D}[x]$ obtained by adding to $\mathcal{D}$ the indeterminate x can be interpreted either as proofs of an intuitionistic sequent $B \to C$ depending on the assumption x, or as polynomials in a cartesian closed category. The following property, proved in [Lambek, 1974, Th. 3.6], subsumes therefore both the deduction theorem of the positive intuitionistic propositional calculus and the functional completeness of a simply typed λ-calculus with finite products (and surjective pairing):

> With every $\varphi(x) : B \to C$ in $\mathcal{D}[x : T \to A]$ there is associated in $\mathcal{D}$ an $f : A \times B \to C$ such that
>
> $$f \circ \langle x \circ 0_B, 1_B\rangle \quad =_x \quad \varphi(x) : B \to C,$$
>
> where $=_x$ is the extension of the equivalence of edges of $\mathcal{D}$ to $\mathcal{D}[x]$.

[56] The theme of equivalence of proofs is a recurring one in Lambek's works: it was already a motivation for his study of the syntactic calculus, leading to a technique for proving coherence theorems in category theory [Kelly and Mac Lane, 1972] based on [Lambek, 1968], and also opened the way for the categorical analysis of proofs pursued in [Szabo, 1974; Szabo, 1978] and [Mann, 1975].

Functional abstraction $\lambda x\colon A\,.\,\varphi(x)$ is then just defined to be $\widehat{\psi} : B \to C^A$, where $\psi = f \circ \langle \pi'_{B,A}, \pi_{B,A} \rangle$. One of the outcomes of the categorical formalization of type theories consisted in providing a way of speaking of types without necessarily assuming that they are sets in a strict sense, because categories with a rich enough structure have an internal logic that still allows one to reproduce set-theoretical arguments to a significant extent. This was exploited systematically by Lawvere in his work on elementary topoi, regarded as universes of "variable sets", [Lawvere, 1970b; Lawvere, 1972; Lawvere, 1976].

This insight set the stage for the most advanced developments in domain theory, especially the idea of synthetic domain theory (§10 below) as a comprehensive framework unifying constructive mathematics and computation theory.

8.1.4 Types as propositions

The interpretation of types as objects of a cartesian closed category, and the observation that such a category corresponds precisely to positive intuitionistic propositional calculus, is only one way of arriving at the analogy between types and formulas that is nowadays called the "propositions as types" correspondence. As mentioned in §4.2, the first published hints of this correspondence date back to [Curry, 1934a] and [Curry, 1934b], and it was first described explicitly in [Curry and Feys, 1958, §9E], as an isomorphism between the provable formulae of intuitionistic implicational logic and the type-expressions assignable to closed terms.

(In the early 1950s, Carew Meredith had observed that proofs in this logic corresponded to composite combinators, and that certain fragments of this logic corresponded to restricted sets of terms, but his observations were not published till later, see §7 of *Notes on the axiomatics of the propositional calculus* by C. A. Meredith and A. N. Prior, *Notre Dame Journal of Formal Logic* 4 (1963), 171–187.)

In 1969 Curry's correspondence was extended by Howard to one between *proofs* in a sequent calculus version of intuitionistic logic and typed λ-terms. He further observed (after [Tait, 1965]) that cut-elimination was a consequence of the normalization of terms. Howard's 1969 manuscript circulated informally and became the *locus classicus* of the correspondence, especially in the proof-theoretic literature, after [Prawitz, 1971] and [Girard, 1972]. It was eventually published in [Howard, 1980]. (Roughly, the idea behind Howard's contribution may be described as a generalization of Gödel's theory of primitive recursive functionals of finite type by extending the set of types to formulas of Heyting arithmetic.)

Independently of these proof-theorists and with very different motivations, but at about the same time, there were also two other efforts that led to formalisms centred systematically around the correspondence between proofs and terms, and whose technical and philosophical insights still underlie current work on applications of typed λ-calculi to the mechanization of mathematical proofs. These were Dana Scott's work on "constructive validity" [Scott, 1970a], motivated explicitly by an attempt to analyze the notion of *construction* as used by the intuitionists, and Nicolaas de Bruijn's Automath family of languages for checking the correctness of ordinary mathematical proofs, [Bruijn, 1980].

The starting point of the Automath project was around 1967, when de Bruijn began to look at the problem of devising a formal language such that the correctness of a completely formalized mathematical proof can be verified by a computer program [Bruijn, 1970].[57] The design of the Automath languages did not depend on any philosophical position concerning the foundations of mathematics; in fact, it is possible to describe the overall format of an Automath text (called a *book* in

[57] Our description here is based on [Bruijn, 1994], which contains several remarks of historical interest on the Automath languages. More details can be found in [Bruijn, 1980], and the editors' introduction to [Nederpelt *et al.*, 1994], which is a collection of the basic papers on Automath by de Bruijn and others.

the Automath jargon) by asking what items of information a machine would need in order to check its correctness. First of all, the text is subdivided into *lines*, each of them composed of the following items: (1) an *identifier* to refer to that line later in the book; (2) a *definition*, indicating whether the identifier is a *primitive notion*, a *free variable* or the abbreviation of a compound *expression* formed from previous identifiers; (3) a *context* that keeps track of which assumptions are valid and which identifiers are currently visible at that line, and finally (4) a *category* for the identifier, including a primitive category type.[58] The context structure of the Automath languages is a way of describing proofs in a natural deduction format, and was suggested by the similarity between the block structure of the programming language Algol [Naur and others, 1963] and the way mathematicians structure proofs.[59] The λ-notation was familar to de Bruijn, who used it systematically in his work in analysis, and it was natural for him to include (typed) λ-terms in Automath in order to discharge assumptions by binding free variables, thus making contexts shrink as well as expand.

More relevant from the point of view of this Section is the interpretation of the category part of an Automath line. The leading formal insight here consists in unifying the treatment of proofs and objects by observing that in informal mathematical discourse the two notions play the same role. In the statement of a theorem T it is usual to declare the category of the objects involved, e.g. "Let x be an integer," and the properties that these objects have to satisfy, e.g. "Assume that $x > 0$." The conclusion of the theorem then states a property that depends on the (free) name x, which can be instantiated to any specific object k provided that k is an integer *and* that we have a proof that $k > 0$. Then the theorem T is more faithfully represented by an expression of the form $\mathrm{T}(k, \pi)$, showing its dependence on both k and a proof π that $k > 0$. In a line where the identifier a has the expression e of category C as *definiens* (in a context Γ), a has two possible interpretations: as the name of the object e of type C, or as the name of a proof e of a proposition P, provided that C can be interpreted as the type $\mathsf{Proof}(P)$ of proofs of P.[60] Types are not formally identical to propositions; rather, to every proposition can be assigned a type, the type of its proofs, and asserting a proposition amounts to saying that the type of its proofs is inhabited [Bruijn, 1970].[61] This paradigm of "proofs as objects" specializes to the constructive interpretation of logical constants following [Heyting, 1956] where, for example, a construction of an implication $A \supset B$ is a function that maps each construction of A to a construction of B. This is recognized in [Bruijn, 1994, p.205] to have been one of the clues to the interpretation of proof classes as types.[62]

Typed λ-abstraction is denoted, in languages of the Automath family, by $[x\!:\!A]B$ where B might be a type. If, in a context Γ extended with the assumption that $x\!:\!A$, the expression e (possibly depending on x) has type B, then the abstraction $[x\!:\!A]e$ has type $[x\!:\!A]B$; but observe that here the expression B can also depend on the variable $x\!:\!A$ (i.e., B may contain free occurrences of x). One interpretation of the type $[x\!:\!A]B$ is set-theoretic: it is the type of functions that assign a result of type

[58] For a heuristic explanation of how the format of Automath lines can be motivated starting from a complete formalization of natural deduction proofs, see [Daalen, 1980]. Observe that, except for the identifier part, the components of a line correspond to those of an ordinary typing judgement $\Gamma \vdash e : C$, for a context Γ, an expression e and a category C.

[59] See [Bruijn, 1994]; the natural deduction formalism used by de Bruijn is close to that used by [Fitch, 1952]. It is worth remarking that the block structure of Algol 60 led Landin to the formal analysis of that language by means of λ-calculus [Landin, 1965].

[60] In de Bruijn's papers, this type was denoted by $TRUE(P)$.

[61] This is similar to Kreisel's notion of *modified realizability* [Kreisel, 1959] for HA^ω, where the type of realizers of a formula $A \supset B$ is the function-set $\sigma \to \tau$, if σ is the type of realizers of A and τ of those of B. See [Oosten, 2002] for a history of realizability.

[62] The constructive explanation of the meaning of logical constants was learned by de Bruijn directly from Heyting, who was one of his colleagues in Amsterdam from 1952 to 1960.

$B(a)$ to every argument a of type A, a type that may be written as $\prod_{x:A} B(x)$.[63] When B does not depend on x, $[x\!:\!A]B$ is just another notation for the function type $A \to B$. Another interpretation is logical: if A and B represent the proof classes $\mathsf{Proof}(P)$ and $\mathsf{Proof}(Q)$, respectively, then the type $[x\!:\!\mathsf{Proof}(P)]\,\mathsf{Proof}(Q)$ represents the proof class $\mathsf{Proof}(\forall x\!:\!P.\,Q)$, where the formula $\forall x\!:\!P.\,Q$ specializes to $P \supset Q$ when Q does not contain free occurrences of x.

A more uniform approach is obtained in later languages of the Automath family, [Daalen, 1973], where besides the category type there is a category prop intended to represent the type of proofs of propositions. If $A\!:\!\mathsf{prop}$ and $B\!:\!\mathsf{prop}$ whenever $x\!:\!A$, then $[x\!:\!A]\,B$ can be read as an *abbreviation* of $\forall x\!:\!A.\,B$ and, when $x\!:\!A$ is not free in B, of $A \supset B$.

A different and more complex background was behind [Scott, 1970a]. Scott was interested in setting up a formal calculus of constructions that would substantiate Heyting's claim that "there is no essential difference between logical and mathematical theorems, because both sorts of theorems affirm that we have succeeded in performing constructions satisfying certain conditions" [Heyting, 1958].[64]

The work [Scott, 1970a] lay at the confluence of several threads. On the one hand Kreisel's papers [Kreisel, 1962; Kreisel, 1965] aroused Scott's interest in the problem of finding a theory of proofs and constructions on which the interpretation of intuitionistic logic could be based. Kreisel's project was taken up by Goodman [Goodman, 1970] (based on his PhD thesis in Stanford in 1968, supervised by Dana Scott), who deviated from the original formulation in allowing partial functions, represented by type-free combinators.

On the other hand, there was the work of Läuchli [Läuchli, 1965; Läuchli, 1970] on a set-theoretical version of Kleene's recursive realizability interpretation of intuitionistic logic [Kleene, 1945], that we have seen in §8.1.3 above to have influenced also Lawvere's work in categorical logic.[65]

Scott took as central the connection, already exploited by Läuchli and Lawvere, between the universal quantifier and the cartesian product of an indexed family of species, and considered also the dual notion of the sum of such a family, with the evident connection to existential quantification. In general terms, Scott's system was "an attempt at axiomatizing in a constructive way a theory of *both* functions and families of sets of functions" [Scott, 1970a, p.241].[66] The system of Scott uses intuitionistic sequents $\Delta \vdash \mathcal{A}$ where formulas $\mathcal{A}$ are of the two shapes $A \in B$ and $A = B$, for terms A, B that may denote functions or species depending on the context, where species here should be understood as *types*. Term-forming operations include typed functional abstraction $\lambda x \in A.\,B$, which satisfies standard rules with

[63] If B contains x free, this is a *dependent type*; for other examples, see §8.3 and **G** in §5.4.

[64] A detailed description of the motivations behind [Scott, 1970a], on which our account is based, is contained in that paper on pp.237–242. The development of Scott's theory of constructions was also influenced at an early stage by the design of Automath: a preliminary version of the above paper, containing a discussion of de Bruijn's language, was presented in November 1968 at Troelstra's seminar [Scott, 1968], while Scott was on sabbatical leave in Amsterdam. The paper itself was read in December of the same year at the Versailles conference on Automatic Demonstration, whose proceedings also include the first published paper on Automath, [Bruijn, 1970].

[65] The Russian logician Ju. T. Medvedev in [Medvedev, 1962] had described an interpretation of the positive part of the intuitionistic propositional calculus in terms of finite problems, following Kolmogorov, that is quite close to that given by Läuchli. In particular, Medvedev interpreted conjunction, disjunction and implication by means of cartesian product, disjoint union and exponentiation of sets, as Läuchli did.

[66] The attempt to develop a purely category-theoretic treatment of parameterized families of sets started around 1970 in Lawvere's lectures at Dalhousie University, was further developed in [Lawvere, 1972] and led to the theory of indexed categories, [Johnstone *et al.*, 1978], that may be seen as a continuation of Lawvere's earlier work on hyperdoctrines (§8.1.3). A related approach was that of [Bénabou, 1985] using fibrations, and both approaches are relevant to the semantics of dependent types and higher-order type theories discussed later, in §8.3.

the notable exception of rule (ξ) to allow for an intensional interpretation. The type of the above λ-abstract is a dependent product given by the rule:

$$\frac{\Delta,\ x \in A \ \vdash\ B \in C}{\Delta \ \vdash\ (\lambda x \in A\,.\,B) \ \in\ (\forall x \in A\,.\,C)} \quad \text{(where } x \notin \Delta\text{)}$$

which is at the same time the $\forall$-introduction rule. The dual elimination rule is the typing rule for application:

$$\Delta,\ f \in (\forall x \in A\,.\,B),\ x \in A \ \vdash\ f(x) \in B.$$

The usual (non-dependent) function type becomes $A \to B =_{\text{def}} \forall x \in A\,.\,B$ where B does not depend on $x \in A$. It is under this identification that the correspondence between types and propositions becomes evident, and in fact a theorem of this calculus of constructions like

$$x \in A \ \vdash\ [\forall f \in (\forall x \in A\,.\,B)\,.\,f(x)] \ \in\ (\forall x \in A\,.\,B) \to B$$

is the logical axiom of universal instantiation [Scott, 1970a, p.248]. The clearest statement of how the correspondence works in general is to be found in this paper (p.260):

> "We began by regarding species and constructions as mathematical objects and found that there were some simple axioms governing their properties. It then became slowly apparent that these properties were highly analogous to properties familiar from formal logic. We then turned this analogy into a dogma by insisting that the logical formulas be read (better: interpreted) as (mathematically meaningful) terms of the theory of constructions. This interpretation requires that validity be asserted by the act of giving an explicit construction belonging to the (interpretation of the) formula. Validity is established by giving a proof from the axioms for constructions of the membership assertion. [...] a proposition does not simply degenerate to one of two truth values but instead is represented by a complex species of possible constructions that conceivably can be used in its validation."

Scott's full description of the correspondence follows almost *verbatim* the explanations of the meaning of the logical constants given in [Heyting, 1956; Heyting, 1958]:

Formula	*Construction*
$A \wedge B$	a pair of constructions, the first of which justifies A, and the second B
$A \vee B$	a pair whose first coordinate is either 0 or 1: if 0 then the second coordinate justifies A; if 1, then B
$\top$	the justification is 0, as $\top =_{\text{def}} \{0\}$
$\bot$	no justification is known, as $\bot =_{\text{def}} \{\ \ \}$
$\forall x \in A\,.\,B$	a construction that maps every element $a \in A$ into a justification of $B(a)$
$\exists x \in A\,.\,B$	a pair whose first coordinate is an element $a \in A$ and whose second coordinate justifies $B(a)$.

The correspondence between proposition and types was used systematically in the intuitionistic type theory of Martin-Löf. That theory was first proposed in 1971 but then underwent several revisions; cf. [Martin-Löf, 1971b; Martin-Löf, 1972b;

Martin-Löf, 1975; Martin-Löf, 1984].[67] Martin-Löf had learned about this correspondence from Howard in early December 1968, and had exploited it already in a paper, written in March 1969 and eventually published as [Martin-Löf, 1972a], on the infinitary proof theory implicit in [Tait, 1965]. The first proposal of intuitionistic type theory included a type constant V such that $A : V$ meant that A is a type, with the axiom that $V : V$. This assumption allowed for a very succinct system: the primitive notions could be reduced, essentially, to typed λ-abstraction, dependent product and the constant V, using the coding of Russell and Prawitz to define the following basic type constructors (i.e., logical connectives) [Russell, 1903, §§18,19], [Prawitz, 1965, Ch. V, Thm. 1]:

$$\begin{array}{rcl} \bot & =_{\mathrm{def}} & \forall X : V \,.\, X \\ A \times B & =_{\mathrm{def}} & \forall X : V \,.\, (A \to B \to X) \to X \\ A + B & =_{\mathrm{def}} & \forall X : V \,.\, (A \to X) \to (B \to X) \to X \\ \exists x : A \,.\, B(x) & =_{\mathrm{def}} & \forall X : V \,.\, (\forall x : A \,.\, (B(x) \to X) \to X. \end{array}$$

There were substantial motivations for making the strong impredicative assumption of a type of all types, based on the simultaneous acceptance of

> "the following three principles. First, quantification over propositions, as in impredicative second order logic. Second, Russell's doctrine of types according to which the ranges of significance of propositional functions form types so that, in particular, it is only meaningful to quantify over all objects of a certain type. Third, the identification of propositions and types" [Martin-Löf, 1971a, p.5].

However, the theory turned out to be inconsistent, as a form of the Burali-Forti paradox could be coded in it.[68] The impredicative character of the theory was removed in its subsequent versions [Martin-Löf, 1972b; Martin-Löf, 1975; Martin-Löf, 1982; Martin-Löf, 1984]. Broadly, the outcome can be described as an intuitionistic theory of iterated inductive definitions developed in the framework of dependent types,[69] but here the correspondence between types and propositions is even more systematically exploited than in the systems discussed earlier. The correspondence suggests a formulation of the rules for type-forming operations as introduction and elimination rules, as in [Gentzen, 1935]. This yields automatically a notion of reduction for terms as deductions as in [Prawitz, 1965; Prawitz, 1971], from which computation rules can be read off directly. The resulting system then becomes a functional programming language including its own logic, whose formulas are regarded as program specifications, as Martin-Löf himself pointed out in [Martin-Löf, 1982].[70] It was that paper that first outlined a correspondence between proof-theory and the theory of functional programming languages that had a widespread conceptual and terminological influence on logically-oriented functional programming. In particular, under the identification of propositions with types, the introduction rules

[67] Martin-Löf's work on this theory began in October 1970. It grew out of earlier proof-theoretical work on normal form theorems (*Hauptsätze*) for the intuitionistic theory of iterated inductive definitions, the theory of species and intuitionistic simple type theory, and also an interest in category theory, which was found inadequate as a foundational framework because of its lack of a clear distinction between extensional and definitional equality.

[68] This was proved by Girard [Girard, 1972, III, Annexe A], and his paradox was studied in [Coquand, 1986]; for a study of this system from a different point of view, see [Jacobs, 1989], who suggested that Girard's paradox could be viewed as a version of the paradox of Mirimanoff on the universe of well-founded sets [Mirimanoff, 1917].

[69] This appropriate description was given in [Coquand, 1999].

[70] The particular technique used in describing the computation rules in [Martin-Löf, 1982] is closely related to what is now called "structured operational semantics", cf. [Plotkin, 1981] and [Kahn, 1988].

behave as the *constructors* of [Landin, 1964], while logical constants correspond to operations that form new types by specifying what are their canonical elements (i.e., those whose outermost symbol is a constructor), like the *concrete data types* of functional programming languages since [Burstall, 1977] (see [Peyton Jones, 1987, §4.1.4]).

On the more philosophical side, these ideas are also in accordance with a theory of meaning that started from [Gentzen, 1935, §II.5.13], according to which the meaning of a logical constant is determined by its introduction rules in a system of natural deduction, whereas the corresponding elimination rules depend on this meaning. It was this idea that led to a precise formulation in [Prawitz, 1965] of the "inversion principle" of [Lorenzen, 1955], where it contributed to isolating the reduction rules for derivations in natural deduction. The meanings of the constants of Martin-Löf's intuitionistic theory of types were first studied in [Hancock and Martin-Löf, 1975] for a simple language of primitive recursive definitions; these studies were brought to a more mature state with [Martin-Löf, 1984; Martin-Löf, 1985; Martin-Löf, 1987], and were very relevant to the work on the semantics of intuitionistic logical constants, and more generally to the constructive theories of meaning and the philosophical foundations of the intuitionistic interpretation of mathematical propositions (see [Sundholm, 1986] for a survey of this area until 1985).

Work related in several ways to Martin-Löf's theory of types had also been carried out at Cornell University by Robert Constable and his collaborators from the early 1970s. Constable attributed to Kleene's work on realizability in the 1940s the awareness that various kinds of constructive proofs can be compiled into executable code, [Constable, 1982, p.3]. For instance a proof of an assertion of the form "for all integers n there is an integer y such that $R(n, y)$" can be compiled into a computable function f such that $R(n, f(n))$.

He proposed in [Constable, 1971] that certain constructive formal systems could be used directly as programming languages, and designed a programming logic based on this idea in [Constable and O'Donnell, 1978]. Further refinements of this system, [Constable, 1980; Constable, 1985; Bates and Constable, 1985], together with the inclusion of ideas from [Martin-Löf, 1982] and from Edinburgh LCF (§8.1.2), led eventually to the Nuprl system [Constable and others, 1986] that manipulated proofs directly in a version of Martin-Löf's intuitionistic type theory.

From the researchers in the Nuprl group also emerged an extension of the "types as propositions" idea to classical logic. In particular, Tim Griffin and Chetan Murthy ([Griffin, 1990], [Murthy, 1991]) observed a correspondence between the *continuation passing style* translation of control operators for λ-calculus [Felleisen *et al.*, 1987] and the double-negation translation of classical into intuitionistic logic as described, for example, in [Friedman, 1978].

8.2 Early normalization proofs

From a syntactic point of view, the most desirable properties of a system of typed terms include the *weak normalization (WN)* property that every typed term has a normal form, and, better, the *strong normalization (SN)* property that no infinite reductions exist.

The first proof of such a property for λ or CL was by Turing, as mentioned in §5.1. Turing's proof covered λ-terms with pure arrow types and β-reduction. It was found in a typewritten note among his papers after his death, and is known to have been composed before 1942, but was not published until about 40 years later, when it appeared as [Turing, 1980].

In the meantime Curry made a very different WN proof in the 1950s using cut-elimination, and this was the first proof to be published; it appeared in [Curry and

Feys, 1958, §9F6, Cor. 9F9.2].[71]

The method used by Turing was re-discovered independently several times in the 1960s and '70s, for example by James H. Morris in his unpublished thesis [Morris, 1968, p.107 Thm. 2], and by Garrel Pottinger in [Pottinger, 1978, p.448] (with ordinals added in). The first published proof using Turing's method was in [Andrews, 1971, p.417, Prop. 2.7.3], where the author's source for the proof was given as James Guard, who was, like Andrews, a 1960s student of Church, and the theorem was attributed to "folklore".

For stronger type-systems, several WN proofs were made in the 1960s. The initial stimulus was Gödel's *Dialectica* paper [Gödel, 1958], together with studies of the ideas behind it led by Kreisel in Stanford University. Roughly speaking, Gödel interpreted first-order arithmetic in a logic-free theory T of *primitive recursive functionals of finite type*, and deduced the consistency of arithmetic by finitary reasoning from an assumption that all closed terms in T of numerical type (closed type-0 terms) computed to unique numerals. One way of formalizing T was as a version of typed CL or λ augmented by primitive-recursion operators, and the above computability property of T then became the statement that WN held for all closed type-0 terms of the augmented CL or λ..[72] This drew the problem of verifying the WN theorem for such extensions of CL and λ to the forefront, and at least five different WN proofs came into being by 1966. Two were made around 1963 by Tait. One of these was published in [Tait, 1965]; it covered a version of λ augmented by certain infinite terms, but based on a weak λ-reduction analogous to weak reduction in CL. The other was reported in the informally circulated notes of a seminar held at Stanford in 1963 (Section V Appendix B), but was not actually published until [Tait, 1967]; it covered CL augmented by operators for primitive recursion and bar-recursion. It introduced the method due to Tait that is now more or less standard, of defining by a carefully organised induction on types a *computability predicate* that implies normalizability, and proving by induction on terms that all terms satisfy this predicate.[73]

Other early normalization proofs for versions of Gödel's T included ones in 1965 by Luis Sanchis for $(\mathrm{CL}{+}\mathbf{R})_t$ [Sanchis, 1967, Thm. 8], in 1966 by Justus Diller for $(\lambda\beta+\mathbf{R}')_t$ [Diller, 1968, §6], in 1966 by Yoshito Hanatani [Hanatani, 1966], in 1967 by Shigeru Hinata [Hinata, 1967], and in 1968 by A. G. Dragalin [Dragalin, 1968]. For more on normalization proofs stemming from Gödel's T, see [Troelstra, 1973, §§2.2.1–2.3.13, esp. 2.2.35].

The property SN can be seen as a safety feature for computing systems. But at first there was no interest in this property, as WN was all that was needed in consistency proofs; the WN proof in [Tait, 1967] would have needed almost no change to give SN had Tait wanted it. The first explicit SN proof was Sanchis' 1965 proof mentioned above. An SN proof by directly assigning ordinals to terms was made by Howard in 1968, [Howard, 1970]; it worked for weak λ-reduction augmented by $\mathbf{R}$, but not for full-strength $\lambda\beta\mathbf{R}$-reduction. (An extension to that reduction had

[71] Curry's WN theorem was for CL with $\beta\eta$-strong reduction. It was deduced via a λ-CL translation from his cut-elimination theorem for a system based on $\lambda\beta$, see §5.4 above.

[72] A simple formalization of Gödel's T is the system $(\mathrm{CL}{+}\mathbf{R})_t$ defined in outline in [Hindley and Seldin, 1986, 1st edition, pp.297–299]. It consists of CL with weak reduction plus a constant $\mathbf{R}_\tau$ for every type-expression τ, with reduction-rules $\mathbf{R}_\tau XY\mathbf{0} \rhd X$, $\mathbf{R}_\tau XY(\mathbf{succ}\,\mathbf{n}) \rhd Y\mathbf{n}(\mathbf{R}_\tau XY\mathbf{n})$. (Types are omitted here, and $\mathbf{0}$ and $\mathbf{succ}$ are atoms.) A proof of WN (indeed SN) for $(\mathrm{CL}{+}\mathbf{R})_t$ is in [Hindley and Seldin, 1986, 1st edition, Appendix 2], based on the method in [Tait, 1967].

By the way, although $\mathbf{R}_\tau$ was adequate to formalize T, [Tait, 1967] and many other formalizations were actually equivalent to a slightly stronger operator $\mathbf{R}'_\tau$ with the rule $\mathbf{R}'_\tau XY(\mathbf{succ}\,Z) \rhd YZ(\mathbf{R}'_\tau XYZ)$.

[73] In [Tait, 2001, §5] Tait expressed the opinion that his computability idea had been anticipated by Gödel in unpublished notes in 1938 and lectures which were given in 1941 but not published in print, see [Gödel, 1995, items *1938a, *1941]. But our interest here is only in explicit proofs of WN, and none occurs in these sources.

to wait for nearly 30 years, until Gunnar Wilken and Andreas Weiermann's [Wilken and Weiermann, 1997].) After the 1960s, most workers studying normalization aimed to prove SN whenever possible.

Normalization proofs for Gentzen-style natural deduction systems of logic are also relevant here, because deductions in such systems can be viewed as a kind of typed λ-terms. Some such proofs will be mentioned in the following sections. For others, the reader is referred to the history of mainstream proof theory.

8.3 Higher-order type theories

In 1953, Gaisi Takeuti formulated a sequent calculus GLC for higher-order logic and conjectured that the cut-elimination property would hold for it, i.e. that if a formula had a derivation D in GLC then it would have a derivation D^* in which the cut-rule was not used, [Takeuti, 1953].

After 13 years, two proofs of Takeuti's conjecture were eventually found for the second-order fragment of GLC [Tait, 1966; Prawitz, 1967], and Moto-o Takahashi and Dag Prawitz found proofs for the whole of GLC [Takahashi, 1967; Prawitz, 1968]. But these proofs gave no procedure for finding D^* from D; they all relied on a result of Kurt Schütte which reduced the conjecture to the problem of proving that every partial valuation could be extended to a total valuation [Schütte, 1960].

The first proof to contain a procedure to find D^* was made by Jean-Yves Girard in 1970, see [Girard, 1971; Girard, 1972].

Girard formulated two systems of typed λ-calculus, $\mathbb{F}$ corresponding to the second-order fragment of GLC, and the system nowadays called $\mathbb{F}_\omega$ corresponding to full GLC, and he solved Takeuti's problem via proofs of SN for these systems. The key idea in his SN proofs, *candidats de réductibilité*, was a development of Tait's computability method, [Tait, 1967], and was very quickly adopted and extended by other workers in proof-theory and has been used widely since.[74]

Girard's system $\mathbb{F}$ is particularly important in our story.[75] Its main novelties were the possibility of forming universally quantified types $\forall t.\sigma$, and the operation of applying a term M of type $\forall t.\sigma$ to a type τ. Corresponding to this operation there was an abstraction operation $\Lambda t.M$ to represent a function whose application to a type τ reduced to the term $M\{\tau\}$ of type $[\tau/t]\sigma$. The impredicativity implicit in this extension, whereby the type variable t in $\forall t.\sigma$ ranged over *all* types, including $\forall t.\sigma$ itself, made the SN proof for reductions in this calculus far from obvious.

The impact of system $\mathbb{F}$ was limited to proof theory in the years immediately following its invention, but the idea of having λ-terms taking types as parameters was also becoming important in those years in the theory of programming languages (see, for example, [Cheatham Jr. *et al.*, 1968]). It is not surprising, then, that almost the same system was independently reinvented in 1974 by John Reynolds, [Reynolds, 1974].[76] One motivation for the development of a system of quantified types was the formalization of ideas on polymorphism in programming languages expressed by Christopher Strachey (see, e.g., [Strachey, 1967, §3.6.4]), especially the notion of *parametric polymorphism*. Take, for example, the function map that applies a function f to all elements of a list. If $f : \sigma \to \tau$, then it is natural to assign to $\mathsf{map}(f)$ the type $\mathsf{List}(\sigma) \to \mathsf{List}(\tau)$ that depends parametrically on σ and τ. By using universally quantified types, map could be assigned the type

$$\forall s.\forall t.\ (s \to t) \to (\mathsf{List}(s) \to \mathsf{List}(t)).$$

[74] An analysis of Girard's method, and of several SN proofs that use it, is [Gallier, 1990].

[75] For an introduction to system $\mathbb{F}$, see [Girard *et al.*, 1989].

[76] Reynolds has often pointed out that he presented his paper in April 1974 at a meeting at Paris VII University, where Girard was teaching, but nobody in the audience noticed the similarity of the two systems or brought Girard's work to his attention.

An important remark was that polymorphic terms can be used to represent basic data structures; for example the natural numbers can be represented by polymorphic Church numerals of the form

$$\Lambda t.\ \lambda f : t \to t.\ \lambda x : t.\ \ f(\cdots(fx)\cdots),$$

each having type

$$\mathsf{nat} \ \ = \ \ \forall t.\ (t \to t) \to (t \to t),$$

see [Girard, 1972, p. I.5.5]. Representations which occurred in [Girard, 1972, pp. I.5.2, I.5.3], also in [Martin-Löf, 1971b, §3] which referred to [Russell, 1903], were

$$\begin{aligned} \sigma \times \tau &= \Lambda t.\ ((\sigma \to (\tau \to t)) \to t), \\ \sigma + \tau &= \Lambda t.\ ((\sigma \to t) \to ((\tau \to t) \to t)). \end{aligned}$$

The boolean values had representations

$$\Lambda t.\ \lambda x : t.\ \lambda y : t.\ x, \qquad \Lambda t.\ \lambda x : t.\ \lambda y : t.\ y$$

of type

$$\mathsf{bool} \ \ = \ \ \forall t.\ t \to (t \to t).$$

The representation of a list $[e_1, \ldots, e_n]$ of elements of some type s appeared in [Reynolds, 1983, p.520] as the term

$$\Lambda t.\ \ \lambda f : s \to (t \to t).\ \ \lambda x : t.\ \ fe_1(\cdots(fe_n x)\cdots)$$

of type

$$\mathrm{list}(s) \ \ = \ \ \forall t.\ (s \to (t \to t)) \to (t \to t)$$

and was said to have been "suggested by a functional encoding of lists devised by C. Böhm".

Another source for representations by polymorphically typed terms was [Leivant, 1983]; this had some influence on Böhm and Berarducci [Böhm and Berarducci, 1985] which contained a systematic account of how to encode many-sorted term algebras as closed types of the polymorphic λ-calculus, together with the iteratively defined functions over them as polymorphic terms.[77]

We have remarked that the main motivation for Reynolds' development of the polymorphic typed λ-calculus was the formal clarification of Strachey's notion of *parametricity*, by describing the uniform dependence of a polymorphic term $\Lambda t.M$ upon an input type. Informally, the behaviour of a parametric function defined by such a term should not depend on the input type or, in other words, should be the same for all input types [Reynolds, 1983, p.519]. The main technical difficulty was to compare values whose types were different, and Reynolds proposed a semantic criterion of parametricity, consisting in the property of terms of the form $\Lambda t.M$ of preserving certain hereditarily defined relations between sets corresponding to different types [Reynolds, 1983].[78] Reynolds' notion of parametricity was based, however, on a set-theoretical interpretation of polymorphic types extending the general models of Henkin (§8.1.2 above). Accordingly, the relational account of parametricity showed several similarities with the theory of logical relations, in particular with the treatment of definability in [Plotkin, 1980].

The existence of a set-theoretic model for the polymorphic typed λ-calculus was conjectured in [Reynolds, 1983]. But a year later Reynolds disproved his own

[77] There are several accounts of these representations in the literature: for example [Girard *et al.*, 1989, §§11.3,11.4,11.5] and [Leivant, 1990, §3], besides the original papers.

[78] Formal systems based on this relational approach to parametricity are described in [Abadi *et al.*, 1993] and [Plotkin and Abadi, 1993]. For a survey of parametricity see also [Longo *et al.*, 1993, §1] and [Longo, 1995].

conjecture by showing that the functor (over the category of sets and functions) $(2^X)^X$ is definable by a type expression, hence should have an initial algebra $A \cong (2^A)^A$, which is impossible by a cardinality argument.[79]

The model theory of system $\mathbb{F}$ was initiated by Girard in his [Girard, 1972, §II.5], which contained the description of a model HEO_2 (and its generalizations to higher-orders HEO_n) based on the Kleene applicative structure $\langle \mathbb{N}, \cdot \rangle$, where $n \cdot m$ denoted the result of applying the n-th partial recursive function to the argument m. Every type σ was interpreted as a pair consisting of a set $D_\sigma \subseteq \mathbb{N} \times \{\sigma\}$ and an equivalence relation $=_\sigma$ over D_σ.

For functional types $\sigma \to \tau$ this interpretation was specified by the clauses:

- $D_{\sigma\to\tau}$ is the set of all pairs $(e,\ \sigma \to \tau)$ such that for all (e_1, σ) and (e_2, σ) in D_σ, $e \cdot e_i$ is defined for $i = 1, 2$ and is an element of D_τ, and $(e \cdot e_1, \tau) =_\tau (e \cdot e_2, \tau)$ whenever $(e_1, \sigma) =_\sigma (e_2, \sigma)$;

- $(e_1, \sigma \to \tau) =_{\sigma\to\tau} (e_2, \sigma \to \tau)$ iff $(e_1 \cdot x, \tau) =_\tau (e_2 \cdot x, \tau)$ for all (x, σ) in D_σ.

The interpretation of for-all-types used the notion of *primary type* P, defined as a set $A \subseteq \mathbb{N}$ containing 0 together with an equivalence relation on A. Then the pair $D_{\forall t.\sigma}$ with the equivalence $=_{\forall t.\sigma}$ was specified by the clauses:

- $D_{\forall t.\sigma}$ is the set of all pairs $(e, \forall t.\sigma)$ such that $(e, [P/t]\sigma) \in D_{[P/t]\sigma}$ for all primary types P;

- $(e_1, \forall t.\sigma) =_{\forall t.\sigma} (e_2, \forall t.\sigma)$ iff $(e_1, [P/t]\sigma) =_{[P/t]\sigma} (e_2, [P/t]\sigma)$ for all primary types P.

With some simplifications, see [Troelstra, 1973, §§2.4.11, 2.9.7], this was to become probably the best-known model for the polymorphic typed λ-calculus, the model based on partial equivalence relations (*PERs*), i.e., symmetric and transitive relations over some type-free structure like $\langle \mathbb{N}, \cdot \rangle$ or, later, over some model of untyped λ-calculus [Scott, 1976, §7].[80]

Reynolds' proof of the non-existence of set-theoretic models of polymorphism made an essential appeal to classical logic. In fact, two years after Reynolds' result, it turned out that, within a constructive metatheory provided by the internal logic of an elementary topos, the set-theoretic intuitions behind the polymorphic λ-calculus could be rescued by internalizing the constructions of category theory within such a topos, looking at the latter as a constructive universe of sets.

Already in 1984, Eugenio Moggi described to Martin Hyland, then visiting Pisa, a way of looking at a model of polymorphic λ-calculus as an internal cartesian closed category $\mathbb{M}$ in a topos E, with $\mathbb{M}$ having all products indexed over the objects of $\mathbb{M}$ itself. Moggi found such a model in 1986 by taking E to be the effective topos of [Hyland, 1982] and $\mathbb{M}$ as the category of *modest sets*, which was equivalent to the familiar category of partial equivalence relations over the applicative structure $\langle \mathbb{N}, \cdot \rangle$, directly related to Girard's HEO_2 model of system $\mathbb{F}$. The right notion of completeness needed to make $\mathbb{M}$ a small complete category was then found by [Hyland, 1988].

Remarkably, the intersection of a family of partial equivalence relations becomes their product under internalization. It is the existence of such a small complete

[79] Reynolds' result was generalized by Plotkin in a letter to Reynolds dated July 16, 1984. His remarks led to [Reynolds and Plotkin, 1990]. (By the way, [Reynolds, 1974] contained a proposal for a model of Reynolds' first polymorphic system, but with a warning that its discussion contained a "serious lacuna". As he has pointed out since, this warning turned out to be justified, in that the proposed model did not work.)

[80] Some historical remarks on the notion of PER as an interpretation of types are given in [Bruce *et al.*, 1990], where we learn that they were introduced in [Myhill and Shepherdson, 1955] for types of first-order functions, and then extended to simple types by [Kreisel, 1959].

category inside a universe of constructive sets that enables one to interpret a polymorphic type $\forall X.A[X]$, roughly, as a product $\prod_{X\in\mathbb{M}} A_X$ of a family of elements of $\mathbb{M}$ indexed by $\mathbb{M}$ itself.[81] In particular, completeness of $\mathbb{M}$ implies that every functor $T : \mathbb{M} \to \mathbb{M}$ has an initial algebra $\langle A,\, a : TA \to A\rangle$, which is the stumbling block to the construction of a set-theoretic model of system $\mathbb{F}$.

From the present point of view, it is of some interest to observe that these developments exploit in an essential way the language of category theory, especially the notion of *indexed category* [Johnstone *et al.*, 1978], which has become a fundamental tool in the semantical description of theories of dependent types and may be regarded as a natural outgrowth of Lawvere's work on hyperdoctrines and topoi in the 1970s.

A polymorphic type $\forall t.\sigma$ is, implicitly, a dependent type $(\Pi t{:}\mathsf{type}).\,\sigma$, provided suitable rules about the constant type are added to systems with dependent types like those of §8.1.4 or §5.4. As mentioned in §5.4, a general system of dependent types was developed and studied from 1972 to 1975 by Seldin, [Seldin, 1979]. But a stronger system offering an integration of polymorphism with dependent types was the *calculus of constructions*, developed by Thierry Coquand and Gérard Huet from 1984 onwards. It was described in a series of papers starting with Coquand's Thèse de Troisième Cycle: [Coquand, 1985], [Coquand and Huet, 1985], [Coquand and Huet, 1988], [Coquand, 1990]. The calculus of constructions has been concisely described as "a version of type theory that expresses naturally Heyting semantics of intuitionistic higher-order logic" [Coquand, 1990, p.91]. Paradoxes like that described by Girard for Martin-Löf's impredicative version of intuitionistic type theory were avoided by adopting a restricted form of the identification between types and propositions, which only assumed that to each proposition is associated the type of its proofs. But the other two points in Martin-Löf's argument for introducing a type of all types quoted on page 34 above, namely quantification over propositions and the doctrine of types as ranges of significance of propositional functions, were preserved. The resulting system was therefore quite similar to that described in [Martin-Löf, 1971b], and also to some of the systems in the Automath family, extended by quantification over elements of type, a possibility already envisaged in [Bruijn, 1970, §12.6]. For the calculus of constructions a weak normalization proof was given in [Coquand, 1985, §4, Cor. 1], together with some hints on how Girard's notion of candidat de réductibilité might be extended to give an SN proof for this system. A proof of SN for deductions and for terms was included in [Seldin, 1987, §4.3] and [Seldin, 1997, §3].

8.4 Intersection types and recursive types

In systems where types denote sets, a natural extension of the language of simple type theory would be to adjoin types of the form $\sigma \cap \tau$ to denote the intersection of whatever sets were denoted by σ and τ.[82] This extension was actually made in the late 1970s. Its history was, however, less simple than might be expected given the naturalness of the idea.

Systems of intersection types are usually "Curry-style" systems, in which types are assigned to untyped terms by rules and an infinite number of types may be assigned to one term, rather than "Church-style" systems, in which each term has a unique built-in type. They also use types, not as safety-devices to avoid paradoxes, as Russell and Church did, but simply as labels to describe the formal computational behaviour of terms.

[81] This result was announced in a famous message by Moggi to the `types` electronic forum, at that time moderated by Albert Meyer, dated February 10, 1986. The result and its background are discussed in much more detail in [Longo and Moggi, 1992] and [Asperti and Longo, 1991].

[82] In the present subsection and §8.5, "type" is short for "type-expression".

According to ideas of Böhm on the use of λ-calculus as a programming language (cf. §6.3), data and programs are represented by λ-terms thought of as states of a computation that evolves by applications of the rule of β-reduction, with normal forms corresponding to final states. The notion of program equivalence that arises from this approach regards two programs (i.e. λ-terms) P and P' as equivalent if, for every argument F, either PF and $P'F$ both have normal form and $PF =_\beta P'F$, or both fail to have a normal form.[83] In order to match this notion of equivalence with convertibility, it turned out to be essential that P, P' and F have normal forms, see [Böhm and Dezani, 1975]. This remark prompted a search for a classification of normal forms according to their normalizability when applied to other normal forms.

Starting in 1975, a suitable classification was defined and analysed by Böhm and his former students in Turin, Mariangiola Dezani and Mario Coppo; see [Böhm and Dezani, 1975], [Dezani, 1975], and [Böhm *et al.*, 1977]. The theme of types was introduced in [Dezani, 1975]; each normal form was given a finite sequence $(m, j, h_1, \ldots, h_n)$ or $(\omega, h_1, \ldots, h_n)$, called its "type", which encoded a description of its computational behaviour. Types appeared again in [Coppo and Dezani, 1978], written in 1977, in which simple types were assigned to all normal forms, including those untypable in previous simple type theories, by new rules involving two formally defined relations between types, an equivalence and a pre-order. The atomic types were 0, to represent the set of all normal forms, and 1 for the set of all terms whose application to any finite sequence of normal forms always reduces to a normal form.

In 1977 Coppo and Dezani put forward the idea of assigning a finite sequence of types to a term to say that this term denoted a member of a finite intersection. This was published in [Coppo and Dezani, 1978], and the results in that paper included the theorem that the types received by a term were invariant under conversion in the λI-calculus.

The sequence idea was extended by Patrick Sallé in Toulouse, [Sallé, 1978, §5]. He adjoined a new atomic type-constant ω to denote the universal set, and by this means he extended the conversion-invariance theorem from λI to the full λK-calculus, and described the connection between non-ω types and head-normal forms, see [Sallé, 1978, Thms. 9–12] and the joint paper [Coppo *et al.*, 1979, Thms. 1, 2]. The system with sequences and ω was further studied in [Coppo *et al.*, 1980] and [Coppo *et al.*, 1981]. In §5 of the former paper, Betti Venneri showed that, by interpreting a closed term as the set of all its types, one could obtain a new model of untyped λ-calculus.

The intersection idea was also invented independently by Pottinger, who proved some of its basic properties in [Pottinger, 1980].

But although the intersection system was perfectly adequate for its original purpose, it was not semantically complete. Suitable extra deduction-rules to make it complete were first formulated in 1980, by Coppo, Dezani and Barendregt together, [Barendregt *et al.*, 1983], and by Hindley independently, [Hindley, 1982]. The system given in [Barendregt *et al.*, 1983] soon became standard: its types were built from variables and ω by means of constructors $\rightarrow$ and $\cap$, and it contained rules defining a relation $\leq$ between types analogous to the subset relation. In what is nowadays often called the "generation lemma" [Barendregt *et al.*, 1983, Lemma 2.8] the invertibility of its main rules was proved, and from this a normalization theorem for deductions was deduced, [Barendregt *et al.*, 1983, Cor. 4.9]. In [Barendregt *et al.*, 1983, §3] the first of the interesting class of λ-models called *filter models* was defined, whose elements were filters (i. e., $\leq$-upward closed and $\cap$-closed sets) in the set of all types. These models turned out to be closely related to the original Scott construction to be discussed below in §9.1.

[83] This is close to the notion of contextual equivalence in [Morris, 1968] discussed in §7.1.

Over the next 10 years the system of [Barendregt *et al.*, 1983] was studied intensively, with many interesting results emerging, for example an analysis of its principal types by Simonetta Ronchi della Rocca and Venneri in [Ronchi della Rocca and Venneri, 1984] and [Ronchi della Rocca, 1988]. Applications of intersection types included their use in type systems for the control of interference in Algol-like languages, [Reynolds, 1989] and in static analysis of functional programs, [Coppo and Ferrari, 1993].

The use of types for describing behavioural properties of terms motivated several other type systems in the 1980s, for statically detecting properties of λ-terms and (functional) programs. Among these, the *recursive types* (see, e.g., [MacQueen *et al.*, 1984], [Coppo, 1985] and [Cardone and Coppo, 1991]) were a natural extension of Curry's type inference system by adjoining types of the form $\mu t.\sigma$ to represent the solution of the equation $t = \sigma$, where the type variable t may occur in σ. (The possibility of having circular type expressions was already considered in [Morris, 1968], see above §7.1.) Clearly, recursively typed λ-terms lack the normalization property, e.g. $\Delta\Delta$, where $\Delta \equiv \lambda x.xx$, can be assigned every type σ, provided x is given type $\mu t.t \to \sigma$ (which is equivalent to $(\mu t.t \to \sigma) \to \sigma$). But there is a principal typing algorithm for the basic type inference system for recursive types (extending fairly directly the one for Curry's system, §8.5), and this makes it a rather practical tool for preventing incorrect applications when special constants like e.g. arithmetical primitives are added to the λ-calculus [Wand, 1987; Cardone and Coppo, 1991].

The semantics of recursive types systematically exploited the techniques invented towards the end of the 1960s for solving recursive equations involving sets. Typical among these is the solution of the equation $D = D \to D$ satisfied by a model of pure, extensional λ-calculus that will be discussed at some length in §9.1.

Recursive types are also interesting when added to higher order type systems like system F, or to systems that include a subtyping relation: such a system was first proposed by Luca Cardelli and Peter Wegner in [Cardelli and Wegner, 1985] as a tool for analyzing certain features of object-oriented programming languages, originating a whole new field of research; see, for example, [Gunter and Mitchell, 1994] and [Pierce, 2002].

8.5 Algorithms for simple types

For a given system of terms and types, a *typability test* decides whether an untyped term M is typable in the system; it is called a *principal type algorithm* if it furthermore assigns to any typable term its principal type, i.e. the most general type it can receive under the system's rules. Such algorithms have no meaning for a Church-style type-system, which has no untyped terms in its language, but only for Curry's style, in which types are assigned to untyped terms by rules, cf. §4.2.

In §4.2 we mentioned that Quine's concept of *stratified formula* was based on a Curry-style approach, [Quine, 1937, pp. 78–79]. In 1942 a test for stratification was invented by M. H. Newman and stratification was proved equivalent to typability, [Newman, 1943, §§1–4]. But Newman also extended his algorithm to give a typability test for λ-calculus, with a proof of its correctness. Unfortunately his exposition was rather abstract, and his paper was not read by any later worker on typability tests until the preparation of the present history, as far as we know.[84]

A more direct test, which moreover gave principal types, was described informally in [Curry and Feys, 1958, §§9B2–4]. Roughly speaking, for a given term M

[84]For an outline of [Newman, 1943], see Hindley's extended review *M. H. Newman's typability algorithm for lambda calculus*, Journal of Logic and Computation, to appear, 2006/7. Newman's test was not a principal type algorithm; indeed his paper contained no concept of principal type.)

its method is to list all M's subterm-occurrences with their types given as type-variables, write out equations connecting these variables, and then solve these equations as generally as possible. Several examples were worked out in detail in [Curry and Feys, 1958], but the method was not stated explicitly as a formal algorithm. Curry later wrote it out as a formal algorithm, with a proof of its correctness, in private notes [Curry, 1966] and a paper [Curry, 1969a].

A different principal type algorithm, which used the unification algorithm of [Robinson, 1965] to save itself some work, was made and proved correct by Hindley for CL in 1967, [Hindley, 1969b, §3]. (Curry and Hindley communicated during the writing of [Curry, 1969a] and [Hindley, 1969b] but deliberately kept their approaches different.)

A Curry-style equation-solving algorithm for λ was made, with a correctness proof, by James H. Morris in his thesis [Morris, 1968, Ch. 4, §E], independently of Curry's and Hindley's 1966–69 work.

Then the 1970s and '80s saw the development of the programming language ML by the group led by Robin Milner in Edinburgh. ML included a type-assignment system for λ plus the extra operator *let*, and Milner described a principal type algorithm for it in [Milner, 1978, §4.1, Algorithm W]. This algorithm was unification-based like that in [Hindley, 1969b], though it was invented independently. It was rewritten and extended in [Damas and Milner, 1982], and its correctness was proved by Luis Damas in his thesis [Damas, 1984].

After 1980, typability and principal-type algorithms were made for various more complex systems, some depending on pre-existing unification algorithms and some being self-sufficient: for a survey see [Tiuryn, 1990]. A long-standing open problem was the existence of a typability test for Girard's system $\mathbb{F}$ (see §8.3 above); this was raised by Daniel Leivant in [Leivant, 1983], and was solved in the negative by Joe Wells in 1994; indeed Wells proved that not even type-checking for this system was decidable, [Wells, 1994; Wells, 1999].

But the history of principal type algorithms is not confined to type theory, strange to say. Under the propositions-as-types correspondence, simple types whose atoms are variables become propositional formulas, and a combinator M with principal type τ becomes a proof of τ by the propositional rule called *condensed detachment*.

Roughly speaking, the condensed detachment rule corresponds to the construction of the principal type of a term PQ from those of P and Q. It was invented by the Dublin logician Carew Meredith at some time before 1954, and was first used in print in [Lemmon *et al.*, 1957, §9]. Meredith never published a formal statement of it, but his cousin David Meredith was taught the rule by him in 1954, wrote it up as a program, and ran it on the computer UNIVAC1 in the U.S.A. in 1957.[85] Thus, by 1957 the key step in a principal-type algorithm had not only been stated formally, but had been implemented on a machine.

Looking further back: an argument very like an analysis of a particular case of condensed detachment was used in [Łukasiewicz, 1939, Engl. edn., p.276]. Łukasiewicz attributed his method to Tarski, and in [Łukasiewicz and Tarski, 1930, Engl. edn., p.44] a method was mentioned which might perhaps be the one to which he referred. But the method was not actually described there, and "might perhaps" is conjecture not history. (See [Kalman, 1983, §4] for further comments.)

Other algorithms for the simple type theory of pure λ and CL include the *counting algorithm* made by Choukri-Bey Ben-Yelles in 1979 for counting and listing the normal inhabitants of a given type, see [Ben-Yelles, 1979, Ch. 3] or [Hindley, 1997,

[85] For a formal statement of the condensed detachment rule, see [Kalman, 1983, §2] or [Hindley, 1997, §7D]. For further historical information, see those references and [Meredith, 1977]. The rule has similarities with the well known resolution rule of J. A. Robinson, and pre-dates it.

Ch. 8].[86] Ben-Yelles' method was later modified by Sabine Broda and Luis Damas to give an algorithm which counts and lists just the principal normal inhabitants, [Broda and Damas, 1999].

Several algorithms for constructing a principal inhabitant of a type from any other inhabitant have also been published. The first was in [Hindley, 1969b, §6], for full λK. Others, which worked also for restricted sets of terms such as λI, were made by Robert Meyer in 1986, Grigori Mints and Tanel Tammet in 1988, and Norman Megill and Martin Bunder in 1994; see [Meyer and Bunder, 1988], [Meyer *et al.*, 1991, §4], [Mints and Tammet, 1991], [Megill and Bunder, 1996].

As David Meredith noted in [Meredith, 1980], such an algorithm provides a proof that the condensed detachment rule is complete for the weak implicational logic corresponding to the restricted λ-calculus in question.

9 Models for λ

9.1 Scott's D_∞ model

The first set-theoretical model for untyped λ was made by Dana Scott while he was in Oxford to work with Strachey during a leave from the Princeton Philosophy Department.[87] In [Scott, 1969c, p.51] he gives a precise date for the idea: 14 November 1969. Only one month before, he had strongly advocated a typed alternative to the (untyped) programming languages defined in [Böhm, 1966; Böhm and Gross, 1966] and [Landin, 1966b], in a privately circulated manuscript written for Strachey ([Scott, 1969e], mentioned in §8.1.2 above). His model was unexpected, and showed that despite his previous criticisms of untyped languages they could be given a mainstream mathematical sense. Further, the means he used for this turned out to be a natural outgrowth of the notions on which his earlier typed alternative had been based.

In the first versions of Scott's model construction, domains were complete lattices, [Scott, 1969b]. In a later account of the construction in [Scott, 1970c], he introduced *continuous lattices*, characterized algebraically as those complete lattices D where, for every $y \in D$,

$$y \;=\; \bigvee\{\bigwedge U \mid y \in U \text{ and } U \text{ is Scott-open and } U \subseteq D\},$$

or topologically as those T_0-spaces such that every continuous $f : X \to D$ from a subspace $X \subseteq Y$ can be extended to a continuous $\overline{f} : Y \to D$.[88] It was this extension property that made continuous lattices especially attractive for "a coherent theory of partial functions (in extension)" [Scott, 1973, p.178], and more precisely for "an extensional theory of functions, related directly to computability, where functions (rather than syntactic representations or programs for the functions) can be treated as objects in themselves" (Scott, in [Stoy, 1977, p.xxviii]).[89]

[86] An *inhabitant* of a type τ is any closed term M with type τ. It is *normal* iff it contains no β-redexes. It is *principal* iff τ is its principal type.

[87] The history of this collaboration, and the circumstances in which the model was invented, are given in Scott's foreword to [Stoy, 1977], in his Turing Award lecture [Scott, 1977], in [Scott, 2000], and in his comments in the 1993 published version of [Scott, 1969e].

[88] See [Scott, 1972] for details. Here $\bigvee$ and $\bigwedge$ are the usual lattice operators. A Scott-open subset U of a domain D is an upward closed set such that if $\bigvee \Delta \in U$ for a directed $\Delta \subseteq D$ then $U \cap \Delta \neq \emptyset$. A function $f : D \to D'$ is (Scott-) continuous iff it is monotonic and preserves the least upper bounds of directed subsets of D. An instance of this topology had already been considered in [Nerode, 1957; Nerode, 1959], and even before, in [Uspenskii, 1955]. The interplay of algebraic and topological notions has been a fruitful one in recent developments of domain theory, see §10 later.

[89] The theory of continuous lattices gained further impetus from their rediscovery and application in analysis, functional analysis and topological algebra, as documented in [Gierz *et al.*, 1980]. Their connection with *interval analysis* was pointed out in [Scott, 1970b, p.87].

From a global point of view, domains and Scott-continuous functions formed cartesian closed categories where the exponential objects $[D \to E]$ were the complete lattices of Scott-continuous functions from D to E: this made them suitable for the semantics of typed λ-calculus (see §8.1.3). But they also contained *reflexive* domains D_∞, that solved the equation $D_\infty = [D_\infty \to D_\infty]$ non-trivially (up to isomorphism), and these yielded models of the untyped λ-K-calculus with $\beta\eta$-conversion.

The insight that led Scott to the construction of reflexive domains consisted basically in extending to operations on domains, in particular $\mathcal{F}(D) = [D \to D]$, the iterative construction of the least fixed-point of a continuous function $f : D \to D$ as

$$\mathrm{fix}(f) \quad =_{\mathrm{def}} \quad \bigvee_{n\in\omega} f^n(\bot). \tag{1}$$

The technical device needed for this extension was later recognized by Scott himself [Scott, 1973, p.181] to be the most original step of his construction: given domains D and E, an *embedding-projection pair* from D to E is a pair of (Scott-) continuous functions $i : D \to E$, $\;j : E \to D$ such that $j \circ i = \mathrm{id}_D$ and $i \circ j \sqsubseteq \mathrm{id}_E$, where the order relation on functions is defined pointwise. Embedding-projection pairs defined an approximation relation on domains in analogy to the approximation relation on elements of domains described by the partial ordering. Reflexive domains were built as inverse limits of sequences of projections

$$\langle D_n \stackrel{j_n}{\leftarrow} D_{n+1}\rangle_{n\in\omega}$$

where D_0 was arbitrary and D_{n+1} was defined as $[D_n \to D_n]$, with a suitable choice of the initial embedding-projection pair $\langle i_0, j_0\rangle$. These spaces were "function-space analogues to the simple types of the Russell-Zermelo theory" [Scott, 1969b, p.24], and Scott always regarded his models as spaces of functions of infinite type, rather than of untyped functions, [Scott, 1975, pp.348–349].[90]

Soon after Scott's D_∞ models, several other models were invented by other workers, see 9.3 below, and there was intensive research on the structure of D_∞ and its application to the denotational semantics of programming languages. Probably the first result in this area was due to David Park [Park, 1976], who proved in early 1970 that the interpretation of the Rosenbloom-Curry combinator $\mathbf{Y}$ in D_∞ coincided with the least fixed point operator defined iteratively as in (1) above. On the negative side, Park also showed that this definability result could be ruined by a different choice of the initial embedding-projection pair $\langle i_0, j_0\rangle$ from D_0 to D_1.

9.2 Computational and denotational properties

A subject of extensive research in the early 1970s was the relation between the operational properties of λ-terms and their denotations as elements of some D_∞ model. A particularly interesting problem was to relate the lattice structure of the models, where the order relation among elements reflected their amount of information in an abstract sense (see, for example, [Scott, 1970c, p.171]), to the intuition that some terms are more informative than others in an operational sense. For example, there are λ-terms X such that, for all contexts $C[\ \ ]$ and all normal forms N,

$$\text{if } C[X] =_\beta N \;\text{ then }\; C[M] =_\beta N \text{ for every other term } M; \tag{2}$$

[90] Actually, in his first note on the construction, in [Scott, 1969a], Scott used direct limits of sequences of embeddings, obtaining an equivalent result. This was the first instance of what is now called the *limit-colimit coincidence* [Smyth and Plotkin, 1982], that lies at the heart of the categorical versions of the construction.

one of these is $\Delta\Delta$, where $\Delta \equiv \lambda x.xx$. Such terms contribute no information to the surrounding context $C[\ \ ]$, and could therefore be interpreted as the least element $\perp$ in any model. This problem was investigated thoroughly by Christopher Wadsworth, who was at the time a graduate student of Strachey at Oxford University, in his 1971 PhD thesis and two later papers, [Wadsworth, 1971; Wadsworth, 1976; Wadsworth, 1978].[91] He found a precise and easily grasped characterization of the sense in which reduction of a term M can be said to develop information out of better and better approximations of a normal form of M (if one exists). He extended the λ-calculus with a new constant Ω representing the undefined value, and defined an *approximate normal form* to be a term of the extended language (a λ-Ω-*term*) without β-redexes.[92] An approximate normal form A could then be said to match an ordinary λ-term except at occurrences of Ω in A, and this was exploited by Wadsworth in defining the set of *approximate normal forms* of M. One of his main results was the "limit theorem", [Wadsworth, 1976, Thm. 5.2], later called the "approximation theorem", [Barendregt, 1981, Thm. 19.2.2], which stated that the interpretation of a term M in Scott's D_∞ model is the least upper bound of the interpretations of its approximate normal forms.

A key technical concept throughout was that of a term M having a *head normal form*, that is, being convertible to a term of the form $\lambda x_1 \ldots x_n.zX_1 \ldots X_m$, for some $n, m \geq 0$. The identification of diverging computations with terms without a head normal form was found to be a natural one: for terms without head normal form, property (2) held, [Wadsworth, 1976, Cor. 5.5], and furthermore the interpretation of every such term in standard D_∞ models was the least element $\perp$ (ibid., Corollary 5.3).

Independently, Henk Barendregt, in June 1971 in his PhD thesis [Barendregt, 1971], proposed identifying diverging computations with what he called *unsolvable* terms, namely those λ-terms M such that

$$MX_1 \ldots X_k =_\beta \mathbf{I} \tag{3}$$

held for no sequence of terms $X_1, \ldots, X_k$ with $k \geq 0$. Barendregt was then attempting to construct a model for the λ-K-calculus based on recursion-theoretic notions, related to the model of hereditarily recursive operations for the simply typed λ-calculus described in [Kreisel, 1958].[93]

A natural problem in this setting was to find out whether non-normalizing terms like $\Delta\Delta$ were defined in the recursion-theoretic interpretation.[94] However, the notion of undefinedness induced by divergence of partial recursive functions turned out to be too strong, and the identification of undefinedness with unsolvability emerged through an analysis of terms that do not have a normal form, yet possess a meaningful computational behavior that prevents them from all being identified consistently.[95] The consistency of the theory resulting from the identification of all unsolvable terms was then proved by proof-theoretic means in [Barendregt, 1971,

[91] Another important contribution of Wadsworth dating back from the same period (October 1970) was the idea of *continuations* as a technique for the denotational semantics of jumps. Publication of this, as a joint report [Strachey and Wadsworth, 1974], was delayed because of work on his thesis. The history of his work on continuations can be found in [Wadsworth, 2000]. The notion of continuation was also invented by several other people at about the same time; see the historical account in [Reynolds, 1993].

[92] This extension was suggested to Wadsworth by Dana Scott in conversation, in October 1970, see [Wadsworth, 1976, fn.4, p.506].

[93] [Barendregt, 1996] gives a very clear account of this attempt and of the positive results arising from it. These include, in particular, the partial validity of the ω-rule (i.e. if $FZ = GZ$ for all closed terms Z then $F = G$) for $\lambda\beta\eta$-conversion, when F and G are not *universal generators* like those constructed later in [Plotkin, 1974].

[94] In this interpretation, Δ becomes a partial recursive function φ_e such that $\varphi_e(x) \simeq \varphi_x(x)$.

[95] An example is the pair of terms $\lambda x.x(\Delta\Delta)\mathbf{K}$ and $\lambda x.x(\Delta\Delta)(\mathbf{KI})$, whose identification would lead immediately to the equation $\mathbf{K} = \mathbf{KI}$.

Cor. 3.2.15 and Remark 3.2.16]. Also, by using unsolvable terms to represent undefined values $f(n)$ of a partial recursive function f, Barendregt obtained a simpler new proof that all partial recursive functions were λ-definable, [Barendregt, 1971, §1.3] or [Barendregt, 1981, §8.4].

It was Wadsworth (and Hyland, independently) who realized that a term M had no head normal form if and only if it was unsolvable [Wadsworth, 1971, p.94a], [Wadsworth, 1976, Cor. 5.3], allowing him to obtain a semantical version of Barendregt's consistency result by an immediate application of his limit theorem. (On the other hand, for the λI-calculus, Barendregt proved in 1973 that a term is unsolvable if and only if it has no normal form, [Barendregt, 1973].)

The interpretation of λ-terms in a D_∞ model induces a preorder on terms, defined by $M \leqslant N$ if and only if the interpretation of M is less than the interpretation of N for every choice of the interpretation of variables free in MN. The characterization of this preorder was given in [Wadsworth, 1976, Thm. 6.3] by introducing a notion of *semi-separability* of λ-terms, according to which M is semi-separable from N ($M \not\lesssim N$) iff there is a head context $C[\ \ ]$ such that $C[M] =_\beta \mathbf{I}$ but $C[N]$ is unsolvable. Then $M \not\lesssim N$ if and only if $M \not\leqslant N$. Independently, Hyland and Reiji Nakajima, then a student of Morris, obtained the same result for D_∞ [Hyland, 1975; Hyland, 1976; Nakajima, 1975], and Hyland found a similar characterization for the model $\mathcal{P}(\omega)$.[96] Behind all these studies lie, more or less explicitly, the techniques introduced by Böhm in the proof of his separability theorem [Böhm, 1968]: [Wadsworth, 1976] refers to a paper on "A general form of a theorem of Böhm and its application to Scott's models for the λ-calculus", that apparently was never published, while all the above results can be reformulated, following [Barendregt, 1977; Barendregt, 1981], using the notion of Böhm tree, cf. §7.1.

Jean-Jacques Lévy in his Thése d'Etat [Lévy, 1978] used $\lambda\Omega$-terms as the building blocks of an "algebraic" model for the λ-calculus, much in the tradition of the French school of algebraic semantics of recursive programs [Nivat, 1973; Vuillemin, 1974b].

At about this time the question arose of saying what exactly the phrase "model of λ" meant in general. This was examined in 1978 by Hindley and Giuseppe Longo, [Hindley and Longo, 1980, §§2–3] and further in 1980 by Albert Meyer [Meyer, 1982, §6]. There turned out to be several variant concepts: *λ-model*, *λ-algebra*, *combinatory algebra*, as well as a category-based definition (see [Koymans, 1982, §§3–4]).[97] Each of these has given rise to further studies, some of which will be mentioned below. Another definition, related to λ as cylindric and polyadic algebras are related to first-order predicate calculus, was that of *lambda abstraction algebra*, introduced by Don Pigozzi and Antonio Salibra (and independently by Z. B. Diskin) around 1993; see [Pigozzi and Salibra, 1998] for a good overview.

We should mention here that the informal set theory in which Scott built D_∞ corresponded to the standard Zermelo-Fraenkel system, in which self-membership and infinite descending $\in$-chains $x_1 \ni x_2 \ni x_3 \ni \ldots$ are forbidden by the axiom of foundation. But "non-well-founded" set theories have been proposed at various times, in which such chains are allowed, and in such a theory the construction of a model of λ is very much simpler to carry out. This was first done by Michael von Rimscha in 1978, [Rimscha, 1980]. Some comments on non-well-founded models are in [Plotkin, 1993, pp.375–377], and [Aczel, 1988] is a general account of non-well-founded set theories.

[96] Hyland played an important role in these developments: for example, in January 1972 he suggested the typed λ-calculus used in Wadsworth's proof of the approximation theorem, [Wadsworth, 1978]; see [Wadsworth, 1976, fn.5, p.508].

[97] A combined account is in [Barendregt, 1981, 2nd. edition, Ch. 5].

9.3 Other models

After Scott's construction of the D_∞ models, Plotkin observed in 1972 [Plotkin, 1993, Part I] that a suitable generalization of the notion of graph of a function yielded a new family of models, now collectively called the *graph models*. For this generalization, he took an arbitrary atom ι and defined inductively a set T_C by the clauses:

- $\iota \in T_C$,
- if μ, ν are finite subsets of T_C, then $\langle \mu, \nu \rangle \in T_C$.

Then the powerset $\mathcal{P}(T_C)$ became a (non-extensional) model of the λ-calculus when application was defined by

$$x \cdot y \ =_{\text{def}} \ \bigcup \{\nu \mid \exists \mu \subseteq y.\ \langle \mu, \nu \rangle \in x\}. \tag{4}$$

A similar idea had also occurred to Bob Meyer in 1973–'74, stimulated by some work of Larry Powers, and resulted in a graph model of CL that he called the *Fool's model*, [Meyer *et al.*, 1991, §5]. But this models reduction not equality. A related structure, for $\lambda\beta$-reduction, is described in [Plotkin, 1994, §4].

Other models in this family include those of [Scott, 1980a] and [Engeler, 1981], but the best-known of them is certainly $\mathcal{P}(\omega)$ of [Scott, 1976]. The model $\mathcal{P}(\omega)$ was motivated by "the project of making the connections with ordinary recursion theory easier to comprehend, since a satisfactory theory of computability and programming language semantics had to face this problem" [Scott, 1976, Appendix B]. The story of this model started at Easter 1973, when it was presented by Scott for the first time in Oberwolfach; later in the same year Scott realized that the idea behind it was essentially the same as Plotkin's definition (4), and that the same idea was "already implicit in a very precise form in much earlier work by Myhill-Shepherdson [Myhill and Shepherdson, 1955] and Friedberg-Rogers [Friedberg and Rogers, 1959] (see also [Rogers, 1967]) on *enumeration operators*" (Scott, ibid. p.576).[98]

The definition of application in $\mathcal{P}(\omega)$ needed the standard Cantor coding of pairs of natural numbers and an enumeration $\{e_n\}_{n\in\omega}$ of finite subsets of ω. Then the graph of a continuous function $f : \mathcal{P}(\omega) \to \mathcal{P}(\omega)$ on the (complete algebraic) lattice $\mathcal{P}(\omega)$ was defined by

$$\{\langle n, m \rangle \mid n \in f(e_m)\},$$

and (4) became, under coding,

$$x \cdot y \ =_{\text{def}} \ \{n \in \omega \mid \exists e_m \subseteq y.\ \langle m, n \rangle \in x\}.$$

One of the aspects of $\mathcal{P}(\omega)$ that turned out to be important, especially for later investigations on the semantics of type theories, was the peculiar semantics of types allowed by the properties of this model. A *closure* is a continuous function $a : \mathcal{P}(\omega) \to \mathcal{P}(\omega)$ such that $a(a(x)) = a(x) \supseteq x$ for all $x \in \mathcal{P}(\omega)$. Then a may be thought of as a type, namely the type of all $x \in \mathcal{P}(\omega)$ such that $a(x) = x$. For example, when a, b are closures, define $a \multimap b$ as $\lambda u.\, b \circ u \circ a$, obtaining thus a closure that represents the type of (continuous) functions from a to b. Furthermore, there is a closure $\mathbf{V} : \mathcal{P}(\omega) \to \mathcal{P}(\omega)$ such that $a \in \mathcal{P}(\omega)$ is a closure if and only if $\mathbf{V}(a) = a$. Such a $\mathbf{V}$ may be defined by $\mathbf{V} =_{\text{def}} \lambda a.\lambda x. \bigcup_{n=0}^{\infty} x^{(n)}$, where $x^{(0)} = \emptyset$ and $x^{(n+1)} = x \cup a(x^{(n)})$, or equivalently by:

$$\mathbf{V}(a)(x) \ =_{\text{def}} \ \bigcap \{y \in \mathcal{P}(\omega) \mid x \subseteq y \text{ and } a(y) \subseteq y\}.$$

[98] This story has had a recent – though implicit – continuation, in the study of *equilogical spaces* of [Scott, 1996], [Bauer *et al.*, 2004].

The lattice of closures therefore directly yields a model for a type system with a type of all types, like Martin-Löf's first version of intuitionistic type theory. The construction of **V** is due essentially to Martin-Löf and Peter Hancock (circa 1974), and was devised in the course of Martin-Löf's attempts to extend Kreisel's topological interpretation to type theory. This extension was carried out essentially by means of spaces described by formal neighborhoods, similar to the information systems studied later by Scott as the canonical description of domains [Scott, 1982a]. In order to handle universes, they needed a "space of spaces", for which the quantifiers and the type operations were continuous. The construction of that space was described by Hancock (then a PhD student in Oxford) to Scott, and he soon adapted it to $\mathcal{P}(\omega)$.

Therefore, the domain $\mathcal{P}(\omega)$ was also the first example of a *universal domain*, in two different senses: every countably based algebraic (continuous) lattice is the image of a closure (retraction) over $\mathcal{P}(\omega)$, where a *retraction* is a continuous endofunction of $\mathcal{P}(\omega)$ idempotent with respect to composition, see [Scott, 1976]. This result set up a powerful tool for solving recursive domain equations (up to equality) inside $\mathcal{P}(\omega)$: in particular, the model D_∞ is represented by the least fixed-point of the closure $\lambda a.\, a \circ\!\!\rightarrow a$ (see [Scott, 1976], and [Drakengren, 1996] who showed that this solution is unique, amending an incorrect statement in [Scott, 1976, p.553]).

These remarks gave rise to two different lines of research. On the one hand, there was a search for other similar universality results: progress in this area was based initially on modifications of the proofs for $\mathcal{P}(\omega)$, like Plotkin's universal domain $\mathbb{T}^\omega$ in [Plotkin, 1978b], or properties of the free Boolean algebra on $\aleph_0$ generators [Scott, 1982b]. Later it turned out that universal domains could be built uniformly by exploiting model-theoretic properties [Gunter, 1987; Gunter and Jung, 1990], in particular the amalgamation property [Droste and Göbel, 1993], that allowed homogeneous universal domains to be constructed for several classes of domains, unique up to isomorphism. On the other hand, the flexibility of retracts and related classes of endofunctions over universal domains in modelling type constructors suggested their use for interpreting higher-order type systems, especially the polymorphic λ-calculus. Types were interpreted as closures over $\mathcal{P}(\omega)$ in [McCracken, 1979], as retracts in [McCracken, 1982], and as finitary projections in [Amadio *et al.*, 1986]. Barendregt and Rezus also used closures for the semantics of Automath and related systems with dependent types [Barendregt and Rezus, 1983].

10 Domain theory

10.1 Classical domain theory

It is not completely clear what we can take as the starting date of a *theory* of domains, but by 1978 there was already a considerable bulk of results on categories of domains, many of them due to the efforts of Gordon Plotkin and collected in his notes for a lecture course in Pisa ([Plotkin, 1978a], complemented later by the notes [Plotkin, 1981]). The guiding analogy in that presentation was that between the complete partial order structure of domains and the approximation structure on domains induced by embedding-projection pairs. The iterative construction of the least fixed-point of a continuous endofunction of a domain corresponded in this analogy to Scott's inverse limit construction; in the meantime the latter had been given a categorical description through the contributions of Reynolds, Wand, Smyth and Plotkin himself [Reynolds, 1972; Wand, 1974; Wand, 1977; Wand, 1979; Smyth and Plotkin, 1982].[99]

[99] Scott had already mentioned in [Scott, 1972, pp.128-129] a suggestion of Lawvere to the effect that the inverse limit construction of D_∞ could be carried out by regarding $[D \to D']$ as a functor

However, the main impetus to the study of the structure of domains was given not so much by the models of untyped λ as by the full abstraction problem for the typed λ-system PCF, the logic-free part of Milner's LCF ([Milner, 1972] and §8.1.2 above). This problem was described in 1975 in two papers, [Milner, 1977] and [Plotkin, 1977],[100] and concerns the relations between operational and denotational equivalence on programs.[101] Operationally, we say that two terms M and N of the same type σ are *observationally equivalent*, written $M \approx N$, when, for every context $C[\ \ ]$ with one hole of type σ such that $C[M]$ and $C[N]$ are programs, we have that $C[M] \Downarrow \mathbf{v}$ if and only if $C[N] \Downarrow \mathbf{v}$. This relation was first introduced in [Milner, 1975] in the context of a semantics of parallel computation, although it owed much to [Morris, 1968] and the early studies of the denotational properties of untyped λ-terms by Hyland, Wadsworth and Plotkin described in §9.2 above. Writing $[\![P]\!]$ for the denotation of a program P in some type structure, we say that programs P, Q are *equivalent* iff $[\![P]\!] = [\![Q]\!]$, and a denotational semantics is *fully abstract* [Milner, 1977] when $[\![P]\!] = [\![Q]\!]$ is equivalent to $P \approx Q$ for all PCF programs P, Q.

Plotkin showed that the standard model of PCF based on domains and Scott-continuous functions was not fully abstract [Plotkin, 1977, pp.234-236], but that it became so if one added parallel conditional operators $:\supset_\sigma$ of type $o \to (\sigma \to (\sigma \to \sigma))$, where $\sigma = o$ or $\sigma = \iota$. The interpretation of $:\supset_\sigma$ was the following continuous function:

$$:\supset_\sigma\ (p)(x)(y) \ = \ \begin{cases} x & \text{if } p = \mathbf{true} \\ y & \text{if } p = \mathbf{false} \\ x & \text{if } x = y \text{ and } p = \bot_o \\ \bot_\sigma & \text{if } x \neq y \text{ and } p = \bot_o. \end{cases}$$

On the other hand, [Milner, 1977, p.19, Cor. 3] proved that there is a unique fully abstract model for PCF (up to isomorphism); he constructed it essentially as a (quotiented) term model. The full abstraction problem consists basically in the search for a syntax-independent description of Milner's model.[102]

The failure of full abstraction for the semantics of PCF based on Scott continuous functions is due to the presence of parallel functions, like the "*parallel or*" connective $\mathsf{V} : o \to (o \to o)$, already considered in [Platek, 1966, pp.127–131] and discussed there and in [Scott, 1969e], defined by the truth-table

V	$\bot$	**false**	**true**
$\bot$	$\bot$	$\bot$	**true**
false	$\bot$	**false**	**true**
true	**true**	**true**	**true**.

This fact drew attention to the problem of developing a semantic notion of *sequentiality* in order to "define a 'smaller' collection of domains containing only functions

(over the category of continuous lattices with projections as morphisms) whose second argument preserved inverse limits and whose first argument turned direct into inverse limits. The proof that $D_\infty \cong [D_\infty \to D_\infty]$ was then a direct consequence of these facts. A more general categorical framework for the solution of recursive domain equations was later described in [Freyd, 1990; Freyd, 1991; Freyd, 1992].

[100] See also the independent work carried out in the same year by Vladimir Yu. Sazonov and described in [Sazonov, 1976c; Sazonov, 1976b; Sazonov, 1976a].

[101] By a *program* in this context is meant a closed PCF-term of type either o, with *values* **true** and **false**, or ι, whose values are the numerals $\mathbf{n}$ for each natural number n. If a program M reduces to a value $\mathbf{v}$, we write $M \Downarrow \mathbf{v}$.

[102] We do not pursue further the technical aspects of full abstraction, although they determined much of the course of research on domains from 1975 onwards, for at least 20 years after. A comprehensive account of the strategies for solving it (and of the technical advances produced by those attempts) is in [Ong, 1995]. The solutions of the full abstraction problem obtained independently by Abramsky, Jagadeesan and Malacaria and Hyland and Ong, rely on a *game semantics* for PCF whose applications are still being investigated, [Abramsky *et al.*, 1994; Hyland and Ong, 2000]. See below, §10.3.

capable of deterministic realization" [Plotkin, 1977, p.236]. Definitions of sequential function had already been given in [Milner, 1977, p.20] and [Vuillemin, 1974a, §3.3]. However, these were coextensive only for Scott-continuous functions defined over flat domains (i. e., those where every chain had at most two elements) and depended upon a specific way of choosing the number of arguments of functions. A more general notion of sequentiality arose from the theory of *concrete domains* introduced in the Fall of 1975 by Gilles Kahn and Gordon Plotkin in [Kahn and Plotkin, 1978], where they axiomatized an abstract notion of "argument place" of a function by means of *information matrices* (or *concrete data structures*), over which computation proceeds by occurrence of *events* that consist in filling *cells* by *values*, subject to an *accessibility* relation that constrains the order in which cells are able to be filled. A domain is generated by an information matrix as the collection of its *configurations*, where each configuration is a set of events that represents the history of a computation over the matrix.

Though the general definition of sequential function based on concrete domains did not appear in [Kahn and Plotkin, 1978], the idea circulated widely, especially in the group of young French researchers at Sophia-Antipolis (including Gérard Berry and Pierre-Louis Curien). In [Berry, 1978], Berry used the algebraic model of [Lévy, 1976] to show that computation in pure untyped λ-calculus was essentially sequential in the sense of Kahn and Plotkin, extending Lévy's continuity theorem (these results are described in detail in [Barendregt, 1981, §§14.3–4]).

Later, Berry and Curien [Curien, 1979; Berry and Curien, 1982] introduced the notion of *sequential algorithm* over concrete data structures as a semantical framework for PCF. While sequential algorithms are not functions in the set-theoretic sense, they do form a cartesian closed category and therefore provide a categorical model for PCF [Berry, 1981; Curien, 1986].[103] The calculations needed in the study of the category of concrete data structures and sequential algorithms also led Curien to introduce *categorical combinators* [Curien, 1985]; these became important in computational applications, starting from [Curien, 1986] (see also [Huet, 1990a]) who used them as instructions of an abstract machine [Cousineau *et al.*, 1987] for the implementation of the functional programming language ML.

A determinism condition on the computation of continuous functions that was close to sequentiality was devised by Berry, who called it *stability*, [Berry, 1976; Berry, 1978]. Stable functions satisfy a minimum data property whereby, for every finite approximation e of a value $f(x)$ there exists a minimum $d \sqsubseteq x$ such that $e \sqsubseteq f(d)$. Observe that the "parallel or" function is not stable. A more algebraic description of stability was possible in most cases of interest [Berry, 1976, Prop.II.2.1], just by saying that a Scott-continuous function is stable iff $f(x \wedge y) = f(x) \wedge f(y)$ provided x and y have a common upper bound. At higher orders, stable functions are ordered by $f \leq_{st} g$ iff $f(x) \sqsubseteq f(y) \wedge g(x)$ whenever $x \sqsubseteq y$; and with this definition, stable functions on appropriate domains (e.g. the dI-domains of [Berry, 1978, Def. 4.3.2], satisfying some of the properties of concrete domains) yield cartesian closed categories and, therefore, models of PCF.

The notion of stability was later rediscovered from a different point of view by [Girard, 1986]. Since the mid-1970s, Girard had been developing a theory of ordinals whose emphasis was on their geometric structure. The basic idea was that ordinal operations can be first defined on natural numbers and then extended to the whole class of ordinals by regarding it as a category whose objects are ordinals and whose morphisms are strictly increasing functions. The essential technical device to achieve this program was the notion of *dilator* [Girard, 1981], namely an endofunctor of the category of ordinals that preserves directed colimits and pullbacks.

[103] There is a large literature on categorical models of typed and untyped λ-calculus, stimulated by the work of Lambek and also by [Scott, 1980b]: see [Koymans, 1982].

When specialised to domains regarded as categories, the preservation properties of dilators defined exactly the stable functions, and this was the starting point of the development of *qualitative domains* in [Girard, 1986] and [Girard, 1988, Annex A], and their simplification as *coherence spaces* in [Girard, 1986]. These yielded interesting models of his System $\mathbb{F}$ in which types were interpreted as domains. They also suggested the analysis of the (stable) function type $D \to_{st} E$ as $!D \multimap E$, where ! was a unary constructor on coherence spaces and $\multimap$ was *linear implication*, that would become the basic connective of *linear logic* [Girard, 1986], a logical system whose importance for the whole field of the semantics of computation is only beginning to be fully appreciated.

The representation of concrete domains as sets of configurations of information matrices was the central result in the monograph [Kahn and Plotkin, 1978, §7]. This suggested similar representation theorems for other categories of domains. In his lecture course at Merton College Oxford in 1981 [Scott, 1982b], Scott represented (Scott-) domains as the partially ordered sets of filters of structures that described the basis of a domain and therefore determined its whole partial order structure (*neighbourhood systems*).[104] The next year, [Scott, 1982a] gave an equivalent representation of domains in terms of *information systems*: these were essentially abstract intuitionistic sequent calculi, which determined a set of *elements* (i.e., the elements of the represented domains) identified with Horn theories.

Another approach to the finitary representation of domains arose from an unexpected connection between Scott domains and the filter models built by means of intersection types (§8.4). The idea of using types to build models was not new, for example it was the main motivation of Plotkin's 1972 report [Plotkin, 1993, Part I, p.365].[105] Extending the ideas that led to the filter model of [Barendregt *et al.*, 1983], Coppo, Dezani, Honsell and Longo proved a precise correspondence between filter models built from what they called *extended applicative type structures* in [Coppo *et al.*, 1983] and [Coppo *et al.*, 1984], generalizing the set of intersection types, and the applicative information systems that form a subclass of the information systems used in [Scott, 1982a]. This was also recognized by Scott (ibid.), who remarked that intersection types could be regarded as just another representation of the finite elements of a domain. Filter models built from intersection types with additional constants – essentially a reformulation in this new context of [Coppo and Dezani, 1978] – were then used in [Coppo *et al.*, 1987] to build a model for untyped λ which was isomorphic to a non-standard D_∞ whose theory coincided with Morris' extensional theory [Morris, 1968].[106]

In the same years, Mike Smyth (and also Plotkin, in unpublished work) proposed looking at the open subsets of a topological space as computable properties of its points ([Smyth, 1983]; [Plotkin, 1981, Ch. 8, Exs.94–96]). Specialized to the Scott topology of a domain, this view complemented the identification of continuity and (abstract) computability suggested originally in [Scott, 1969e; Scott, 1969d] and [Plotkin, 1978a, §1]. This led to further representation results for domains, as characterized by the structure of the complete lattices of their open subsets. These results (thoroughly surveyed in [Abramsky and Jung, 1994]) were in effect an extension of the representation theory for Boolean algebras via Stone duality [Johnstone, 1982], influenced by closely related results in the theory of continuous

[104] The *basis* of a domain D is the set of its finite elements, with the partial order inherited from D, where $d \in D$ is *finite* (or *algebraic*) iff $d \sqsubseteq \bigvee \Delta$ for some directed $\Delta \subseteq D$ implies $d \sqsubseteq e$ for some $e \in \Delta$. A cpo D is *algebraic* iff every $d \in D$ is the least upper bound of the directed set of finite elements below it, and a *Scott domain* is an algebraic cpo with a countable basis and least upper bounds of upper bounded subsets. This definition goes back to [Scott, 1969d].

[105] In fact, the graph models and filter models are presented together in [Plotkin, 1993, Part II, p.373ff.].

[106] The connections between filter models and the inverse limit construction have been explored in [Coppo *et al.*, 1984] and in [Plotkin, 1993].

lattices [Gierz *et al.*, 1980, §V.5] and the domain interpretation of Martin-Löf's type theory [Martin-Löf, 1983; Martin-Löf, 1986]. They were exploited in a logic of observable properties of domains first described systematically in [Vickers, 1989] and [Abramsky, 1991], and opened the way to the application of point-less topology in a computational setting, which was seen as very desirable from a constructive standpoint [Johnstone, 1982; Fourman and Grayson, 1982; Coquand, 1992, p.119], and led to the point-free study of domain theory within Martin-Löf's intuitionistic type theory [Sambin, 1987].

10.2 Effective domains and Synthetic Domain Theory

We have remarked that computability considerations were at the basis of the very notions of domain and continuous function. Soon after those workers involved came to see algebraic and consistently complete partial orders as a convenient class of domains, much research was devoted to studying computability theory over *effectively given* domains; these allowed researchers to consider computable constructions in the concrete sense of classical recursion theory, [Constable and Egli, 1974], [Egli and Constable, 1976], [Smyth, 1977], [Kanda, 1979], [Plotkin, 1981, Ch. 7], [Scott, 1982b, Lecture VII]. Early ideas in this area had been put forward by Yu. Ershov using the theory of numbered sets proposed by Malcev, where a numbered set A consists of a set A with a surjective coding function e_A from A onto ω. In particular, in Ershov's framework the notion of *constructive domain* was defined, and it was proved that the Scott topology on such domains coincided with a topology that was induced in a natural way by its numbering (the *Malcev-Ershov topology*, see [Giannini and Longo, 1984; Rosolini, 1986] and the survey in [Longo, 1987]).

However, the resulting theory was still complicated by the need to manipulate the codes of elements of domains explicitly. A proposal to overcome these complications was put forward by Dana Scott, as the program of *synthetic domain theory*. A synthetic approach to computation consists in the study of constructive universes in which it is possible to regard domains just as special kinds of sets.[107] The name was suggested by an analogy with Synthetic Differential Geometry, where generalized manifolds are manipulated as sets of a special kind. Scott proposed such a research program in a talk at the Peripatetic Seminar on Sheaves and Logic in Sussex University in 1980, and later described its aim as that of having "a universe that combines logic and mathematics in a natural way and permits a development of finitary recursion theory in a sufficiently abstract way" [Scott, 1986].

The need for a treatment of partialness within intuitionistic universes of sets, described as toposes, led Scott's student Giuseppe Rosolini to introduce the basic notion in synthetic domain theory, namely that of *dominance*, a subset Σ of the set Ω of truth values that classifies the domains of definition of partial functions [Rosolini, 1986]. Intuitively, the elements of Σ are interpretations of the Σ^0_1 propositions in Ω. An important example of dominance arises in the effective topos $\mathcal{E}\!\mathit{ff}$ of [Hyland, 1982], where all functions on natural numbers are recursive, by setting

$$\Sigma \quad = \quad \{p \in \Omega \mid \ \exists f \in \mathbb{N}^{\mathbb{N}}.\ p \leftrightarrow (\exists z.\, f(z) = 0)\},$$

as was first done by Scott in a lecture in Pisa in the Fall of 1983. By exploiting this notion it was possible to define subcategories of the ambient topos whose objects might be taken as "Scott domains", and to prove for them the basic properties that categories of domains must have in order to support the denotational semantics of programming languages. The work on effective domains was therefore

[107] Our short account of synthetic domain theory is based on the introduction to Rosolini's PhD thesis [Rosolini, 1986], on [Hyland, 1991], and on the section on synthetic domain theory by Moggi and Rosolini in [Jung (editor), 1996].

subsumed under the synthetic approach. Early contributions to this area include the work of Wesley Phoa [Phoa, 1990] on complete Σ-spaces as a synthetic counterpart to (pre)domains, and the alternative approach of [Freyd *et al.*, 1990]; and the axiomatic presentation of the fundamental ideas in [Hyland, 1991] and [Taylor, 1991]. Even more important, in demonstrating the unification achieved by means of a synthetic approach within the effective topos, was the discovery by Moggi and Hyland mentioned in §8.3 above that the category of modest sets yields a model of the polymorphic λ-calculus.

10.3 Game semantics

The full abstraction problem for PCF, that oriented much research on domain theory since the mid-70s [Milner, 1977; Plotkin, 1977], was given a solution by two independent teams: Abramsky, Jagadeesan and Malacaria from Imperial College, London [Abramsky *et al.*, 1994], and Hyland and Ong from Cambridge. Both groups announced their result in a message to the `types` mailing list on July 27th, 1993. Both used *games* as the semantic counterparts of types, and interpreted typed λ-terms as *strategies*, achieving what is called an *intensionally fully abstract* model for PCF, namely an algebraic model where every isolated element is definable.[108] The use of games (and strategies) did not come abruptly out of the blue, however, and has in fact a long history in the semantics of logical systems; under the correspondence between propositions and types, part of that tradition can be seen to be directly relevant to typed λ-calculi. From our point of view, the starting point of game semantics can be found in the works of Paul Lorenzen and his school from 1961 onward on dialogue games formalizing a debate between a Proponent and an Opponent, where the Opponent tries to attack a first-order formula defended by the Proponent [Lorenzen, 1961; Felscher, 1986; Lorenz, 2001]. The game-theoretical notion of validity was intended by Lorenzen to coincide with intuitionistic validity; a connection with typed λ-calculus was not made, even implicitly. This had to wait until the early 1990s, when it was exploited in the context of a semantical analysis of Girard's Linear Logic. The first mention of games in the context of linear logic goes back to a paper by Yves Lafont and Thomas Streicher [Lafont and Streicher, 1991], where a game was seen as a structure $\langle A^\star,\ A_\star,\ e : (A^\star \times A_\star) \to K\rangle$ consisting of two sets with a "payoff" function valued in a set K. A little earlier, Valeria de Paiva had given a related categorical account of Gödel's Dialectica interpretation [Paiva, 1989a; Paiva, 1989b] and had used it to interpret linear logic.[109]

The Lorenzen tradition was then brought into the semantics of linear logic by Andreas Blass in [Blass, 1992], building on his previous work on determinacy of (infinite) games [Blass, 1972]. With Blass' paper the use of dialogue games and strategies became a powerful tool for model construction, and in fact he was able to obtain a completeness theorem for the additive fragment of linear logic with respect to his games, stating that a sequent is provable if and only if there is a winning strategy for Proponent in the associated game.[110] One important drawback of this

[108] A quotient of the game model then yielded the order-extensional fully abstract model for PCF, as announced in a further message by Abramsky to the `types` mailing list on September 8th of the same year. It can be argued that the way to a direct (i.e., not quotiented) construction of the fully abstract model for PCF is barred by the undecidability of observational equivalence, shown by [Loader, 2001].

[109] Both these approaches are related to a method for building $*$-autonomous categories from suitably complete symmetric monoidal categories, invented by Po-Hsiang Chu and described in [Barr, 1979, Appendix]. [Seely, 1989] observed that $*$-autonomous categories yield categorical models for linear logic (see also [Barr, 1991]).

[110] Blass' model actually validates the Weakening rule (from $\Gamma \vdash B$ deduce $\Gamma, A \vdash B$), and is therefore a model of what is called *affine* logic.

model was that composition of strategies was not associative, hence there was no category of games arising from Blass' work. On the other hand, a monoidal closed category of games (in the sense of [Conway, 1976]) had been already studied by André Joyal in [Joyal, 1977], who had noticed that it was a natural context for a "combinatorial" calculus of strategies. A category of games suitable for interpreting the multiplicative fragment of linear logic was introduced in 1992 by [Abramsky and Jagadeesan, 1994], who proved what they called a "full completeness" theorem: proofs in the multiplicative fragment of linear logic plus the MIX rule

$$\frac{\vdash \Gamma \quad \vdash \Delta}{\vdash \Gamma, \Delta}$$

correspond bijectively to winning strategies.[111] The perfect harmony of semantical objects (strategies) and syntactical structures (proofs) established by the full completeness theorem was exactly the kind of correspondence needed for building the fully abstract model of PCF of [Abramsky *et al.*, 1994], which in fact owed much to the framework set up for obtaining that theorem. In addition, the interpretation of composition of strategies as "parallel composition plus hiding" suggested by Abramsky [Abramsky, 1994; Abramsky and Jagadeesan, 1994] opened the way to the application of game semantics and ideas from linear logic to the theory of concurrent processes, especially process algebras; [Abramsky, 1994; Abramsky, 2000].[112]

Less closely related to the developments in linear logic was the approach of Hyland and Ong [Hyland and Ong, 2000]. They too used games, but their approach was more directly influenced by attempts to find a suitable notion of sequentiality at higher types, much in the same line as the theory of sequentiality of [Kahn and Plotkin, 1978] and [Berry and Curien, 1982], see [Ong, 1995], and by the compositional approach to games arising from [Blass, 1972] and [Joyal, 1977]. (In later presentations of game semantics, however, the technical framework set up by Hyland and Ong was used to build categories of games suitable for modelling (some fragment of) linear logic; see e.g., [Abramsky and McCusker, 1998].)

In the same direction, and with a bias towards the problem of extending Church's thesis to higher-type computability, there was already in the 1980s a series of papers by Kleene, who used dialogue games to interpret his higher-type recursion schemes [Kleene, 1978; Kleene, 1980; Kleene, 1982; Kleene, 1985]. Kleene's problem, the attempt to find "a class of functions which shall coincide with all the partial functions which are 'computable' or 'effectively decidable', so that Church's 1936 Thesis will apply with the higher types included" [Kleene, 1978, §1.2], led him to consider, though implicitly, the full abstraction problem for PCF.[113] Kleene's ideas were then pursued by Gandy and his student Giovanni Pani for continuous functionals [Gandy, 1993]. They emphasized the conditions of "no dangling question mark" for the game-theoretic analysis of computability in PCF, and this influenced, through informal discussions with Hyland, the formulation of the games of Hyland and Ong. Also seeming to be rooted in Kleene's work were the games used by [Nickau, 1994], that were in fact of the same kind as those of Hyland and Ong.

[111] This kind of result can be traced back to Läuchli's completeness theorem for his interpretation of intuitionistic logic [Läuchli, 1965; Läuchli, 1970].

[112] The research in this direction was strongly influenced by Girard's idea of a *geometry of interaction* [Girard, 1987b; Girard, 1989a; Girard, 1989b], where the correctness criterion for (multiplicative) proof-nets – the natural deduction formulation of (the multiplicative fragment of) linear logic, [Girard, 1987a] – is expressed in terms of geometrical properties of graphs [Danos and Regnier, 1989].

[113] Kleene's work was also independent from the parallel work on characterizations of sequentiality at higher-types by means of concrete data structures as in [Berry and Curien, 1982], that have been shown to have substantial game theoretic content, e.g., by François Lamarche [Lamarche, 1992] and by Pierre-Louis Curien [Curien, 1994], [Amadio and Curien, 1998, §14.3].

Beside the applications of game semantics to the interpretation of programming languages (surveyed, for example, in [Abramsky and McCusker, 1998]), there are developments of this area that pertain closely to the foundations of logic. An important example is *ludics*, a research program started by Girard in the late 1990s [Girard, 1998; Girard, 2000; Girard, 2001]. On a generically philosophical side, ludics is motivated by a critical revision of the traditional relations between syntax and semantics of logic, with a special emphasis on the meaning of logical rules: according to [Girard, 1998, p.215], this "is to be found in the well-hidden geometrical structure of the rules themselves: typically, negation should [be interpreted] by the exchange between *Player* and *Opponent*". The resulting emphasis on the symmetries of computation, already brought to the fore by linear logic, currently inspires much of the research at the border between logic and computation.

But this belongs to the 21st century, so our history stops here.

References

[Abadi *et al.*, 1991] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1:375–416, 1991.

[Abadi *et al.*, 1993] M. Abadi, L. Cardelli, and P.-L. Curien. Formal parametric polymorphism. *Theoretical Computer Science*, 121:9–58, 1993.

[Abdali, 1976] S. K. Abdali. An abstraction algorithm for combinatory logic. *Journal of Symbolic Logic*, 41:222–224, 1976.

[Abramsky and Jagadeesan, 1994] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59:543–574, 1994.

[Abramsky and Jung, 1994] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, Oxford, England, 1994.

[Abramsky and McCusker, 1998] S. Abramsky and G. McCusker. Game semantics. In U. Berger and H. Schwichtenberg, editors, *Computational Logic*, pages 1–56. Springer-Verlag, Berlin, 1998.

[Abramsky *et al.*, 1994] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (extended abstract). In H. Hagiya and J. C. Mitchell, editors, *Theoretical Aspects of Computer Software. International Symposium TACS '94*, volume 789 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, Berlin, 1994.

[Abramsky, 1991] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[Abramsky, 1994] S. Abramsky. Proofs as processes. *Theoretical Computer Science*, 135:5–9, 1994.

[Abramsky, 2000] S. Abramsky. Process realizability. In F. L. Bauer and R. Steinbrüggen, editors, *Foundations of Secure Computation*, pages 167–180. IOS Press, Amsterdam, 2000.

[Aczel and Feferman, 1980] P. Aczel and S. Feferman. Consistency of the unrestricted abstraction principle using an intensional equivalence operator. In Hindley and Seldin [1980], pages 67–98.

[Aczel, 1980] P. Aczel. Frege structures and the notions of proposition, truth and set. In J. Barwise, H. J. Keisler, and K. Kunen, editors, *The Kleene Symposium*, pages 31–59. North-Holland Co., Amsterdam, 1980.

[Aczel, 1988] P. Aczel. *Non-Well-Founded Sets*. CSLI (Centre for the Study of Language and Information), Ventura Hall, Stanford Univ., Stanford, CA 94305-4115, U.S.A., 1988.

[Ajdukiewicz, 1935] K. Ajdukiewicz. Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27, 1935. English transl: *Syntactic Connexion* in *Polish Logic 1920–1939*, ed. by Storrs McCall, Clarendon Press, Oxford, 1967, pp. 207–231.

[Amadio and Curien, 1998] R. Amadio and P.-L. Curien. *Domains and Lambda-Calculi*, volume 46 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, England, 1998.

[Amadio *et al.*, 1986] R. Amadio, K. Bruce, and G. Longo. The finitary projection model for second order lambda calculus and solutions to higher order domain equations. In *Proceedings First Annual IEEE Symposium on Logic In Computer Science*, pages 122–130. IEEE Computer Society, Los Alamitos, California, U.S.A., 1986.

[Anderson, 1998] C. A. Anderson. Alonzo Church's contributions to philosophy and intensional logic. *Bulletin of Symbolic Logic*, 4:129–171, 1998.

[Andrews, 1965] P. B. Andrews. *A Transfinite Type Theory with Type Variables*. North-Holland Co., Amsterdam, 1965.

[Andrews, 1971] P. B. Andrews. Resolution in type theory. *Journal of Symbolic Logic*, 36:414–432, 1971.

[Asperti and Longo, 1991] A. Asperti and G. Longo. *Categories, Types and Structures. An Introduction to Category Theory for the Working Computer Scientist*. Foundations of Computing. M.I.T. Press, Cambridge, Mass., U.S.A., 1991.

[Asperti, 1995] A. Asperti. Linear logic, comonads and optimal reductions. *Fundamenta Informaticae*, 22(1–2):3–22, 1995.

[Backus, 1978] J. Backus. Can programming be liberated from the von Neumann style? *Communications of the ACM*, 21(8):613–641, 1978.

[Bar-Hillel, 1953] Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.

[Bar-Hillel, 1959] Y. Bar-Hillel. Decision procedures for structure in natural languages. *Logique et analyse*, 2:19–29, 1959.

[Barendregt and Rezus, 1983] H. P. Barendregt and A. Rezus. Semantics for classical AUTOMATH and related systems. *Information and Control*, 59:127–147, 1983. (Journal now called *Information and Computation*).

[Barendregt *et al.*, 1976] H. P. Barendregt, J. Bergstra, J. W. Klop, and H. Volken. Representability in lambda algebras. *Koninkl. Nederlands Akad. van Wetensch. Proc. Ser. A*, 79:377–387, 1976. Vol. also publ. as *Indagationes Mathematicae* 38.

[Barendregt *et al.*, 1983] H. P. Barendregt, M. Coppo, and M. Dezani. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48:931–940, 1983.

[Barendregt *et al.*, 1993] H. Barendregt, M. Bunder, and W. Dekkers. Systems of illative combinatory logic complete for first order propositional and predicate calculus. *Journal of Symbolic Logic*, 58:769–788, 1993.

[Barendregt, 1971] H. P. Barendregt. *Some extensional term models for combinatory logics and λ-calculi*. PhD thesis, Univ. Utrecht, Netherlands, 1971.

[Barendregt, 1973] H. P. Barendregt. A characterization of terms of the λ-I-calculus having a normal form. *Journal of Symbolic Logic*, 38:441–445, 1973.

[Barendregt, 1976] H. P. Barendregt. A global representation of the recursive functions in the λ-calculus. *Theoretical Computer Science*, 3:225–242, 1976.

[Barendregt, 1977] H. P. Barendregt. The type-free lambda calculus. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 1091–1132. North-Holland Co., Amsterdam, 1977.

[Barendregt, 1981] H. P. Barendregt. *The Lambda Calculus, its Syntax and Semantics*. North-Holland Co., Amsterdam, 1981. 2nd (revised) edn. 1984.

[Barendregt, 1993] H. P. Barendregt. Constructive proofs of the range property in lambda calculus. *Theoretical Computer Science*, 121 (Böhm Volume):59–69, 1993.

[Barendregt, 1996] H. P. Barendregt. Kreisel, lambda calculus, a windmill and a castle. In P. G. Odifreddi, editor, *Kreiseliana, about and around Georg Kreisel*, pages 3–14. A. K. Peters, Wellesley, Massachusetts, 1996.

[Barendregt, 1997] H. P. Barendregt. The impact of the lambda calculus in logic and computer science. *Bulletin of Symbolic Logic*, 3:181–215, 1997.

[Barr, 1979] M. Barr. *$*$-autonomous Categories*, volume 752 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1979.

[Barr, 1991] M. Barr. $*$-autonomous categories and linear logic. *Mathematical Structures in Computer Science*, 1:159–178, 1991.

[Bates and Constable, 1985] J. L. Bates and R. L. Constable. Proofs as programs. *ACM Transactions on programming Languages and Systems*, 7(1):113–136, 1985.

[Bauer *et al.*, 2004] A. Bauer, L. Birkedal, and D. S. Scott. Equilogical spaces. *Theoretical Computer Science*, 315:35–59, 2004.

[Ben-Yelles, 1979] C-B. Ben-Yelles. *Type-assignment in the Lambda-calculus*. PhD thesis, Math. Dept., Univ. Wales Swansea, Swansea SA2 8PP, U.K., 1979.

[Bénabou, 1985] J. Bénabou. Fibered categories and the foundations of naive category theory. *Journal of Symbolic Logic*, 50:10–37, 1985.

[Benaissa *et al.*, 1996] Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. λv, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6:699–722, 1996.

[Bergstra and Klop, 1980] J. Bergstra and J. W. Klop. Invertible terms in the lambda calculus. *Theoretical Computer Science*, 11:19–37, 1980.

[Bernays and Schönfinkel, 1928] P. Bernays and M. Schönfinkel. Zum Entscheidungsproblem der mathematischen Logik. *Mathematische Annalen*, 99:342–372, 1928.

[Bernays, 1958] P. Bernays. Über eine natürliche Erweiterung des Relationenkalkuls. In A. Heyting, editor, *Constructivity in Mathematics*, pages 1–14. North-Holland Co., Amsterdam, 1958.

[Berry and Curien, 1982] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–321, 1982.

[Berry, 1976] G. Berry. Bottom-up computation of recursive programs. *RAIRO Informatique Théorique et Applications*, 10:47–82, 1976.

[Berry, 1978] G. Berry. Stable models of typed λ-calculi. In G. Ausiello and C. Böhm, editors, *Automata, Languages and Programming, Fifth Colloquium*, volume 62 of *Lecture Notes in Computer Science*, pages 72–89. Springer-Verlag, Berlin, 1978.

[Berry, 1981] G. Berry. On the definition of λ-calculus models. In J. Diaz and I. Ramos, editors, *International Colloquium on Formalization of Programming Concepts, Proceedings*, volume 107 of *Lecture Notes in Computer Science*, pages 218–230. Springer-Verlag, Berlin, 1981.

[Blass, 1972] A. Blass. Degrees of indeterminacy of games. *Fundamenta Mathematicae*, 77:151–166, 1972.

[Blass, 1992] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56(1–3):183–220, 1992.

[Bloo and Geuvers, 1999] R. Bloo and J. H. Geuvers. Explicit substitution: on the edge of strong normalization. *Theoretical Computer Science*, 211:375–395, 1999.

[Böhm and Berarducci, 1985] C. Böhm and A. Berarducci. Automatic synthesis of typed λ-programs on term algebras. *Theoretical Computer Science*, 39:135–154, 1985.

[Böhm and Dezani, 1972] C. Böhm and M. Dezani. A CUCH machine: The automatic treatment of bound variables. *International Journal of Computer and Information Sciences*, 1:171–191, 1972. See also *Notes on "A CUCH machine: the automatic treatment of bound variables"*, *ibid* 2 (1973), 157–160.

[Böhm and Dezani, 1974] C. Böhm and M. Dezani. A parenthesis machine for string manipulation. *Revue Français d'Automatique, Informatique et Recherche Opérationelle*, 8:37–46, 1974.

[Böhm and Dezani, 1975] C. Böhm and M. Dezani. λ-terms as total or partial functions on normal forms. In Böhm [1975], pages 96–121.

[Böhm and Dezani, 1989] C. Böhm and M. Dezani. Combinatory logic as monoids. *Fundamenta Informaticae*, 12(4):525–540, 1989.

[Böhm and Giovannucci, 1964] C. Böhm and R. Giovannucci. Circuiti sequenziali e analogici e loro descrizione mediante il CUCH. Istituto per le Applicazioni del Calcolo, C.N.R., Roma, 1964.

[Böhm and Gross, 1966] C. Böhm and W. Gross. Introduction to the CUCH. In E. Caianiello, editor, *Automata Theory*, pages 35–65. Academic Press, New York, 1966. (Preprints available 1964).

[Böhm *et al.*, 1977] C. Böhm, M. Coppo, and M. Dezani. Termination tests inside λ-calculus. In A. Salomaa and M. Steinby, editors, *Automata, Languages and Programming, Fourth Colloquium*, volume 52 of *Lecture Notes in Computer Science*, pages 95–110. Springer-Verlag, Berlin, 1977.

[Böhm, 1954] C. Böhm. Calculatrices digitales. Du déchiffrage des formules logico-mathématiques par la machine même dans la conception du programme. *Annali di Matematica Pura e Applicata*, 37(4):1–51, 1954.

[Böhm, 1966] C. Böhm. The CUCH as a formal and description language. In T. B. Steel, Jr., editor, *Formal Language Description Languages for Computer Programming*, pages 179–197. North-Holland Co., Amsterdam, 1966. Proc. conference in 1964.

[Böhm, 1968] C. Böhm. Alcune proprietà delle forme β-η-normali nel λ-K-calcolo. Pubblicazione no. 696, Istituto per le Applicazioni del Calcolo, C.N.R., Roma, 1968.

[Böhm, 1975] C. Böhm, editor. *λ-Calculus and Computer Science Theory*, volume 37 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1975.

[Böhm, 1979] C. Böhm. Un modèle arithmétique des termes de la logique combinatoire. In B. Robinet, editor, *Lambda Calcul et Sémantique Formelle des Langages de Programmation. Actes de la Sixième Ecole de Printemps d'Informatique Theorique, La Châtre, 1978*, pages 97–108. LITP et ENSTA, Paris, 1979.

[Böhm, 1980] C. Böhm. Logic and computers. In E. Agazzi, editor, *Modern Logic — A Survey*, pages 297–309. D. Reidel Co., Dordrecht, Netherlands, 1980.

[Böhm, 1982] C. Böhm. Combinatory foundations of functional programming. In *Proceedings of the 1982 ACM Symposium on Lisp and Functional Programming*, pages 29–36. Association for Computing Machinery, New York, 1982.

[Böhm, 1986] C. Böhm. Reducing recursion to iteration by algebraic extension. In B. Robinet and R. Wilhelm, editors, *European Symposium on Programming*, volume 213 of *Lecture Notes in Computer Science*, pages 111–118. Springer-Verlag, Berlin, 1986.

[Böhm, 1988a] C. Böhm. Functional programming and combinatory algebras. In M. P. Chytil, L. Janiga, and V. Koubek, editors, *Mathematical Foundations of Computer Science*, volume 324 of *Lecture Notes in Computer Science*, pages 14–26. Springer-Verlag, Berlin, 1988.

[Böhm, 1988b] C. Böhm. Reducing recursion to iteration by means of pairs and n-tuples. In M. Boscarol, L. Carlucci Aiello, and G. Levi, editors, *Foundations of Logic and Functional Programming*, volume 306 of *Lecture Notes in Computer Science*, pages 58–66. Springer-Verlag, Berlin, 1988. Proc. workshop in 1986.

[Böhm, 1989] C. Böhm. Subduing self-application. In G. Ausiello, M. Dezani, and S. Ronchi della Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium*, volume 372 of *Lecture Notes in Computer Science*, pages 108–122. Springer-Verlag, Berlin, 1989.

[Broda and Damas, 1999] S. Broda and L Damas. Counting a type's principal inhabitants. In J.-Y. Girard, editor, *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99*, volume 1581 of *Lecture Notes in Computer Science*, pages 69–82. Springer-Verlag, Berlin, 1999. Fuller version: *Fundamenta Informaticae* 45 (2001), 33–51.

[Bruce *et al.*, 1990] K. Bruce, A. A. Meyer, and J. C. Mitchell. The semantics of second-order lambda calculus. In Huet [1990b], pages 273–284.

[Bruijn, 1970] N. G. de Bruijn. The mathematical language AUTOMATH. In M. Laudet, D. Lacombe, L. Nolin, and M. Schützenberger, editors, *Symposium on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*, pages 29–61. Springer-Verlag, Berlin, 1970.

[Bruijn, 1972] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation. *Indagationes Mathematicae*, 34:381–392, 1972.

[Bruijn, 1980] N. G. de Bruijn. A survey of the project AUTOMATH. In Hindley and Seldin [1980], pages 579–606.

[Bruijn, 1994] N. G. de Bruijn. Reflections on Automath. In Nederpelt et al. [1994], pages 201–228.

[Bunder *et al.*, 2003] M. W. Bunder, W. Dekkers, and H. Geuvers. Equivalences between pure type systems and systems of illative combinatory logic. *Journal of Symbolic Logic*, 2003.

[Bunder, 1983a] M. W. Bunder. A one axiom set theory based on higher order predicate calculus. *Archiv für Mathematische Logik*, 23:99–107, 1983.

[Bunder, 1983b] M. W. Bunder. Predicate calculus of arbitrarily high finite order. *Archiv für Mathematische Logik*, 23:1–10, 1983.

[Bunder, 1983c] M. W. Bunder. Set theory in predicate calculus with equality. *Archiv für Mathematische Logik*, 23:109–113, 1983.

[Burali-Forti, 1894] C. Burali-Forti. *Logica Matematica*. Manuali Hoepli. U. Hoepli, Milano, 1894.

[Burge, 1978] W. H. Burge. *Recursive Programming Techniques*. Addison-Wesley, Reading, Mass., U.S.A., 1978.

[Burstall, 1977] R. M. Burstall. Design considerations for a functional programming language. In *The Software Revolution; Proceedings of the Infotech State of the Art Conference, Copenhagen 1977*, pages 45–57. Infotech and Pergamon Press, U.K., 1977.

[Cardelli and Wegner, 1985] L. Cardelli and P. Wegner. On understanding types, data abstraction and polymorphism. *ACM Computing Surveys*, 17:471–522, 1985.

[Cardone and Coppo, 1991] F. Cardone and M. Coppo. Type inference with recursive types. Syntax and Semantics. *Information and Computation*, 92(1):48–80, 1991.

[Chauvin, 1979] A. Chauvin. Theory of objects and set theory: introduction and semantics. *Notre Dame Journal of Formal Logic*, 10:37–54, 1979. (English transl. of part of thesis *Théorie des Objets et Théorie des Ensembles*, Université de Clermont-Ferrand, France, 1974.).

[Cheatham Jr. *et al.*, 1968] T. E. Cheatham Jr., P. Fischer, and P. Jorrand. On the basis of ELF – An extensible language facility. In *AFIPS Conference Proceedings, San Francisco, California*, pages 937–948. Thompson Book Co., Washington, D.C., 1968.

[Church and Rosser, 1936] A. Church and J. B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39:472–482, 1936.

[Church, 1932] A. Church. A set of postulates for the foundation of logic. *Annals of Mathematics, Series 2*, 33:346–366, 1932.

[Church, 1933] A. Church. A set of postulates for the foundation of logic (second paper). *Annals of Mathematics, Series 2*, 34:839–864, 1933.

[Church, 1935] A. Church. A proof of freedom from contradiction. *Proceedings of the National Academy of Sciences of the U.S.A.*, 21:275–281, 1935.

[Church, 1936a] A. Church. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1:40–41, 1936. See also correction in pp. 101–102.

[Church, 1936b] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[Church, 1937] A. Church. Combinatory logic as a semi-group. *Bulletin of the American Mathematical Society*, 43:333, 1937. Only an abstract.

[Church, 1940] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Church, 1941] A. Church. *The Calculi of Lambda Conversion*. Princeton Univ. Press, Princeton, New Jersey, U.S.A., 1941. Reprinted 1951 and 2000.

[Church, 1951a] A. Church. A formulation of the logic of sense and denotation. In P. Henle, H. Kallen, and S. Langer, editors, *Structure, Method and Meaning, Essays in Honor of Henry M. Sheffer*, pages 3–24. Liberal Arts Press, New York, 1951. See also abstract with same title, in J. Symbolic Logic 11 (1946), p. 31.

[Church, 1951b] A. Church. The weak theory of implication. In A. Menne, A. Wilhelmy, and H. Angstl, editors, *Kontrolliertes Denken, Untersuchungen zum Logikkalkül und zur Logik der Einzelwissenschaften (Festgabe zum 60. Geburtstag von Prof. W. Britzelmayr)*, pages 22–37. Verlag Karl Alber, München, 1951. (Sonderdruck).

[Church, 1956] A. Church. *Introduction to Mathematical Logic*. Princeton Univ. Press, Princeton, New Jersey, U.S.A., 1956.

[Church, 1964] A. Church, 7 July 1964. Unpublished letter to Harald Dickson.

[Church, 1973] A. Church. Outline of a revised formulation of the logic of sense and denotation, Part I. *Noûs*, 7:24–33, 1973.

[Chwistek, 1922] L. Chwistek. Über die Antinomien der Prinzipen der Mathematik. *Mathematische Zeitschrift*, 14:236–243, 1922.

[Constable and Egli, 1974] R. L. Constable and H. Egli. Computability on continuous higher types and its role in the semantics of programming languages. Technical report TR-74-209, Department of Computer Science, Cornell Univ. , Ithaca, New York, 1974.

[Constable and O'Donnell, 1978] R. L. Constable and M. J. O'Donnell. *A Programming Logic*. Winthrop, Cambridge, 1978.

[Constable and others, 1986] R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Englewood Cliffs, N.J., U.S.A., 1986.

[Constable, 1971] R. L. Constable. Constructive mathematics and automatic program writers. In *Proceedings of the IFIP Congress*, pages 229–233, Ljubljana, 1971.

[Constable, 1980] R. L. Constable. Programs and types. In *21st IEEE Symposium on the Foundations of Computer Science*, pages 118–128, Los Alamitos, California, U.S.A., 1980. IEEE Computer Society.

[Constable, 1982] R. L. Constable. Programs as proofs. Technical report TR-82-532, Dept. of Computer Science, Cornell Univ. , Ithaca, New York, 1982.

[Constable, 1985] R. L. Constable. Constructive mathematics as a programming logic I: Some principles of theory. *Annals of Discrete Mathematics*, 24:21–38, 1985.

[Conway, 1976] J. H. Conway. *On Numbers and Games*, volume 6 of *London Mathematical Society Monographs*. Academic Press, London, 1976.

[Coppo and Dezani, 1978] M. Coppo and M. Dezani. A new type assignment for λ-terms. *Archiv für Mathematische Logik*, 19:139–156, 1978.

[Coppo and Ferrari, 1993] M. Coppo and A. Ferrari. Type inference, abstract interpretation and strictness analysis. *Theoretical Computer Science*, 121:113–143, 1993.

[Coppo *et al.*, 1979] M. Coppo, M. Dezani, and P. Sallé. Functional characterization of some semantic equalities inside λ-calculus. In H. Maurer, editor, *Automata, Languages and Programming, Sixth Colloquium*, volume 71 of *Lecture Notes in Computer Science*, pages 133–146. Springer-Verlag, Berlin, 1979.

[Coppo *et al.*, 1980] M. Coppo, M. Dezani, and B. Venneri. Principal type-schemes and λ-calculus semantics. In Hindley and Seldin [1980], pages 535–560.

[Coppo *et al.*, 1981] M. Coppo, M. Dezani, and B. Venneri. Functional characters of solvable terms. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 27:45–58, 1981.

[Coppo *et al.*, 1983] M. Coppo, M. Dezani, F. Honsell, and G. Longo. Applicative information systems. In G. Ausiello and M. Protasi, editors, *Colloquium on Trees and Algebra in Programming*, volume 159 of *Lecture Notes in Computer Science*, pages 33–64. Springer-Verlag, Berlin, 1983.

[Coppo *et al.*, 1984] M. Coppo, M. Dezani, F. Honsell, and G. Longo. Extended type structures and filter lambda models. In G. Lolli, G. Longo, and A. Marcja, editors, *Logic Colloquium '82*, pages 241–262. North-Holland Co., Amsterdam, 1984.

[Coppo *et al.*, 1987] M. Coppo, M. Dezani, and M. Zacchi. Type theories, normal forms and D_∞-lambda models. *Information and Computation*, 72:85–116, 1987.

[Coppo, 1985] M. Coppo. A completeness theorem for recursively defined types. In W. Brauer, editor, *Automata, Languages and Programming, 12th International Colloquium*, volume 194 of *Lecture Notes in Computer Science*, pages 120–129, Berlin, 1985. Springer-Verlag.

[Coquand and Huet, 1985] T. Coquand and G. Huet. Constructions: a higher order proof system for mechanizing mathematics. In B. Buchberger, editor, *EUROCAL'85, Proceedings Volume 1*, volume 203 of *Lecture Notes in Computer Science*, pages 151–184, Berlin, 1985. Springer-Verlag.

[Coquand and Huet, 1988] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[Coquand, 1985] T. Coquand. Une théorie des constructions. Thèse de Troisième Cycle, Université Paris VII, 1985.

[Coquand, 1986] T. Coquand. An analysis of Girard's paradox. In *Proceedings First Annual IEEE Symposium on Logic In Computer Science*, pages 227–236. IEEE Computer Society, Los Alamitos, California, U.S.A., 1986.

[Coquand, 1990] T. Coquand. Metamathematical investigations of a calculus of constructions. In P. Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Studies in Data Processing*, pages 91–122. Academic Press, London, 1990.

[Coquand, 1992] T. Coquand. An intuitionistic proof of Tychonoff's theorem. *Journal of Symbolic Logic*, 57:28–32, 1992.

[Coquand, 1999] T. Coquand. Inductive definitions and type theory: an introduction, 1999. Preliminary draft for August 1999 TYPES summer school.

[Cousineau *et al.*, 1987] G. Cousineau, P.-L. Curien, and M. Mauny. The categorical abstract machine. *Science of Computer Programming*, 8:173–202, 1987.

[Craig, 1974] W. Craig. *Logic in an Algebraic Form, Three Languages and Theories*. North-Holland Co., Amsterdam, 1974.

[Crossley, 1975] J. N. Crossley. Reminiscences of Logicians. In J. Crossley, editor, *Algebra and Logic*, volume 450 of *Lecture Notes in Mathematics*, pages 1–62. Springer-Verlag, Berlin, 1975.

[Curien *et al.*, 1996] P.-L Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the Association for Computing Machinery*, 43:362–397, 1996.

[Curien, 1979] P.-L. Curien. Algorithmes sequentiels sur structures de donnees concretes. In B. Robinet, editor, *Lambda Calcul et Sémantique Formelle des Langages de Programmation. Actes de la Sixième Ecole de Printemps d'Informatique Théorique*, pages 157–182. LITP et Ecole Nationale Supérieure de Techniques Avancées, 1979.

[Curien, 1985] P.-L. Curien. Typed categorical combinatory logic. In H. Ehrig, C. Floyd, M. Nivat, and J. Thatcher, editors, *Mathematical Foundations of Software Development, Proceedings 1985, Volume 1, CAAP'85*, volume 185 of *Lecture Notes in Computer Science*, pages 157–172. Springer-Verlag, Berlin, 1985.

[Curien, 1986] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Pitman (London) and Wiley (New York), 1986. (2nd edn. 1993 publ. by Birkhäuser, U.S.A.).

[Curien, 1994] P.-L. Curien. On the symmetry of sequentiality. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics, 9th International Conference*, volume 802 of *Lecture Notes in Computer Science*, pages 29–71. Springer-Verlag, Berlin, 1994.

[Curry and Feys, 1958] H. B. Curry and R. Feys. *Combinatory Logic, Volume I*. North-Holland Co., Amsterdam, 1958. (3rd edn. 1974).

[Curry *et al.*, 1972] H. B. Curry, J. R. Hindley, and J. P. Seldin. *Combinatory Logic, Volume II*. North-Holland Co., Amsterdam, 1972.

[Curry, 1927] H. B. Curry. *Schönfinkel, M. Über die Bausteine der mathematischen Logik*, 1927. Note dated Nov. 28th. 1927, 3 pages, filed as T271128A in Curry archive, Pennsylvania State Univ., U.S.A. Reproduced on website of Southern Alberta Digital Library, http:// www.sadl.uleth.ca.

[Curry, 1930] H. B. Curry. Grundlagen der kombinatorischen Logik. *American Journal of Mathematics*, 52:509–536, 789–834, 1930.

[Curry, 1932] H. B. Curry. Some additions to the theory of combinators. *American Journal of Mathematics*, 54:551–558, 1932.

[Curry, 1934a] H. B. Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences of the U.S.A.*, 20:584–590, 1934.

[Curry, 1934b] H. B. Curry. Some properties of equality and implication in combinatory logic. *Annals of Mathematics, Series 2*, 35:849–860, 1934.

[Curry, 1936] H. B. Curry. First properties of functionality in combinatory logic. *Tôhoku Mathematical Journal*, 41:371–401, 1936.

[Curry, 1937] H. B. Curry. Review of Church's lecture notes *Mathematical Logic* given at Princeton Univ. in 1936. *Journal of Symbolic Logic*, 2:39–40, 1937.

[Curry, 1942a] H. B. Curry. The combinatory foundations of mathematical logic. *Journal of Symbolic Logic*, 7:49–64, 1942.

[Curry, 1942b] H. B. Curry. The inconsistency of certain formal logics. *Journal of Symbolic Logic*, 7:115–117, 1942.

[Curry, 1948] H. B. Curry. The logical structure of grammar, 1948. Unfinished manuscript mimeographed at Univ. Chicago and privately distributed, from a lecture given in Nov. 1948.

[Curry, 1949] H. B. Curry. A simplification of the theory of combinators. *Synthese*, 7:391–399, 1949.

[Curry, 1951] H. B. Curry. *Outlines of a Formalist Philosophy of Mathematics*. North-Holland Co., Amsterdam, 1951.

[Curry, 1954] H. B. Curry. The logic of program composition. In *Applications Scientifiques de la Logique Mathématique, Actes du Deuxieme Colloque International de Logique Mathématique, Paris 1952*, pages 97–102. Gauthier-Villars, Paris, 1954.

[Curry, 1961] H. B. Curry. Some logical aspects of grammatical structure. In R. Jakobson, editor, *Structure of Language and its Mathematical Aspects*, number 12 in Proceedings of Symposia in Applied Mathematics, pages 56–68. American Mathematical Society, Providence, R.I., U.S.A., 1961.

[Curry, 1963] H. B. Curry. *Foundations of Mathematical Logic*. McGraw-Hill, New York, 1963. Reprinted 1977 by Dover, Inc., New York.

[Curry, 1964] H. B. Curry. The elimination of variables by regular combinators. In M. Bunge, editor, *The Critical Approach to Science and Philosophy*, pages 127–143. Free Press, U.S.A., 1964.

[Curry, 1966] H. B. Curry. Technique for evaluating principal functional character, 1966. Note dated March 17th. 1966, 5 pages, filed as T660317A in Curry archive, Pennsylvania State Univ. , U.S.A.

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

[Curry, 1969b] H. B. Curry. The undecidability of λK-conversion. In J. Bulloff, T. Holyoke, and S. Hahn, editors, *Foundations of Mathematics, Symposium Papers Commemorating the Sixtieth Birthday of Kurt Gödel*, pages 10–14. Springer-Verlag, Berlin, 1969.

[Curry, 1980] H. B. Curry. Some philosophical aspects of combinatory logic. In J. Barwise, H. J. Keisler, and K. Kunen, editors, *The Kleene Symposium*, pages 85–101. North-Holland Co., Amsterdam, 1980.

[Daalen, 1973] D. T. van Daalen. A description of Automath and some aspects of its language theory. In P. Braffort, editor, *Proceedings of the Symposium APLASM*, volume I, Orsay, 1973. Also in [Nederpelt *et al.*, 1994], Item A.3 pp. 101–126.

[Daalen, 1980] D. T. van Daalen. *The Language Theory of AUTOMATH.* PhD thesis, Technische Hogeschool Eindhoven, Netherlands, 1980. Main parts also in [Nederpelt *et al.*, 1994], Items A.6 (pp.163–200), B.6 (pp.303–312), C.5 (pp.493–654).

[Damas and Milner, 1982] L. Damas and R. Milner. Principal type-schemes for functional programming languages. In *Ninth Annual A.C.M. Symposium on the Principles of Programming Languages (POPL)*, pages 207–212. Association for Computing Machinery, New York, 1982.

[Damas, 1984] L. Damas. *Type Assignment in Programming Languages.* PhD thesis, Computer Science Dept., Univ. Edinburgh, U.K., 1984.

[Danos and Regnier, 1989] V. Danos and L. Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[Davis, 1982] M. Davis. Why Gödel didn't have Church's Thesis. *Information and Control*, 54:3–24, 1982. Journal now called *Information and Computation.*

[Dekkers *et al.*, 1998] W. Dekkers, M. W. Bunder, and H. Barendregt. Completeness of the propositions-as-types interpretation of intuitionistic logic into illative combinatory logic. *Journal of Symbolic Logic*, 63:869–890, 1998.

[Dezani and Ermine, 1982] M. Dezani and F. Ermine. Maximal monoids of normal forms. *Fundamenta Informaticae*, 2:130–141, 1982.

[Dezani *et al.*, 1993] M. Dezani, S. Ronchi della Rocca, and M. Venturini Zilli, editors. *A Collection of Contributions in Honour of Corrado Böhm*, 1993. *Theoretical Computer Science* special issue Vol. 121.

[Dezani, 1975] M. Dezani. A type theory for λ-β-normal forms. Proceedings of the Symposium Informatica 75, Bled, Slovenia, 1975.

[Dezani, 1976] M. Dezani. Characterization of normal forms possessing inverse in the λ-β-η-calculus. *Theoretical Computer Science*, 2:323–337, 1976.

[Diller, 1968] J. Diller. Zur Berechenbarkeit primitiv-rekursiver Funktionale endlicher Typen. In H. A. Schmidt, editor, *Contributions to Mathematical Logic*, pages 109–120. North-Holland Co., Amsterdam, 1968. Proc. conf. Hannover 1966.

[Dragalin, 1968] A. Dragalin. The computation of primitive recursive terms of finite type, and primitive recusive realization. *Zapiski Naucnyh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta imeni V.A. Steklova, Akademii Nauk S.S.S.R. (L.O.M.I.)*, 8:32–45, 1968. In Russian.

[Drakengren, 1996] T. Drakengren. Uniqueness of Scott's reflexive domain in $\mathbf{P}\omega$. *Theoretical Computer Science*, 155:267–276, 1996.

[Droste and Göbel, 1993] M. Droste and R. Göbel. Universal domains and the amalgamation property. *Mathematical Structures in Computer Science*, 3:137–159, 1993.

[Dybyg, 1996] K. Dybyg. *The Scheme Programming Language*. Prentice-Hall, U.S.A., 1996.

[Egli and Constable, 1976] H. Egli and R. Constable. Computability concepts for programming language semantics. *Theoretical Computer Science*, 2:133–145, 1976.

[Eilenberg and Kelly, 1966] S. Eilenberg and G. M. Kelly. Closed categories. In *Proceedings of the Conference on Categorical Algebra, La Jolla, 1965*, pages 412–562. Springer-Verlag, Berlin, 1966.

[Enderton, 1995] H. B. Enderton. In memoriam: Alonzo Church 1903–1995. *Bulletin of Symbolic Logic*, 1:486–488, 1995.

[Enderton, 1998] H. B. Enderton. Alonzo Church and the Reviews. *Bulletin of Symbolic Logic*, 4:172–181, 1998.

[Engeler, 1981] E. Engeler. Algebras and combinators. *Algebra Universalis*, 13:389–392, 1981.

[Feferman, 1975a] S. Feferman. A language and axioms for explicit mathematics. In J. Crossley, editor, *Algebra and Logic*, volume 450 of *Lecture Notes in Mathematics*, pages 87–139. Springer-Verlag, Berlin, 1975.

[Feferman, 1975b] S. Feferman. Non-extensional type-free theories of partial operations and classifications, I. In J. Diller and G. Müller, editors, *Proof Theory Symposion, Kiel 1974*, volume 500 of *Lecture Notes in Mathematics*, pages 73–118. Springer-Verlag, Berlin, 1975.

[Feferman, 1977] S. Feferman. Categorical foundations and foundations of category theory. In R. Butts and J. Hintikka, editors, *Logic, Foundations of Mathematics and Computability Theory*, pages 149–169. D. Reidel Co., Dordrecht, Netherlands, 1977.

[Feferman, 1984] S. Feferman. Towards useful type-free theories, I. *Journal of Symbolic Logic*, 49:75–111, 1984. No Part II has been published.

[Felleisen *et al.*, 1987] M. Felleisen, D. Friedman, E. Kohlbecker, and B. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52:205–237, 1987.

[Felscher, 1986] W. Felscher. Dialogues as a foundation for intuitionistic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 3, pages 341–372. D. Reidel Co., Dordrecht, Netherlands, 1986.

[Feys, 1946a] R. Feys. La technique de la logique combinatoire. *Revue Philosophique de Louvain*, 44:74–103, 237–270, 1946.

[Feys, 1946b] R. Feys. Les méthodes récentes de déduction naturelle. *Revue Philosophique de Louvain*, 44:370–400, 1946.

[Fitch, 1936] F. B. Fitch. A system of formal logic without an analogue to the Curry W operator. *Journal of Symbolic Logic*, 1:92–100, 1936.

[Fitch, 1952] F. B. Fitch. *Symbolic Logic, an Introduction*. The Ronald Press, New York, 1952.

[Fitch, 1958] F. B. Fitch. Representation of sequential circuits in combinatory logic. *Philosophy of Science*, 25:263–279, 1958. Talk given 1957.

[Fitch, 1963] F. B. Fitch. The system $C\Delta$ of combinatory logic. *Journal of Symbolic Logic*, 28:87–97, 1963.

[Fitch, 1974] F. B. Fitch. *Elements of Combinatory Logic*. Yale Univ. Press, New Haven, U.S.A., 1974.

[Forster, 1995] T. L. Forster, editor. *Set Theory with a Universal Set*. Clarendon Press, Oxford, 1995. 1st edn. was 1992.

[Fourman and Grayson, 1982] M. Fourman and R. Grayson. Formal spaces. In A. S. Troelstra and D. van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium*, pages 107–122. North-Holland Co., Amsterdam, 1982.

[Fox, 1966] L. Fox, editor. *Advances in Programming and Non-numerical Computation*. Pergamon Press, Oxford, 1966. Proceedings of a 1963 summer school.

[Frege, 1879] G. Frege. *Begriffsschrift*. Verlag Louis Nebert, Halle, 1879. Reprinted in *Begriffsschrift und Andere Aufsätze*, ed. by I. Angelelli, publ. 1964 by Georg Olms, Hildesheim, Germany. Engl. transl: *Begriffschrift*, in [Heijenoort, 1967], pp. 1–82.

[Frege, 1891] G. Frege. Funktion und Begriff. Vortrag Gehalten in der Sitzung vom 9. Januar 1891 der Jenaischen Gesellschaft für Medizin und Naturwissenschaft, 1891. English transl: *Function and Concept* in *Translations from the Philosophical Writings of Gottlob Frege*, ed. P. Geach and M. Black, publ. Blackwell, Oxford, England, 1960 (2nd edn.), pp. 21–41. Same transl. also in *Gottlob Frege Collected Papers on Mathematics, Logic and Philosophy*, ed. B. McGuinness, Blackwell 1984, pp. 137–156.

[Frege, 1893] G. Frege. *Grundgesetze der Arithmetik*. Verlag Hermann Pohle, Jena, 1893. Two vols. Reprinted as one vol. in 1962 by Georg Olms, Hildesheim, Germany, and in 1966 as No. 32 in series *Olms Paperbacks*. Partial English transl. in *The Basic Laws of Arithmetic, Exposition of the System*, ed. M. Furth, Univ. California Press, Berkeley, U.S.A., 1964.

[Freyd *et al.*, 1990] P. Freyd, P. Mulry, G. Rosolini, and D. Scott. Extensional PERs. In *Proceedings Fifth Annual IEEE Symposium on Logic In Computer Science*, pages 346–354. IEEE Computer Society, Los Alamitos, California, U.S.A., 1990.

[Freyd, 1990] P. J. Freyd. Recursive types reduced to inductive types. In *Proceedings Fifth Annual IEEE Symposium on Logic In Computer Science*, pages 498–507. IEEE Computer Society, Los Alamitos, California, U.S.A., 1990.

[Freyd, 1991] P. J. Freyd. Algebraically complete categories. In A. Carboni, M. Pedicchio, and G. Rosolini, editors, *Proceedings of the 1990 Como Category Theory Conference*, volume 1488 of *Lecture Notes in Mathematics*, pages 131–156. Springer-Verlag, 1991.

[Freyd, 1992] P. J. Freyd. Remarks on algebraically compact categories. In M. Fourman, P. Johnstone, and A. Pitts, editors, *Applications of Categories in Computer Science*, volume 177 of *London Math. Soc. Lecture Notes Series*, pages 95–106. Cambridge Univ. Press, 1992.

[Friedberg and Rogers, 1959] R. M. Friedberg and H. Rogers. Reducibilities and completeness for sets of integers. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 5:117–125, 1959.

[Friedman, 1975] H. Friedman. Equality between functionals. In R. Parikh, editor, *Logic Colloquium, Symposium on Logic held at Boston, 1972–73*, volume 453 of *Lecture Notes in Mathematics*, pages 22–37. Springer-Verlag, Berlin, 1975.

[Friedman, 1978] H. Friedman. Classically and intuitionistically provably recursive functions. In G. Müller and D. Scott, editors, *Higher Set Theory*, volume 669 of *Lecture Notes in Mathematics*, pages 21–27. Springer-Verlag, 1978.

[Gallier, 1990] J. H. Gallier. On Girard's "Candidats de reductibilité". In P. Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Studies in Data Processing*, pages 123–203. Academic Press, London, 1990.

[Gandy, 1977] R. O. Gandy. The simple theory of types. In R. Gandy and M. Hyland, editors, *Logic Colloquium '76*, pages 173–181. North-Holland Co., Amsterdam, 1977.

[Gandy, 1988] R. O. Gandy. The confluence of ideas in 1936. In R. Herken, editor, *The Universal Turing Machine: A Half-Century Survey*, pages 51–102. Oxford Univ. Press, England, 1988. (2nd edn. 1995, publ. Springer-Verlag).

[Gandy, 1993] R. O. Gandy. Dialogues, Blass games, sequentiality for objects of finite type. Unpublished manuscript, 1993.

[Gentzen, 1935] G. Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English transl: *Investigations into logical deduction*, in *The Collected Papers of Gerhard Gentzen*, ed. by M. Szabo, North-Holland Co., Amsterdam, 1969.

[Giannini and Longo, 1984] P. Giannini and G. Longo. Effectively given domains and lambda calculus semantics. *Information and Control*, 62:36–63, 1984. Journal now *Information and Computation*.

[Gierz *et al.*, 1980] G. Gierz, K. Hofmann, K. Keimel, J. Lawson, M. Mislove, and D. Scott. *A Compendium of Continuous Lattices*. Springer-Verlag, Berlin, 1980.

[Gilmore, 1963] P. C. Gilmore. An abstract computer with a LISP-like machine language without a label operator. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 71–86. North-Holland Co., Amsterdam, 1963.

[Girard *et al.*, 1989] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge Univ. Press, England, 1989.

[Girard, 1971] J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 63–92. North-Holland Co., Amsterdam, 1971.

[Girard, 1972] J.-Y. Girard. Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur. Thèse de Doctorat d'État, Univ. Paris VII, Place Jussieu, Paris, 1972.

[Girard, 1981] J.-Y. Girard. Π^1_2-logic, part I: dilators. *Annals of Mathematical Logic*, 21:75–219, 1981.

[Girard, 1986] J.-Y. Girard. The system F of variable types, fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986.

[Girard, 1987a] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Girard, 1987b] J.-Y. Girard. Multiplicatives. *Rendiconti del Seminario Matematico dell' Università e del Politecnico di Torino*, Fascicolo speciale: "Logic and Computer Science: New Trends and Applications":11–34, 1987. Edited by G. Lolli.

[Girard, 1988] J.-Y. Girard. Normal functors, power series and λ-calculus. *Annals of Mathematical Logic*, 37:129–177, 1988.

[Girard, 1989a] J.-Y. Girard. Geometry of interaction I: interpretation of system F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Logic Colloquium '88*, pages 221–260. North-Holland Co., Amsterdam, 1989.

[Girard, 1989b] J.-Y. Girard. Towards a geometry of interaction. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 69–108. American Mathematical Society, Providence, Rhode Island, 1989.

[Girard, 1998] J.-Y. Girard. On the meaning of logical rules I: syntax vs. semantics. In U. Berger and H. Schwichtenberg, editors, *Computational Logic*, pages 215–272. Springer-Verlag, Berlin, 1998. Proc. of meeting 1997.

[Girard, 2000] J.-Y. Girard. On the meaning of logical rules II: multiplicative/additive case. In F. L. Bauer and R. Steinbrüggen, editors, *Foundations of Secure Computation*. IOS Press, Amsterdam, 2000. Preprint 1998, Inst. de Math. de Luminy, Marseilles.

[Girard, 2001] J.-Y. Girard. Locus solum. *Mathematical Structures in Computer Science*, 11:299–506, 2001.

[Givant and Andréka, 2002] S. Givant and H. Andréka. Groups and algebras of binary relations. *Bulletin of Symbolic Logic*, 8:38–64, 2002.

[Gödel, 1958] K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958. Engl. transl: *On a hitherto unexploited extension of the finitary standpoint*, in *Journal of Philosophical Logic* 9 (1980) pp. 133–142. Two other Engl. transls. with notes: in [Gödel, 1990, pp. 217–251, 271–280].

[Gödel, 1990] K. Gödel. *Collected Works, Vol. 2: Publications 1938–1974*. Oxford Univ. Press, New York and Oxford, 1990. (Eds. S. Feferman et al.).

[Gödel, 1995] K. Gödel. *Collected Works, Vol. 3: Unpublished Essays and Lectures*. Oxford Univ. Press, New York and Oxford, 1995. (Eds. S. Feferman et al.).

[Gonthier *et al.*, 1992] G. Gonthier, M. Abadi, and J.-J. Levy. The geometry of optimal lambda reduction. In *Nineteenth Annual A.C.M. Symposium on the Principles of Programming Languages (POPL)*, pages 15–26, Albuquerque, New Mexico, 1992. Association for Computing Machinery, New York.

[Goodman, 1970] N. D. Goodman. A theory of constructions equivalent to arithmetic. In A. Kino, J. Myhill, and R. Vesley, editors, *Intuitionism and Proof Theory*, pages 101–120. North-Holland Co., Amsterdam, 1970.

[Gordon *et al.*, 1979] M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF. A Mechanical Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1979.

[Gordon, 2000] M. Gordon. From LCF to HOL: a short history. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. M.I.T. Press, Cambridge, Mass., U.S.A., 2000.

[Griffin, 1990] T. Griffin. The formulae-as-types notion of control. In *Seventeenth Annual A.C.M. Symposium on the Principles of Programming Languages (POPL)*, pages 47–58. Association for Computing Machinery, New York, 1990.

[Gunter and Jung, 1990] C. Gunter and A. Jung. Coherence and consistency in domain. *Journal of Pure and Applied Algebra*, 63:49–66, 1990.

[Gunter and Mitchell, 1994] C. Gunter and J. C. Mitchell. *Theoretical Aspects of Object-Oriented Programming: Types, Semantics and Language Design*. M.I.T. Press, Cambridge, Mass., U.S.A., 1994.

[Gunter, 1987] C. Gunter. Universal profinite domains. *Information and Computation*, 72:1–30, 1987.

[Halmos, 1962] P. Halmos. *Algebraic Logic*. Chelsea Publishing Co., New York, 1962.

[Hanatani, 1966] Y. Hanatani. Calculabilité des fonctionnels recursives primitives de type fini sur les nombres naturels. *Annals of the Japan Association for the Philosophy of Science*, 3:19–30, 1966. (Revised version: *Calculability of the primitive recursive functionals of finite type over the natural numbers*, in *Proof Theory Symposion, Kiel 1974*, ed. by J. Diller and G. Müller, Lecture Notes in Mathematics 500, Springer-Verlag, Berlin, 1975, pp. 152–163).

[Hancock and Martin-Löf, 1975] P. Hancock and P. Martin-Löf. Syntax and semantics of the language of primitive recursive functions. Technical Report 3, Univ. Stockholm, 1975.

[Hankin, 1994] C. Hankin. *Lambda Calculi*. Clarendon Press, Oxford, England, 1994.

[Harper *et al.*, 1993] R. Harper, F. Honsell, and G. D. Plotkin. A framework for defining logics. *Journal of the ACM*, 40:143–184, 1993.

[Heijenoort, 1967] J. van Heijenoort. *From Frege to Gödel: A Source Book in Mathematical Logic 1879–1931*. Harvard University Press, Cambridge, Mass., U.S.A., 1967.

[Henkin, 1950] L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15:81–91, 1950.

[Heyting, 1956] A. Heyting. *Intuitionism, an Introduction*. North-Holland Co., Amsterdam, 1956. (3rd edn. 1973).

[Heyting, 1958] A. Heyting. Intuitionism in mathematics. In R. Klibansky, editor, *Philosophy in the Mid-century*, pages 101–115. La Nuova Italia, Firenze, 1958.

[Hilbert and Ackermann, 1928] D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik*. Springer-Verlag, Berlin, 1928. (English transl. of 2nd (1938) edn., with some corrections: *Principles of Mathematical Logic*, Chelsea Publ. Co., New York 1950).

[Hinata, 1967] S. Hinata. Calculability of primitive recursive functionals of finite type. *Science Reports of the Tokyo Kyoiku Daigaku, Section A*, 9(226):42–59, 1967.

[Hindley and Longo, 1980] J. R. Hindley and G. Longo. Lambda-calculus models and extensionality. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 26:289–310, 1980.

[Hindley and Seldin, 1980] J. R. Hindley and J. P. Seldin, editors. *To H. B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, London, 1980.

[Hindley and Seldin, 1986] J. R. Hindley and J. P. Seldin. *Introduction to Combinators and λ-calculus*. Cambridge Univ. Press, England, 1986.

[Hindley *et al.*, 1972] J. R. Hindley, B. Lercher, and J. P. Seldin. *Introduction to Combinatory Logic*. Cambridge Univ. Press, England, 1972.

[Hindley, 1969a] J. R. Hindley. An abstract form of the Church-Rosser theorem, I. *Journal of Symbolic Logic*, 34:545–560, 1969. From author's PhD thesis, Univ. Newcastle upon Tyne, England, 1964.

[Hindley, 1969b] J. R. Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

[Hindley, 1978a] J. R. Hindley. Reductions of residuals are finite. *Transactions of the American Mathematical Society*, 240:345–361, 1978.

[Hindley, 1978b] J. R. Hindley. Standard and normal reductions. *Transactions of the American Mathematical Society*, 241:253–271, 1978.

[Hindley, 1982] J. R. Hindley. The simple semantics for Coppo-Dezani-Sallé types. In M. Dezani and U. Montanari, editors, *International Symposium on Programming, Proceedings 1982*, volume 137, pages 212–226. Springer-Verlag, Berlin, 1982.

[Hindley, 1997] J. R. Hindley. *Basic Simple Type Theory*. Cambridge Univ. Press, England, 1997.

[Hodges, 1983] A. Hodges. *Alan Turing: The Enigma*. Burnett Books, London, 1983. Also published 1988 by Vintage, London.

[Holmes, 1991] M. R. Holmes. Systems of combinatory logic related to Quine's 'New Foundations'. *Annals of Pure and Applied Logic*, 53:103–133, 1991.

[Holmes, 1995] M. R. Holmes. Untyped λ-calculus with relative typing. In M. Dezani and G. Plotkin, editors, *Typed Lambda Calculi and Applications, Second International Conference, TLCA '95*, number 902 in Lecture Notes in Computer Science, pages 235–248. Springer-Verlag, 1995.

[Howard, 1970] W. A. Howard. Assignment of ordinals to terms for primitive recursive functionals of finite type. In A. Kino, J. Myhill, and R. Vesley, editors, *Intuitionism and Proof Theory*, pages 443–458. North-Holland Co., Amsterdam, 1970. Proc. conf. at Buffalo, N.Y., 1968.

[Howard, 1980] W. A. Howard. The formulæ-as-types notion of construction. In Hindley and Seldin [1980], pages 479–490. Manuscript circulated 1969.

[Huet, 1980] G. Huet. Confluent reductions. *Journal of the Association for Computing Machinery*, 27:797–821, 1980.

[Huet, 1990a] G. Huet. Cartesian closed categories and lambda-calculus. In Huet [1990b], pages 7–23.

[Huet, 1990b] G. Huet, editor. *Logical Foundations of Functional Programming*. Univ. Texas at Austin Year of Programming. Addison-Wesley, Reading, Mass., U.S.A., 1990.

[Huet, 1993] G. Huet. An analysis of Böhm's theorem. *Theoretical Computer Science*, 121 (Böhm Volume):154–167, 1993.

[Husserl, 1922] E. Husserl. *Logische Untersuchungen*. Max Niemeyer, Halle, Germany, 3rd edition, 1922. (1st edn. was 1900–1901).

[Hyland and Ong, 2000] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. *Information and Computation*, 163:285–408, 2000.

[Hyland, 1975] J. M. E. Hyland. A survey of some useful partial order relations on terms of the lambda calculus. In Böhm [1975], pages 83–95.

[Hyland, 1976] J. M. E. Hyland. A syntactic characterization of the equality in some models for the lambda calculus. *Journal of the London Mathematical Society, Series 2*, 12:361–370, 1976.

[Hyland, 1982] J. M. E. Hyland. The effective topos. In A. S. Troelstra and D. van Dalen, editors, *The L.E.J. Brouwer Centenary Symposium*, pages 165–216. North-Holland Co., Amsterdam, 1982.

[Hyland, 1988] J. M. E. Hyland. A small complete category. *Annals of Pure and Applied Logic*, 40:135–165, 1988.

[Hyland, 1991] J. M. E. Hyland. First steps in synthetic domain theory. In A. Carboni, editor, *Conference on Category Theory*, number 1488 in Lecture Notes in Mathematics, pages 131–156. Springer-Verlag, 1991.

[Jacobs, 1989] B. Jacobs. The inconsistency of higher order extensions of Martin-Löf's type theory. *Journal of Philosophical Logic*, 18(4):399–422, 1989.

[Johnstone *et al.*, 1978] P. Johnstone, R. Paré, R. Rosebrugh, S. Schumacher, R. Wood, and G. Wraith. *Indexed Categories and their Applications*. Number 661 in Lecture Notes in Mathematics. Springer-Verlag, 1978.

[Johnstone, 1982] P. Johnstone. *Stone Spaces*. Number 3 in Cambridge Studies in Advanced Mathematics. Cambridge Univ. Press, 1982.

[Joyal, 1977] A. Joyal. Remarques sur la théorie des jeux a deux personnes. *Gazette des Sciences Mathématiques du Quebec*, 1(4), 1977.

[Jung (editor), 1996] A. Jung (editor). Domains and denotational semantics: history, accomplishments and open problems. *Bulletin of the European Association for Theoretical Computer Science*, 59:227–256, 1996.

[Kahn and Plotkin, 1978] G. Kahn and G. D. Plotkin. Domaines concrèts. Rapport 336, IRIA Laboria, 1978. English transl. in *Theoretical Computer Science* 121 (Böhm Volume 1993), pp. 187–277.

[Kahn, 1988] G. Kahn. Natural semantics. In K. Fuchi and M. Nivat, editors, *Programming of Future Generation Computers*, pages 237–258. Elsevier, Amsterdam, 1988.

[Kalman, 1983] J. Kalman. Condensed detachment as a rule of inference. *Studia Logica*, 42:443–451, 1983.

[Kamareddine and Nederpelt, 1993] F. Kamareddine and R. P. Nederpelt. On stepwise explicit substitutions. *International Journal of Foundations of Computer Science*, 4:197–240, 1993.

[Kamareddine and Rios, 1997] F. Kamareddine and A. Rios. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, 7:395–420, 1997.

[Kamareddine and Rios, 2000] F. Kamareddine and A. Rios. Relating the lambda-sigma and lambda-s styles of explicit substitutions. *Journal of Logic and Computation*, 10:349–380, 2000.

[Kamareddine *et al.*, 2002] F. Kamareddine, T. Laan, and R. P. Nederpelt. Types in logic and mathematics before 1940. *Bulletin of Symbolic Logic*, 8:185–245, 2002.

[Kanda, 1979] A. Kanda. Fully effective solutions of recursive domain equations. In J. Beçvar, editor, *Mathematical Foundations of Computer Science*, volume 74 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1979.

[Kathail, 1990] V. Kathail. *Optimal interpreters for lambda-calculus based functional languages*. PhD thesis, Massachusetts Institute of Technology, 1990.

[Kearns, 1973] J. T. Kearns. The completeness of combinatory logic with discriminators. *Notre Dame Journal of Formal Logic*, 14:323–330, 1973.

[Kelly and Mac Lane, 1972] G. M. Kelly and S. Mac Lane. Coherence in closed categories. *Journal of Pure and Applied Algebra*, 1:97–140, 1972.

[Kleene and Rosser, 1935] S. C. Kleene and J. B. Rosser. The inconsistency of certain formal logics. *Annals of Mathematics, Series 2*, 36:630–636, 1935.

[Kleene, 1936] S. C. Kleene. λ-definability and recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.

[Kleene, 1945] S. C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109–124, 1945.

[Kleene, 1952] S. C. Kleene. *Introduction to Metamathematics*. Van Nostrand, New York, 1952.

[Kleene, 1978] S. C. Kleene. Recursive functionals and quantifiers of finite types revisited I. In J. E. Fenstad, R. O. Gandy, and G. E. Sacks, editors, *Generalized Recursion Theory II*, pages 185–222. North-Holland, Amsterdam, 1978.

[Kleene, 1980] S. C. Kleene. Recursive functionals and quantifiers of finite types revisited II. In J. Barwise, H. J. Keisler, and K. Kunen, editors, *The Kleene Symposium*, pages 1–29. North-Holland, Amsterdam, 1980.

[Kleene, 1981] S. C. Kleene. Origins of recursive function theory. *Annals of the History of Computing*, 3:52–67, 1981. Preliminary version in: *20th IEEE Symposium on the Foundations of Computer Science*, publ. by IEEE Computer Society, Los Alamitos, California, U.S.A., 1979, pp. 371–382.

[Kleene, 1982] S. C. Kleene. Recursive functionals and quantifiers of finite types revisited III. In G. Metakides, editor, *Patras Logic Symposium*, pages 1–40. North-Holland Co., Amsterdam, 1982.

[Kleene, 1985] S. C. Kleene. Unimonotone functions of finite types (Recursive functionals and quantifiers of finite types revisited IV). In A. Nerode and R. A. Shore, editors, *Recursion Theory*, pages 119–138. American Mathematical Society, Providence, Rhode Island, 1985.

[Klement, 2003] K. C. Klement. Russell's 1903–1905 anticipation of the lambda calculus. *History and Philosophy of Logic*, 24:15–37, 2003.

[Kline, 1951] G. L. Kline. Review of: Yanovskaya, S. A.: Osnovaniya matematiki i matematicheskaya logika (Foundations of mathematics and mathematical logic). *Journal of Symbolic Logic*, 16:46–48, 1951. Article reviewed: pp. 9–50 of *Matematika v SSSR za Tridkat let 1917–1947* (Mathematics in the USSR in the 30 Years 1917–1947), in Russian, publ. OGIZ, Moscow, 1948.

[Klop *et al.*, 1993] J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121(1–2):279–308, 1993.

[Klop, 1980] J. W. Klop. *Combinatory Reduction Systems.* PhD thesis, Univ. Utrecht, 1980. Publ. by Mathematisch Centrum, 413 Kruislaan, Amsterdam.

[Klop, 1992] J. W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 2: Background, Computational Structures*, pages 1–116. Oxford Univ. Press, England, 1992.

[Komori, 1990] Y. Komori. λ-calculus with simultaneous substitution as a primitive symbol. In *Proceedings of the 13th Symposium on Semigroups*, pages 20–26. Kyoto Sangyo Univ., 1990. (Symposium in 1989).

[Koymans, 1982] C. P. J. Koymans. Models of the lambda calculus. *Information and Control*, 52:306–332, 1982. (Journal now called *Information and Computation*).

[Kreisel, 1958] G. Kreisel. Constructive mathematics, 1958. Mimeographed notes of a course given at Stanford Univ., 1958–59.

[Kreisel, 1959] G. Kreisel. Interpretation of analysis by means of constructive functionals of finite type. In A. Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North-Holland Co., Amsterdam, 1959.

[Kreisel, 1962] G. Kreisel. Foundations of intuitionistic logic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Logic, Methodology, and the Philosophy of Science*, pages 198–210, Stanford, 1962.

[Kreisel, 1965] G. Kreisel. Mathematical logic. In T. L. Saaty, editor, *Lectures on Modern Mathematics*, volume 3, pages 95–195. Wiley, New York, 1965.

[Krivine, 1990] J.-L. Krivine. *Lambda-calcul, Types et Modèles*. Masson, Paris, 1990. English transl: *Lambda-Calculus, Types and Models*, Ellis-Horwood (U.S.A.) and Prentice-Hall (U.K.), 1993.

[Kuzichev, 1980] A. S. Kuzichev. Sequential systems of λ-conversion and of combinatory logic. In Hindley and Seldin [1980], pages 141–155.

[Kuzichev, 1983] A. S. Kuzichev. Set theory in type-free combinatorially complete systems. *Vestnik, Moscow State University, Series Math./Mechan.*, pages 36–42, 1983. In Russian.

[Kuzichev, 1999] A. S. Kuzichev. A version of formalization of Cantor's set theory. *Doklady Mathematics*, 60:424–426, 1999.

[Laan, 1997] T. D. L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. PhD thesis, Tech. Univ. Eindhoven, Netherlands, 1997.

[Lafont and Streicher, 1991] Y. Lafont and T. Streicher. Games semantics for linear logic. In *Proceedings Sixth Annual IEEE Symposium on Logic In Computer Science*, pages 43–50. IEEE Computer Society, Los Alamitos, California, U.S.A., 1991.

[Lamarche, 1992] F. Lamarche. Sequentiality, games and linear logic, 1992. Unpublished paper, from a lecture to the CLiCS Symposium (Aarhus Univ., March 23–27, 1992).

[Lambek and Findlay, 1955] J. Lambek and G. D. Findlay. Calculus of bimodules, 1955. Unpublished manuscript.

[Lambek, 1958] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958.

[Lambek, 1961] J. Lambek. On the calculus of syntactic types. In R. Jakobson, editor, *Structure of Language and its Mathematical Aspects*, number 12 in Proceedings of Symposia in Applied Mathematics, pages 56–68. American Math. Soc. Providence, R.I., U.S.A., 1961.

[Lambek, 1968] J. Lambek. Deductive systems and categories, I: syntactic calculus and residuated categories. *Mathematical Systems Theory*, 2(4):287–318, 1968.

[Lambek, 1969] J. Lambek. Deductive systems and categories, II: standard constructions and closed categories. In P. Hilton, editor, *Category Theory, Homology Theory and their Applications II*, volume 86 of *Lecture Notes in Mathematics*, pages 76–122. Springer-Verlag, Berlin, 1969.

[Lambek, 1972] J. Lambek. Deductive systems and categories, III: cartesian closed categories, intuitionist propositional calculus, and combinatory logic. In W. F. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lecture Notes in Mathematics*, pages 57–82. Springer-Verlag, Berlin, 1972.

[Lambek, 1974] J. Lambek. Functional completeness of cartesian categories. *Annals of Mathematical Logic*, 6:259–292, 1974.

[Lambek, 1980a] J. Lambek. From λ-calculus to cartesian closed categories. In Hindley and Seldin [1980], pages 375–402.

[Lambek, 1980b] J. Lambek. From types to sets. *Advances in Mathematics*, 36:113–164, 1980.

[Lambek, 1988] J. Lambek. Categorial grammars and natural language structures. In R. Oehrle, E. Bach, and D. Wheeler, editors, *Categorial and Categorical Grammars*. D. Reidel Co., Dordrecht, Netherlands, 1988.

[Lambek, 1995] J. Lambek. Bilinear logic in algebra and linguistics. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 43–59. Cambridge Univ. Press, England, 1995. (Proc. 1993 Workshop on Linear Logic, Ithaca, N.Y., U.S.A.).

[Lamping, 1990] J. Lamping. An algorithm for optimal lambda calculus reduction. In *Seventeenth Annual A.C.M. Symposium on the Principles of Programming Languages (POPL)*, pages 16–30. Association for Computing Machinery, New York, 1990.

[Landin, 1964] P. J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6:308–320, 1964. Actually appeared 1963.

[Landin, 1965] P. J. Landin. A correspondence between ALGOL 60 and Church's lambda notation. *Communications of the ACM*, 8:89–101, 158–165, 1965.

[Landin, 1966a] P. J. Landin. A lambda-calculus approach. In L. Fox, editor, *Advances in Programming and Non-numerical Computation*, pages 97–141. Pergamon Press, England, 1966.

[Landin, 1966b] P. J. Landin. The next 700 programming languages. *Communications of the ACM*, 9(3):157–166, 1966.

[Läuchli, 1965] H. Läuchli. Intuitionistic propositional calculus and definably nonempty terms. *Journal of Symbolic Logic*, 30:263, 1965. Abstract only.

[Läuchli, 1970] H. Läuchli. An abstract notion of realizability for which intuitionistic predicate calculus is complete. In A. Kino, J. Myhill, and R. Vesley, editors, *Intuitionism and Proof Theory*, pages 227–234. North-Holland Co., Amsterdam, 1970. Proc. 1968 conference at Buffalo, N.Y.

[Lawvere, 1963] F. W. Lawvere. *Functorial semantics of algebraic theories*. PhD thesis, Columbia Univ., 1963.

[Lawvere, 1964] F. W. Lawvere. An elementary theory of the category of sets. *Proceedings of the National Academy of Sciences of the U.S.A.*, 52(6):1506–1511, 1964. Extended abstract of a typescript with the same title (1964, Univ. Chicago).

[Lawvere, 1966] F. W. Lawvere. The category of categories as a foundation for Mathematics. In *Proceedings of the La Jolla conference on Categorical Algebra*, pages 1–20. Springer-Verlag, 1966.

[Lawvere, 1969a] F. W. Lawvere. Adjointness in Foundations. *Dialectica*, 23:281–296, 1969.

[Lawvere, 1969b] F. W. Lawvere. Diagonal arguments and cartesian closed categories. In P. Hilton, editor, *Category Theory, Homology Theory and their Applications II*, volume 92 of *Lecture Notes in Mathematics*, pages 134–145. Springer-Verlag, Berlin, 1969.

[Lawvere, 1970a] F. W. Lawvere. Equality in hyperdoctrines and comprehension schema as an adjoint functor. In A. Heller, editor, *Applications of Categorical Algebra*, volume 17 of *Proceedings of Symposia on Pure Mathematics*, pages 1–14. American Math. Soc., Providence, R.I., U.S.A., 1970. Proc. of a symposium in New York 1968.

[Lawvere, 1970b] F. W. Lawvere. Quantifiers and sheaves. In *Actes du Congrès International des Mathèmatiques, Nice 1970*, pages 329–334, 1970. tome I.

[Lawvere, 1972] F. W. Lawvere. Category theory over a base topos, 1972. Unpubl. lecture notes.

[Lawvere, 1976] F. W. Lawvere. Variable quantities and variable structures in topoi. In *Algebra, Topology and Category Theory. A collection of Papers in Honor of Samuel Eilenberg*, pages 101–131. Academic Press, London and New York, 1976.

[Lawvere, 1979] F. W. Lawvere. Categorical dynamics. In *Proceedings of the Aarhus Open House on Topos Theoretic Methods in Geometry, May 1978*, Aarhus, Denmark, 1979.

[Lawvere, 1996] F. W. Lawvere. Adjoints in and among bicategories. In A. Ursini and P. Aglianò, editors, *Logic and Algebra. Proceedings of the 1994 Siena Conference in memory of Roberto Magari*, volume 180 of *Lecture Notes in Pure and Applied Algebra*, pages 181–189. Marcel Dekker, Inc., New York, Basel and Hong Kong, 1996.

[Leivant, 1983] D. Leivant. Polymorphic type inference. In *Tenth Annual A.C.M. Symposium on the Principles of Programming Languages (POPL)*, pages 88–98. Association for Computing Machinery, New York, 1983. Cf. also pp. 155–166.

[Leivant, 1990] D. Leivant. Contracting proofs to programs. In P. Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Studies in Data Processing*, pages 279–327. Academic Press, London, 1990.

[Lemmon *et al.*, 1957] E. J. Lemmon, C. A. Meredith, D. Meredith, A. N. Prior, and I. Thomas. Calculi of pure strict implication. Informally distributed, 1957. Publ. 1969 in *Philosophical logic*, ed. by Davis and Hockney and Wilson, D. Reidel Co., Dordrecht, Netherlands, pp. 215–250.

[Lercher, 1966] B. Lercher. Review of: K. Menger: the algebra of functions: past, present, future. *Journal of Symbolic Logic*, 31:272, 1966.

[Leśniewski, 1929] S. Leśniewski. Grundzüge eines neuen Systems der Grundlagen der Mathematik. *Fundamenta Mathematicæ*, 14:1–81, 1929.

[Lévy, 1976] J.-J. Lévy. An algebraic interpretation of $\lambda\beta$K-calculus, and an application of a labelled λ-calculus. *Theoretical Computer Science*, 2:97–114, 1976. Preliminary version 1975 in [Böhm, 1975] pp. 147–165.

[Lévy, 1978] J.-J. Lévy. *Réductions correctes et optimales dans le λ-calcul*. Thèse de Doctorat d'Etat, Univ. Paris VII, Paris, 1978.

[Loader, 2001] R. Loader. Finitary PCF is not decidable. *Theoretical Computer Science*, 266:341–364, 2001.

[Longley, 2001] J. R. Longley. Notions of computability at higher types, I. In *Logic Colloquium 2000*, volume 19 of *Lecture Notes in Logic*, pages 32–142. A. K. Peters, 2001.

[Longo and Moggi, 1992] G. Longo and E. Moggi. Constructive natural deduction and its modest interpretation. *Mathematical Structures in Computer Science*, 1:215–254, 1992.

[Longo *et al.*, 1993] G. Longo, K. Milsted, and S. Soloviev. The genericity theorem and parametricity in the polymorphic λ-calculus. *Theoretical Computer Science*, 121:323–349, 1993.

[Longo, 1987] G. Longo. From numbered sets to type theories. *Rendiconti del Seminario Matematico dell'Università e del Politecnico di Torino*, Fascicolo speciale: "Logic and Computer Science: New Trends and Applications":41–73, 1987. Ed. by G. Lolli.

[Longo, 1995] G. Longo. Parametric and type-dependent polymorphism. *Fundamenta Informaticae*, 22:69–92, 1995.

[Lorenz, 2001] K. Lorenz. Basic objectives of dialogue logic in historical perspective. *Synthese*, 127:255–263, 2001.

[Lorenzen, 1955] P. Lorenzen. *Einführung in die Operative Logik und Mathematik*. Springer-Verlag, Berlin, Göttingen, Heidelberg, 1955.

[Lorenzen, 1961] P. Lorenzen. Ein dialogisches Konstructivitätskriterium. In *Infinitistic Methods*, pages 193–200. Pergamon Press and PWN, Oxford and Warsaw, 1961.

[Löwenheim, 1940] L. Löwenheim. Einkleidung der Mathematik in Schröderschen Relativkalkul. *Journal of Symbolic Logic*, 5:1–15, 1940.

[Łukasiewicz and Tarski, 1930] J. Łukasiewicz and A. Tarski. Untersuchungen über den Aussagenkalkül. *Comptes Rendus des Séances de la Societé des Sciences et des Lettres de Varsovie*, 23:30–50, 1930. Engl. transl: *Investigations into the sentential calculus*, in *Logic, Semantics, Metamathematics*, by A. Tarski, Clarendon Press, Oxford, 1956, pp. 38–59.

[Łukasiewicz, 1939] J. Łukasiewicz. Der äquivalenzenkalkul. *Collectanea Logica*, 1:145–169, 1939. Journal vol. never appeared. Engl. transl: *The equivalential calculus*, in *Jan Łukasiewicz Selected Works*, ed. by L. Borkowski, North-Holland Co., Amsterdam, 1970, pp. 250–277.

[MacQueen *et al.*, 1984] D. MacQueen, G. D. Plotkin, and R. Sethi. An ideal model for recursive polymorphic types. In *Eleventh Annual A.C.M. Symposium on the Principles of Programming Languages (POPL)*, pages 165–174. Association for Computing Machinery, New York, 1984.

[Mann, 1975] C. R. Mann. The connection between equivalence of proofs and cartesian closed categories. *Proceedings of the London Mathematical Society*, 31(3):289–310, 1975.

[Manzano, 1997] M. Manzano. Alonzo Church: his life, his work and some of his miracles. *History and Philosophy of Logic*, 18:211–232, 1997.

[Martin-Löf, 1971a] P. Martin-Löf. On the strength of intuitionistic reasoning, 1971. Unpublished contribution to Symposium on Perspectives in the Philosophy of Mathematics at 4th International Congress for Logic, Methodology and Philosophy of Science, Bucharest.

[Martin-Löf, 1971b] P. Martin-Löf. A Theory of Types, 1971. Informally circulated.

[Martin-Löf, 1972a] P. Martin-Löf. Infinite terms and a system of natural deduction. *Compositio Mathematica*, 24(1):93–103, 1972.

[Martin-Löf, 1972b] P. Martin-Löf. An Intuitionistic Theory of Types. Technical report, Dept. of Mathematics, Univ. Stockholm, 1972. Published in *Twenty-Five Years of Constructive Type Theory*, edited by Giovanni Sambin and Jan Smith, Clarendon Press, Oxford, 1998, pp. 127–172.

[Martin-Löf, 1975] P. Martin-Löf. An intuitionistic theory of types: Predicative part. In H. Rose and J. Shepherdson, editors, *Logic Colloquium '73*, pages 73–118. North-Holland Co., Amsterdam, 1975.

[Martin-Löf, 1982] P. Martin-Löf. Constructive mathematics and computer programming. In L. J. Cohen, J. Łos, H. Pfeiffer, and K-P. Podewski, editors, *Logic, Methodology and Philosophy of Science, VI*, pages 153–175. North-Holland Co., Amsterdam, 1982.

[Martin-Löf, 1983] P. Martin-Löf. The domain interpretation of type theory. In K. Karlsson and K. Petersson, editors, *Workshop on the Semantics of Programming Languages, Abstracts and Notes*. Programming Methodology Group, Chalmers Univ. Technology and Univ. Göteborg, August 1983.

[Martin-Löf, 1984] P. Martin-Löf. *Intuitionistic Type Theory*. Studies in Proof Theory. Bibliopolis, Napoli, Italy, 1984. Notes by Giovanni Sambin of lectures given in Padova, June 1980.

[Martin-Löf, 1985] P. Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. In C. Bernardi and P. Pagli, editors, *Atti degli Incontri di Logica Matematica*, pages 203–281. Univ. di Siena, 1985. Reprinted with bibliographical additions in *Nordic Journal of Philosophical Logic*, 1:11–60, 1996.

[Martin-Löf, 1986] P. Martin-Löf. Unifying Scott's theory of domains for denotational semantics and intuitionistic type theory. In V. Michele Abrusci and Ettore Casari, editors, *Atti del congresso Logica e Filosofia della Scienza, Oggi, San Gimignano, 7–11 Dicembre 1983*, volume I: Logica, pages 79–82. CLUEB, Bologna, 1986.

[Martin-Löf, 1987] P. Martin-Löf. Truth of a proposition, evidence of a judgement, validity of a proof. *Synthese*, 73:407–420, 1987.

[McCarthy, 1960] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine. *Communications of the ACM*, 3:184–195, 1960.

[McCarthy, 1963a] J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–70. North-Holland Co., Amsterdam, 1963. Earlier version publ. in Proc. of 1961 Western Joint Computer Conference.

[McCarthy, 1963b] J. McCarthy. Towards a mathematical science of computation. In C. M. Popplewell, editor, *Information Processing 62: Proceedings of the IFIP Congress 1962*, pages 21–28. North-Holland Co., Amsterdam, 1963.

[McCarthy, 1981] J. McCarthy. History of LISP. In R. Wexelblat, editor, *History of Programming Languages*, pages 173–197. Academic Press and Association for Computing Machinery (A.C.M.), New York, 1981. Also published in *A.C.M. SIGPLAN Notices* 13 (1978), 217–223.

[McCracken, 1979] N. J. McCracken. *An investigation of a programming language with a polymorphic type structure*. PhD thesis, Syracuse Univ., N.Y., U.S.A., 1979.

[McCracken, 1982] N. J. McCracken. A finitary retract model for the polymorphic lambda-calculus. Technical report, Syracuse Univ., N.Y., U.S.A., 1982. Unpublished.

[Medvedev, 1962] J. T. Medvedev. Finite problems. *Soviet Mathematics Doklady*, 3:227–230, 1962.

[Megill and Bunder, 1996] N. Megill and M. W. Bunder. Weaker D-complete logics. *Journal of the Interest Group on Propositional Logics*, 4:215–225, 1996.

[Menger, 1944] K. Menger. Tri-operational algebra. *Reports of a Mathematical Colloquium*, 5:3–10, 1944. Series publ. by Univ. Notre Dame, Indiana, U.S.A.

[Menger, 1964] K. Menger. On substitutive algebra and its syntax. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 10:81–104, 1964.

[Meredith, 1977] D. Meredith. In memoriam Carew Arthur Meredith. *Notre Dame Journal of Formal Logic*, 18:513–516, 1977.

[Meredith, 1980] D. Meredith. Problems in the analogy between λ-calculus and positive logic. *Bulletin of the European Association for Theoretical Computer Science*, 11:136–137, 1980.

[Meyer and Bunder, 1988] R. K. Meyer and M. W. Bunder. Condensed detachment and combinators. Technical Report TR-ARP-8/88, Research School of the Social Sciences, Australian National Univ., Canberra, Australia, 1988.

[Meyer *et al.*, 1991] R. K. Meyer, M. W. Bunder, and L. Powers. Implementing the "fool's model" of combinatory logic. *Journal of Automated Reasoning*, 7:597–630, 1991.

[Meyer, 1982] A. R. Meyer. What is a model of the lambda calculus? *Information and Control*, 52:87–122, 1982. Preprint circulated 1980. Journal title now *Information and Computation*.

[Mezghiche, 1984] M. Mezghiche. Une nouvelle Cβ-réduction dans la logique combinatoire. *Theoretical Computer Science*, 31:151–164, 1984.

[Milner, 1972] R. Milner. Logic for computable functions; description of a machine implementation. Technical Report STAN-CS-72-288, Stanford Univ., 1972.

[Milner, 1975] R. Milner. Processes: a mathematical model of computing agents. In H. Rose and J. Shepherdson, editors, *Logic Colloquium '73*, pages 157–174. North-Holland Co., Amsterdam, 1975.

[Milner, 1977] R. Milner. Fully abstract models of typed λ-calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[Milner, 1978] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[Mints and Tammet, 1991] G. Mints and T. Tammet. Condensed detachment is complete for relevance logic: a computer-aided proof. *Journal of Automated Reasoning*, 7:587–596, 1991.

[Mirimanoff, 1917] D. Mirimanoff. Les antinomies de Russell et de Burali-Forti et le problème fondamental de la théorie des ensembles. *L'Enseignement Mathematique*, 19:37–52, 1917.

[Mitschke, 1973] G. Mitschke. Ein algebraischer Beweis für das Church-Rosser Theorem. *Archiv für Mathematische Logik*, 15:146–157, 1973. From author's 1970 PhD thesis, Univ. Bonn, *Eine Algebraische Behandlung von λK-Kalkül und Kombinatorischer Logik*.

[Mitschke, 1979] G. Mitschke. The standardization theorem for the λ-calculus. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 25:29–31, 1979.

[Moldestad, 1977] J Moldestad. *Computations in Higher Types*. Springer-Verlag, Berlin, 1977. Series *Lecture Notes in Mathematics* No. 574.

[Montague, 1974] R. Montague. *Formal Philosophy. Selected Papers*. Yale Univ. Press, New Haven U.S.A. and London, 1974.

[Morris, 1968] J. H. Morris. *Lambda-calculus Models of Programming Languages*. PhD thesis, Massachusetts Inst. Technology, Cambridge, Mass., U.S.A., 1968.

[Moses, 1970] J. Moses. The Function of FUNCTION in LISP, 1970. Massachusetts Institute of Technology, Project MAC report AI–199. Available on Web at ftp://publications.ai.mit.edu/ ai-publications/0-499/AIM-199.ps.

[Murthy, 1991] C. Murthy. An evaluation semantics for classical proofs. In *Proceedings Sixth Annual IEEE Symposium on Logic In Computer Science*, pages 96–107. IEEE Computer Society, Los Alamitos, California, U.S.A., 1991.

[Myhill and Shepherdson, 1955] J. Myhill and J. C. Shepherdson. Effective operations on partial recursive functions. *Zeitschrift für Mathematische Logik*, 1:310–317, 1955.

[Nakajima, 1975] R. Nakajima. Infinite normal forms for the λ-calculus. In Böhm [1975], pages 62–82.

[Naur and others, 1963] P. Naur et al. Revised report on the algorithmic language ALGOL 60. *Communications of the ACM*, 6:1–17, 1963.

[Nederpelt and Geuvers, 1994] R. P. Nederpelt and J. H. Geuvers. Twenty-Five Years of Automath Research. In Nederpelt et al. [1994], pages 3–54.

[Nederpelt *et al.*, 1994] R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer, editors. *Selected Papers on Automath*. Elsevier, Amsterdam, 1994.

[Nederpelt, 1973] R. Nederpelt. *Strong normalization in a typed lambda calculus with lambda structured types*. PhD thesis, Technical Univ. Eindhoven, Netherlands, 1973. Publ. 1994 in [Nederpelt *et al.*, 1994], pp. 389–468.

[Nerode, 1957] A. Nerode. General topology and partial recursive functionals. Summaries of talks at the Cornell Summer Institute of Symbolic Logic, 1957.

[Nerode, 1959] A. Nerode. Some Stone spaces and recursion theory. *Duke Mathematical Journal*, 26:397–405, 1959.

[Neumann, 1925] J. von Neumann. Eine Axiomatizierung der Mengenlehre. *Journal für die Reine und Angewandte Mathematik*, 154:219–240, 1925. (Correction in *Ibid.* 155, p. 128.) English transl: *An Axiomatization of Set Theory*, in [Heijenoort, 1967], pp. 393–413.

[Neumann, 1928] J. von Neumann. Die Axiomatizierung der Mengenlehre. *Mathematische Zeitschrift*, 27:669–752, 1928. Based on author's doctoral thesis in Hungarian, Univ. Budapest Sept. 1925 (author's footnote p.669). Reprinted in *The Collected Works of J. von Neumann, Vol. 1*, ed. by A. H. Taub, Pergamon Press, Oxford, England 1961, pp.339–422.

[Newman, 1942] M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[Newman, 1943] M. H. A. Newman. Stratified systems of logic. *Proceedings of the Cambridge Philosophical Society*, 39:69–83, 1943. Reviewed by A. Church in J. Symbolic Logic 9 (1944), pp. 50–52.

[Nickau, 1994] H. Nickau. Hereditarily sequential functionals. In A. Nerode and Y. V. Matiyasevich, editors, *Proceedings of the Symposium on Logical Foundations of Computer Science*, volume 813 of *Lecture Notes in Computer Science*, pages 253–264. Springer-Verlag, 1994.

[Nivat, 1973] M. Nivat. Langages algébriques sur le magma libre et sémantique des schémas de programme. In M. Nivat, editor, *Automata, Languages and Programming*. North-Holland Co., Amsterdam, 1973. Proc. of 1972 conference.

[Nordström *et al.*, 1990] B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's Type Theory*. Oxford Univ. Press, England, 1990.

[Ong, 1995] C.-H. L. Ong. Correspondence between operational and denotational semantics. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 269–356. Oxford Univ. Press, England, 1995.

[Oosten, 2002] J. van Oosten. Realizability: a historical essay. *Mathematical Structures in Computer Science*, 12:239–263, 2002.

[Paiva, 1989a] V. de Paiva. The Dialectica categories. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 47–62. American Math. Society, Providence, R.I., U.S.A., 1989.

[Paiva, 1989b] V. de Paiva. A Dialectica-like model of linear logic. In *Proceedings of the Conference on Category Theory and Computer Science*, volume 389 of *Lecture Notes in Computer Science*, pages 341–356. Springer-Verlag, Berlin, 1989.

[Park, 1976] D. Park. The Y combinator in Scott's lambda calculus models. Theory of Computation Report 13, Dept. of Computer Science, Univ. Warwick, 1976. First version presented 1970 at Symposium on Theory of Programming, Univ. Warwick.

[Peano, 1889] G. Peano. *Arithmetices Principia, Nova Methodo Exposita*. Fratelli Bocca, Torino, Italy, 1889. Also in [Peano, 1958], Vol. 2, Item 16, pp. 20–55. Engl. transls: (1) The Principles of Arithmetic, in *Selected Works of Giuseppe Peano*, ed. H. Kennedy, publ. Allen and Unwin, London, 1973, pp. 101–134; (2) The Principles of Arithmetic, in [Peano, 1958], pp. 83–97.

[Peano, 1895] G. Peano. *Formulaire de Mathématiques*. Fratelli Bocca, Torino, Italy, 1895. Publ. in 5 vols. 1895–1908. Later vols. overlapped and revised earlier vols. Vol. 3 publ. Carré et Naud, Paris, 1901. Vol. 5 titled *Formulario Mathematico* in Peano's language Latino sine flexione.

[Peano, 1958] G. Peano. *Opere Scelte*. Edizioni Cremonese, Milano, 1958. Three volumes.

[Peirce, 1883] C. S. Peirce. The logic of relatives. In C. S. Peirce, editor, *Johns Hopkins Studies in Logic*, pages 187–203. Little, Brown & Co., Boston, U.S.A., 1883. Reprinted in *Collected Papers of Charles Sanders Peirce, Vol. 3, Exact Logic*, ed. by C. Hartshorne & P. Weiss, Harvard Univ. Press, U.S.A., pp. 195–209.

[Peyton Jones, 1987] S. Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice-Hall, England, 1987.

[Phoa, 1990] W. Phoa. Effective domains and intrinsic structure. In *Proceedings Fifth Annual IEEE Symposium on Logic In Computer Science*, pages 366–377. IEEE Computer Society, Los Alamitos, California, U.S.A., 1990.

[Pierce, 2002] B. C. Pierce. *Types and Programming Languages*. MIT Press, Cambridge, Mass., 2002.

[Pigozzi and Salibra, 1998] D. Pigozzi and A. Salibra. Lambda abstraction algebras: coordinatizing models of lambda calculus. *Fundamenta Informaticae*, 33:149–200, 1998.

[Piperno, 1989] A. Piperno. Abstraction problems in combinatory logic; a compositive approach. *Theoretical Computer Science*, 66:27–43, 1989.

[Platek, 1966] R. A. Platek. *Foundations of Recursion Theory*. PhD thesis, Stanford Univ., 1966.

[Plotkin and Abadi, 1993] G. D. Plotkin and M. Abadi. A logic for parametric polymorphism. In M. Bezem and J. Groote, editors, *Typed Lambda Calculi and Applications, TLCA '93*, volume 664 of *Lecture Notes in Computer Science*, pages 361–375. Springer-Verlag, Berlin, 1993.

[Plotkin, 1973] G. D. Plotkin. Lambda-definability and logical relations. Technical Report SAI-RM-4, School of Artificial Intelligence, Univ. Edinburgh, U.K., 1973.

[Plotkin, 1974] G. D. Plotkin. The λ-calculus is ω-incomplete. *Journal of Symbolic Logic*, 39:313–317, 1974.

[Plotkin, 1975] G. D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[Plotkin, 1977] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–257, 1977.

[Plotkin, 1978a] G. D. Plotkin. The category of complete partial orders: a tool for making meanings. Lectures, summer school, Dipartimento di Informatica, Università di Pisa, Italy, 1978.

[Plotkin, 1978b] G. D. Plotkin. T^ω as a universal domain. *Journal of Computer and System Sciences*, 17:209–236, 1978.

[Plotkin, 1980] G. D. Plotkin. Lambda-definability in the full type hierarchy. In Hindley and Seldin [1980], pages 363–373.

[Plotkin, 1981] G. D. Plotkin. Post-graduate lecture notes in advanced domain theory. Dept. Computer Science, Univ. Edinburgh, 1981.

[Plotkin, 1993] G. D. Plotkin. Set-theoretical and other elementary models of the λ-calculus. *Theoretical Computer Science*, 121 (Böhm Volume):351–409, 1993. (Part 1 issued 1972 as *A set-theoretical definition of application*, Report MIP-R-95 of School of Artificial Intelligence, Univ. Edinburgh.).

[Plotkin, 1994] G. D. Plotkin. A semantics for static type-inference. *Information and Computation*, 109:256–299, 1994.

[Pottinger, 1978] G. Pottinger. Proofs of the normalization and Church-Rosser theorems for the typed λ-calculus. *Notre Dame Journal of Formal Logic*, 19:445–451, 1978.

[Pottinger, 1980] G. Pottinger. A type assignment for the strongly normalizable λ-terms. In Hindley and Seldin [1980], pages 561–579.

[Prawitz, 1965] D. Prawitz. *Natural Deduction*. Almqvist and Wiksell, Stockholm, 1965.

[Prawitz, 1967] D. Prawitz. Completeness and Haupsatz for second order logic. *Theoria*, 33:246–258, 1967.

[Prawitz, 1968] D. Prawitz. Hauptsatz for higher order logic. *Journal of Symbolic Logic*, 33:452–457, 1968.

[Prawitz, 1971] D. Prawitz. Ideas and results in proof theory. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 235–307. North-Holland Co., Amsterdam, 1971.

[Quine, 1936a] W. V. Quine. A reinterpretation of Schönfinkel's logical operators. *Bulletin of the American Mathematical Society*, 42:87–89, 1936.

[Quine, 1936b] W. V. Quine. Towards a calculus of concepts. *Journal of Symbolic Logic*, 1:2–25, 1936.

[Quine, 1937] W. V. Quine. New foundations for mathematical logic. *American Mathematical Monthly*, 44:70–80, 1937.

[Quine, 1963] W. V. Quine. *Set Theory and its Logic*. Harvard Univ. Press, Cambridge, Mass., U.S.A., 1963. (Revised edn. 1969).

[Quine, 1972] W. V. Quine. Algebraic logic and predicate functors. In R. Rudner and I. Scheffler, editors, *Logic and Art: Essays in Honor of Nelson Goodman*, pages 214–238. Bobbs Merrill Co., Indianapolis and New York, 1972. (Also publ. 1971 by Bobbs Merrill Co. as a booklet *Algebraic Logic and Predicate Functors*).

[Ramsey, 1926] F. P. Ramsey. The foundations of mathematics. *Proceeding of the London Mathematical Society, Series 2*, 25:338–384, 1926. Reprinted in: *F. P. Ramsey: The Foundations of Mathematics and other Logical Essays*, ed. by R. B. Braithwaite, publ. Routledge and Kegan Paul, New York and London, 1931, pp. 1–61.

[Révész, 1978] G. Révész. λ-calculus without substitution. Technical report, Computer and Automation Institute, Hungarian Academy of Sciences, 1978.

[Révész, 1985] G. Révész. Axioms for the theory of lambda-conversion. *SIAM Journal of Computing*, 14:373–382, 1985.

[Reynolds and Plotkin, 1990] J. C. Reynolds and G. D. Plotkin. On functors expressible in the polymorphic typed lambda calculus. In Huet [1990b], pages 127–152.

[Reynolds, 1972] J. C. Reynolds. Notes on a lattice-theoretic approach to the theory of computation. Report, Syracuse Univ., Syracuse, New York, October 1972. Revised March 1979.

[Reynolds, 1974] J. C. Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Programming Symposium, Proceedings, Colloque sur la Programmation, Paris*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425, Berlin, 1974. Springer-Verlag.

[Reynolds, 1983] J. C. Reynolds. Types, abstraction and parametric polymorphism. In R. Mason, editor, *Information Processing 83*, pages 513–523. North-Holland Co., Amsterdam, 1983.

[Reynolds, 1989] J. C. Reynolds. Syntactic control of interference, part 2. In G. Ausiello, M. Dezani, and S. Ronchi della Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium*, volume 372 of *Lecture Notes in Computer Science*, pages 704–722. Springer-Verlag, Berlin, 1989.

[Reynolds, 1993] J. C. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computation*, 6:233–247, 1993.

[Rimscha, 1980] M. von Rimscha. Mengentheoretische Modelle des λK-Kalküls. *Archiv für Mathematische Logik*, 20:65–74, 1980.

[Robinson, 1965] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41, 1965.

[Rogers, 1967] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw Hill, New York, 1967.

[Ronchi della Rocca and Venneri, 1984] S. Ronchi della Rocca and B. Venneri. Principal type schemes for an extended type theory. *Theoretical Computer Science*, 28:151–171, 1984.

[Ronchi della Rocca, 1988] S. Ronchi della Rocca. Principal type schemes and unification for intersection type discipline. *Theoretical Computer Science*, 59:181–209, 1988.

[Rosen, 1973] B. K. Rosen. Tree manipulating systems and Church-Rosser theorems. *Journal of the Association for Computing Machinery*, 20:160–187, 1973.

[Rosenbloom, 1950] P. Rosenbloom. *The Elements of Mathematical Logic*. Dover Inc., New York, 1950.

[Rosolini, 1986] G. Rosolini. *Continuity and Effectiveness in Topoi*. PhD thesis, Carnegie Mellon Univ., 1986. Tech. Report CMU-CS-86-123, Dept. of Computer Science, Carnegie Mellon Univ., Pittsburgh, Pennsylvania, U.S.A.

[Rosser, 1935] J. B. Rosser. A mathematical logic without variables, Part 1. *Annals of Mathematics, Series 2*, 36:127–150, 1935. Also Part 2: *Duke Mathematical Journal* 1 (1935), pp. 328–355.

[Rosser, 1942] J. B. Rosser. New sets of postulates for combinatory logics. *Journal of Symbolic Logic*, 7:18–27, 1942.

[Rosser, 1984] J. B. Rosser. Highlights of the history of the lambda calculus. *Annals of the History of Computing*, 6:337–349, 1984.

[Russell and Whitehead, 1913] B. Russell and A. N. Whitehead. *Principia Mathematica*. Cambridge Univ. Press, Cambridge, England, 1913. In 3 parts, appeared 1910–1913. New edn. 1925–1927.

[Russell, 1903] B. Russell. *The Principles of Mathematics*. Cambridge Univ. Press, England, 1903. New edn. publ. 1992 by Routledge and Kegan Paul, England.

[Russell, 1919] B. Russell. *Introduction to Mathematical Philosophy*. Allen and Unwin, London, 1919.

[Sallé, 1978] P. Sallé. Une extension de la théorie des types en λ-calcul. In G. Ausiello and C. Böhm, editors, *Automata, Languages and Programming, Fifth Colloquium*, volume 62 of *Lecture Notes in Computer Science*, pages 398–410. Springer-Verlag, Berlin, 1978.

[Sambin, 1987] G. Sambin. Intuitionistic formal spaces: a first communication. In D. Skordev, editor, *Mathematical Logic and its Applications*, pages 187–204. Plenum Press, New York, 1987.

[Sanchis, 1967] L. E. Sanchis. Functionals defined by recursion. *Notre Dame Journal of Formal Logic*, 8:161–174, 1967. Informally circulated 1965.

[Sazonov, 1976a] V. Yu. Sazonov. Degrees of parallelism in computations. In *Mathematical Foundations of Computer Science*, volume 45 of *Lecture Notes in Computer Science*, pages 517–523. Springer-Verlag, Berlin, 1976.

[Sazonov, 1976b] V. Yu. Sazonov. Expressibility of functionals in D. Scott's LCF language. *Algebra i Logika*, 15(3):308–330, 1976.

[Sazonov, 1976c] V. Yu. Sazonov. Functionals computable in series and in parallel. *Sibirskii Matematicheskii Zhurnal*, 17:648–672, 1976.

[Schönfinkel, 1924] M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924. English transl: *On the building blocks of mathematical logic*, in [Heijenoort, 1967], pp. 355–366.

[Schröder, 1905] E. Schröder. *Vorlesungen über die Algebra der Logik (Exakte Logik)*. Teubner, Leipzig, Germany, 1905. (In fact publ. in parts. I: 1890, II/1: 1891, III: 1895, II/2: 1905).

[Schroer, 1965] D. E. Schroer. *The Church-Rosser Theorem*. PhD thesis, Cornell Univ., 1965. Informally circulated 1963. 673 pp. Obtainable from University Microfilms Inc., Ann Arbor, Michigan, U.S.A., Publication No. 66-41.

[Schütte, 1960] K. Schütte. Syntactical and semantical properties of simple type theory. *Journal of Symbolic Logic*, 25:305–326, 1960.

[Scott, 1963] D. S. Scott. A System of Functional Abstraction, 1963. Lecture notes, Stanford Univ., informally distributed.

[Scott, 1968] D. S. Scott. An Abstract Theory of Constructions, 1968. Lecture notes, Amsterdam Univ., informally distributed.

[Scott, 1969a] D. S. Scott. A construction of a model for the λ-calculus. Informally distributed, 1969. Notes for a November 1969 seminar, Oxford Univ.

[Scott, 1969b] D. S. Scott. Lattice-theoretic models for the λ-calculus, 1969. Incomplete typescript, 50 pp., Merton College, Oxford Univ.

[Scott, 1969c] D. S. Scott. Models for the λ-calculus. Informally distributed, 1969. Notes, December 1969, Oxford Univ.

[Scott, 1969d] D. S. Scott. A theory of computable functions of higher type. Informally distributed, 1969. Notes for a November 1969 seminar, Oxford Univ.

[Scott, 1969e] D. S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. Informally distributed, 1969. Dated October 1969. Publ. with additions by author in *Theoretical Computer Science* 121 (1–2) (1993), pp. 411–420.

[Scott, 1970a] D. S. Scott. Constructive validity. In M. Laudet, D. Lacombe, L. Nolin, and M. Schützenberger, editors, *Symposium on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*, pages 237–275. Springer-Verlag, Berlin, 1970. Proc. conference in Versailles 1968.

[Scott, 1970b] D. S. Scott. Lattice theory, data types and semantics. In R. Rustin, editor, *Formal Semantics of Programming Languages*, pages 65–106. Prentice-Hall, Englewood Cliffs, N.J., U.S.A., 1970.

[Scott, 1970c] D. S. Scott. Outline of a mathematical theory of computation. In *Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems*, pages 169–176. Dept. of Electrical Engineering, Princeton Univ., 1970. Also publ. 1970 as Tech. Monograph PRG-2, Programming Research Group, Oxford Univ., England.

[Scott, 1972] D. S. Scott. Continuous lattices. In F. W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lecture Notes in Mathematics*, pages 97–136, Berlin, 1972. Springer-Verlag. Preliminary version publ. 1970 as Tech. Monograph PRG-7, Programming Research Group, Oxford Univ., England.

[Scott, 1973] D. S. Scott. Models for various type-free calculi. In P. Suppes and others, editor, *Logic, Methodology and Philosophy of Science IV*, pages 157–187. North-Holland Co., Amsterdam, 1973. Proc. conference in 1971.

[Scott, 1975] D. S. Scott. Some philosophical issues concerning theories of combinators. In Böhm [1975], pages 346–366.

[Scott, 1976] D. S. Scott. Data types as lattices. *SIAM Journal on Computing*, 5:522–587, 1976.

[Scott, 1977] D. S. Scott. Logic and programming languages. *Communications of the ACM*, 20:634–641, 1977.

[Scott, 1980a] D. S. Scott. Lambda calculus: some models, some philosophy. In J. Barwise et al., editors, *The Kleene Symposium*, pages 223–265. North-Holland Co., Amsterdam, 1980.

[Scott, 1980b] D. S. Scott. Relating theories of the λ-calculus. In Hindley and Seldin [1980], pages 403–450.

[Scott, 1982a] D. S. Scott. Domains for denotational semantics. In M. Nielsen and E. Schmidt, editors, *Automata, Languages and Programming, Ninth International Colloquium*, volume 140 of *Lecture Notes in Computer Science*, pages 577–613. Springer-Verlag, Berlin, 1982.

[Scott, 1982b] D. S. Scott. Lectures on a mathematical theory of computation. In M. Broy and G. Schmidt, editors, *Theoretical Foundations of Programming Methodology*. D. Reidel Co., Dordrecht, Netherlands, 1982. Also publ. 1981 as Tech. Monograph PRG-19, Programming Research Group, Oxford Univ., England.

[Scott, 1986] D. S. Scott. Church's thesis and a unification of type universes. Lecture at conference in Utrecht on Church's Thesis fifty years later, 14–15 June 1986.

[Scott, 1996] D. S. Scott. A new category? Domains, Spaces and Equivalence relations. Unpublished manuscript, December 1996.

[Scott, 2000] D. S. Scott. Some reflections on Strachey and his work. *Higher order and Symbolic Computation*, 13:103–114, 2000.

[Seely, 1989] R. A. G. Seely. Linear logic, $*$-autonomous categories and cofree coalgebras. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 371–382. American Mathematical Society, Providence, R.I., 1989.

[Seldin, 1968] J. P. Seldin. *Studies in Illative Combinatory Logic*. PhD thesis, Inst. voor Grondslagenonderzoek, Univ. Amsterdam, 1968.

[Seldin, 1979] J. P. Seldin. Progress report on generalized functionality. *Annals of Mathematical Logic*, 17:29–59, 1979. Condensed from manuscript *Theory of Generalized functionality* informally circulated 1975. Journal now *Annals of Pure and Applied Logic*.

[Seldin, 1980a] J. P. Seldin. Curry's program. In Hindley and Seldin [1980], pages 3–33.

[Seldin, 1980b] J. P. Seldin. A short biography of Haskell B. Curry. In Hindley and Seldin [1980], pages vii–xx. Includes a bibliography.

[Seldin, 1987] J. P. Seldin. MATHESIS: The mathematical foundation of ULYSSES. Technical Report RADC-TR-87-223, Odyssey Research Associates, Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, NY 13441-5700, 1987.

[Seldin, 1997] J. P. Seldin. On the proof theory of Coquand's calculus of constructions. *Annals of Pure and Applied Logic*, 83:23–101, 1997.

[Seldin, 2002] J. P. Seldin. Curry's anticipation of the types used in programming languages. *Proceedings of the Canadian Society for History and Philosophy of Mathematics*, 15:148–163, 2002.

[Seldin, 2007] J. P. Seldin. Church and Curry: the lambda calculus and combinatory logic. In D. M. Gabbay and J. Woods, editors, *Handbook of the History of Logic*, volume 5. Elsevier Co., 2007.

[Sieg, 1997] W. Sieg. Step by recursive step: Church's analysis of effective calculability. *Bulletin of Symbolic Logic*, 3:154–180, 1997.

[Smyth and Plotkin, 1982] M. B. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *S.I.A.M. Journal of Computing*, 11:761–783, 1982.

[Smyth, 1977] M. B. Smyth. Effectively given domains. *Theoretical Computer Science*, 5:257–274, 1977.

[Smyth, 1983] M. B. Smyth. Powerdomains and predicate transformers: a topological view. In J. Diaz, editor, *Automata, Languages and Programming, Tenth International Colloquium*, volume 154 of *Lecture Notes in Computer Science*, pages 662–675. Springer-Verlag, Berlin, 1983.

[Staples, 1981] J. Staples. Efficient combinatory reduction. *Zeitschrift für Mathematische Logik*, 27:391–402, 1981.

[Statman, 1983] R. Statman. λ-definable functionals and $\beta\eta$-conversion. *Archiv für Mathematische Logik*, 23:21–26, 1983.

[Statman, 1985] R. Statman. Logical relations and the typed λ-calculus. *Information and Computation*, 65:85–97, 1985.

[Steele and Gabriel, 1996] G. L. Steele and R. P. Gabriel. The evolution of LISP. In T. Bergin and R. Gibson, editors, *History of Programming Languages – II*, pages 233–330. Addison-Wesley and Association for Computing Machinery (A.C.M.), New York, 1996.

[Steinby and Thomas, 2000] M. Steinby and W. Thomas. Trees and term rewriting in 1910: on a paper by Axel Thue. *Bulletin of the European Association for Theoretical Computer Science*, 72:256–269, 2000. October 2000.

[Stenlund, 1972] S. Stenlund. *Combinators, λ-terms and Proof Theory.* D. Reidel Co., Dordrecht, Netherlands, 1972.

[Stoy, 1977] J. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* M.I.T. Press, Cambridge, Mass., U.S.A., 1977. (Foreword by D. Scott).

[Strachey and Wadsworth, 1974] C. Strachey and C. Wadsworth. Continuations: a mathematical semantics for handling full jumps. Technical Report PRG-11, Oxford Univ. Computing Laboratory, 1974. Reprinted in *Higher-Order and Symbolic Computation* 13:135–152, 2000.

[Strachey, 1967] C. Strachey. Fundamental concepts in programming languages. International Summer School in Computer Programming, Copenhagen, 1967. Reprinted in *Higher Order and Symbolic Computation* 13:11–49, 2000.

[Sundholm, 1986] G. Sundholm. Proof theory and meaning. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Volume III*, pages 471–506. D. Reidel Co., Dordrecht, Netherlands, 1986.

[Szabo, 1974] M. E. Szabo. A categorical equivalence of proofs. *Notre Dame Journal of Formal Logic*, 15:177–191, 1974.

[Szabo, 1978] M. E. Szabo, editor. *Algebra of Proofs.* North-Holland Co., Amsterdam, 1978.

[Tait, 1965] W. W. Tait. Infinitely long terms of transfinite type. In J. Crossley and M. Dummett, editors, *Formal Systems and Recursive Functions*, pages 176–185. North-Holland Co., Amsterdam, 1965. Proc. conference 1963.

[Tait, 1966] W. W. Tait. A nonconstructive proof of Gentzen's Haupsatz for second order predicate logic. *Bulletin of the American Math. Society*, 72:980–983, 1966.

[Tait, 1967] W. W. Tait. Intensional interpretations of functionals of finite type. *Journal of Symbolic Logic*, 32:198–212, 1967.

[Tait, 2001] W. W. Tait. Gödel's unpublished papers on foundations of mathematics. *Philosophiae Mathematica*, 9:87–126, 2001.

[Tait, 2003] W. W. Tait. The completeness of Heyting first-order logic. *Journal of Symbolic Logic*, 68:751–763, 2003.

[Takahashi, 1967] Moto-o Takahashi. A proof of cut-elimination in simple type-theory. *Journal of the Mathematical Society of Japan*, 19:399–410, 1967.

[Takahashi, 1989] Masako Takahashi. Parallel reductions in λ-calculus. *Journal of Symbolic Computation*, 7:113–123, 1989. (Revised version: *Information and Computation* 118 (1995), pp. 120–127).

[Takahashi, 1991] Masako Takahashi. *Theory of Computation, Computability and Lambda-calculus*. Kindai Kagaku Sha, Tokyo, 1991. In Japanese.

[Takeuti, 1953] G. Takeuti. On a generalised logical calculus. *Japanese Journal of Mathematics*, 23:39–96, 1953.

[Tarski and Givant, 1987] A. Tarski and S. Givant. *A Formalization of Set Theory Without Variables*, volume 41 of *Colloquium publications*. American Mathematical Society, 1987.

[Tarski, 1941] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6:73–89, 1941.

[Tarski, 1953] A. Tarski. A formalization of set theory without variables. *Journal of Symbolic Logic*, 18:189, 1953. (Abstract only).

[Taylor, 1991] P. Taylor. The fixed point property in synthetic domain theory. In *Proceedings Sixth Annual IEEE Symposium on Logic In Computer Science*, pages 152–160. IEEE Computer Society, Los Alamitos, California, U.S.A., 1991.

[Terese, 2003] Terese. *Term Rewriting Systems*. Cambridge Univ. Press, England, 2003. Authors: 'Terese' research group. Editors M. Bezem, J.-W. Klop, R. de Vrijer.

[Thiel, 1995] C. Thiel. Schönfinkel, Moses. In J. Mittelstrass, editor, *Enzyklopädie Philosophie und Wissenschaftstheorie*, pages 726–727. J. B. Metzler, Stuttgart, Germany, 1995.

[Thue, 1910] A. Thue. Die Lösung eines Spezialfalles eines generellen logischen Problems. *Cristiania Videnskabsselskabs Skrifter. I. Mat.-Nat. Kl. 1910*, 8, 1910. Reprinted in *Selected Mathematical Papers of Axel Thue*, ed. by T. Nagel et al., Universitetsforlaget Oslo, 1977, pp. 273–310.

[Tiuryn, 1990] J. Tiuryn. Type inference problems: a survey. In B. Rovan, editor, *Mathematical Foundations of Computer Science 1990*, volume 452 of *Lecture Notes in Computer Science*, pages 105–120. Springer-Verlag, Berlin, 1990.

[Troelstra, 1973] A. S. Troelstra, editor. *Metamathematical Investigations of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1973. (2nd edn. 1993, publ. as Preprint no. X-93-05 by Institute for Logic, Language and Computation, Univ. Amsterdam, Plantage Muidergracht 24, 1018TV Amsterdam).

[Turing, 1936] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 42:230–265, 1936. Also see correction in *Ibid.* 43 (1937), pp. 544–546.

[Turing, 1937a] A. M. Turing. Computability and λ-definability. *Journal of Symbolic Logic*, 2:153–163, 1937.

[Turing, 1937b] A. M. Turing. The p-function in λ-K-conversion. *Journal of Symbolic Logic*, 2:164, 1937.

[Turing, 1980] A. M. Turing. Proof that every typed formula has a normal form. Manuscript undated but probably 1941 or '42. Publ. in *An early proof of normalization by A. M. Turing* by R. O. Gandy, in [Hindley and Seldin, 1980], pages 453–455, 1980.

[Turner, 1979] D. A. Turner. A new implementation technique for applicative languages. *Software – Practice and Experience*, 9:31–49, 1979.

[Uspenskii, 1955] V. A. Uspenskii. On enumeration operators. *Doklady Akademii Nauk S.S.S.R.*, 103:773–776, 1955.

[Vickers, 1989] S. Vickers. *Topology via Logic*. Cambridge Univ. Press, England, 1989.

[Vuillemin, 1974a] J. Vuillemin. Correct and optimal implementations of recursion in a simple programming language. *Journal of Computer and System Science*, 9:332–354, 1974.

[Vuillemin, 1974b] J. Vuillemin. *Syntaxe, sémantique et axiomatique d'un langage de programmation simple*. Thèse de Doctorat d'Etat, Université de Paris VII, Paris, 1974.

[Wadsworth, 1971] C. P. Wadsworth. *Semantics and pragmatics of the lambda-calculus*. PhD thesis, Univ. Oxford, England, 1971.

[Wadsworth, 1976] C. P. Wadsworth. The relation between computational and denotational properties for Scott's D_∞ models of the lambda-calculus. *SIAM Journal of Computing*, 5:488–521, 1976.

[Wadsworth, 1978] C. P. Wadsworth. Approximate reduction and lambda-calculus models. *SIAM Journal of Computing*, 7:337–356, 1978.

[Wadsworth, 1980] C. P. Wadsworth. Some unusual λ-calculus numeral systems. In Hindley and Seldin [1980], pages 215–230.

[Wadsworth, 2000] C. P. Wadsworth. Continuations revisited. *Higher-Order and Symbolic Computation*, 13:131–133, 2000.

[Wand, 1974] M. Wand. On the recursive specification of data types. In E. G. Manes, editor, *Category Theory Applied to Computation and Control*, volume 25 of *Lecture Notes in Computer Science*, pages 214–217. Springer-Verlag, Berlin, 1974.

[Wand, 1977] M. Wand. Fixed-point constructions in order-enriched categories. Technical Report 23, Computer Science Dept., Indiana Univ., Bloomington, Indiana, 1977.

[Wand, 1979] M. Wand. Fixed-point constructions in order-enriched categories. *Theoretical Computer Science*, 8:13–30, 1979.

[Wand, 1987] M. Wand. A simple algorithm and proof for type inference. *Fundamenta Informaticae*, 10:115–122, 1987.

[Welch, 1975] P. H. Welch. Continuous semantics and inside-out reductions. In Böhm [1975], pages 122–146.

[Wells, 1994] J. B. Wells. Typability and type checking in the second-order lambda-calculus are equivalent and undecidable. In *Proceedings Ninth Annual IEEE Symposium on Logic In Computer Science*, pages 176–185. IEEE Computer Society, Los Alamitos, California, U.S.A., 1994.

[Wells, 1999] J. B. Wells. Typability and type checking in System F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1–3):111–156, 1999.

[Wilken and Weiermann, 1997] G. Wilken and A. Weiermann. Sharp upper bounds for the depths of reduction trees of typed λ-calculus with recursors, 1997. Unpublished manuscript.

[Wolfengagen, 2004] V. E. Wolfengagen. *Methods and Means for Computation with Objects*. JurInfoR Ltd., Moscow, Russia, 2004. In Russian.

[Yanovskaya, 1948] S. A. Yanovskaya. Osnovaniya matematiki i matematicheskaya logika. In *Matematika v SSSR za Tridkat let 1917–1947*, pages 9–50. OGIZ, Moscow, 1948. In Russian. Reviewed by G. L. Kline in *J. Symbolic Logic* 16 (1951), pp.46–48.