VOLUME 2

HANDBOOK OF ALGEBRA

M. HAZEWINKEL

EDITOR

NORTH-HOLLAND

HANDBOOK OF ALGEBRA
VOLUME 2

ELSEVIER
AMSTERDAM • LAUSANNE • NEW YORK • OXFORD • SHANNON • SINGAPORE • TOKYO

HANDBOOK OF ALGEBRA

Volume 2

edited by
M. HAZEWINKEL
CWI, Amsterdam

2000

ELSEVIER
AMSTERDAM • LAUSANNE • NEW YORK • OXFORD • SHANNON • SINGAPORE • TOKYO

ELSEVIER SCIENCE B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

First edition 2000

Library of Congress Cataloging-in-Publication Data

A catalog record from the Library of Congress has been applied for.

ISBN: 0 444 50396 X

♾ The paper used in this publication meets the requirements of ANSI/NISO Z39.48-1992 (Permanence of Paper).
Printed in The Netherlands.

Preface

Basic philosophy

Algebra, as we know it today, consists of many different ideas, concepts and results. A reasonable estimate of the number of these different "items" would be somewhere between 50 000 and 200 000. Many of these have been named and many more could (and perhaps should) have a "name" or a convenient designation. Even the nonspecialist is likely to encounter most of these, either somewhere in the literature, disguised as a definition or a theorem, or to hear about them and feel the need for more information. If this happens, one should be able to find at least something in this Handbook; and hopefully enough to judge if it is worthwile to pursue the quest. In addition to the primary information, references to relevant articles, books or lecture notes should help the reader to complete his understanding. To make this possible, we have provided an index which is more extensive than usual and not limited to definitions, theorems and the like.

For the purpose of this Handbook, algebra has been defined, more or less arbitrarily as the union of the following areas of the Mathematics Subject Classification Scheme:

- 20 (Group theory)
- 19 (K-theory; will be treated at an intermediate level; a separate Handbook of K-theory which goes into far more detail than the section planned for this Handbook of Algebra is under consideration)
- 18 (Category theory and homological algebra; including some of the uses of categories in computer science, often classified somewhere in section 68)
- 17 (Nonassociative rings and algebras; especially Lie algebras)
- 16 (Associative rings and algebras)
- 15 (Linear and multilinear algebra, Matrix theory)
- 13 (Commutative rings and algebras; here there is a fine line to tread between commutative algebras and algebraic geometry; algebraic geometry is not a topic that will be dealt with in this Handbook; a separate Handbook on that topic is under consideration)
- 12 (Field theory and polynomials)
- 11 (As far as it used to be classified under old 12 (Algebraic number theory))
- 08 (General algebraic systems)
- 06 (Certain parts; but not topics specific to Boolean algebras as there is a separate three-volume Handbook of Boolean Algebras)

Planning

Originally, we hoped to cover the whole field in a systematic way. Volume 1 would be devoted to what we now call Section 1 (see below), Volume 2 to Section 2 and so on. A detailed and comprehensive plan was made in terms of topics which needed to be covered and authors to be invited. That turned out to be an inefficient approach. Different authors have different priorities and to wait for the last contribution to a volume, as planned originally, would have resulted in long delays. Therefore, we have opted for a dynamically evolving plan. This means that articles are published as they arrive and that the reader will find in this second volume articles from five different sections. The advantages of this scheme are two-fold: accepted articles will be published quickly and the outline of the series can be allowed to evolve as the various volumes are published. Suggestions from readers both as to topics to be covered and authors to be invited are most welcome and will be taken into serious consideration.

The list of the sections now looks as follows:

Section 1: Linear algebra. Fields. Algebraic number theory
Section 2: Category theory. Homological and homotopical algebra. Methods from logic
Section 3: Commutative and associative rings and algebras
Section 4: Other algebraic structures. Nonassociative rings and algebras. Commutative and associative rings and algebras with extra structure
Section 5: Groups and semigroups
Section 6: Representations and invariant theory
Section 7: Machine computation. Algorithms. Tables
Section 8: Applied algebra
Section 9: History of algebra

For a more detailed plan, the reader is referred to the Outline of the Series following the Preface.

The individual chapters

It is not the intention that the handbook as a whole can also be a substitute undergraduate or even graduate, textbook. The treatment of the various topics will be much too dense and professional for that. Basically, the level is graduate and up, and such material as can be found in P.M. Cohn's three-volume textbook "Algebra" (Wiley) will, as a rule, be assumed. An important function of the articles in this Handbook is to provide professional mathematicians working in a different area with sufficient information on the topic in question if and when it is needed.

Each chapter combines some of the features of both a graduate-level textbook and a research-level survey. Not all of the ingredients mentioned below will be appropriate in each case, but authors have been asked to include the following:

- Introduction (including motivation and historical remarks)
- Outline of the chapter

- Basic concepts, definitions, and results (proofs or ideas/sketches of the proofs are given when space permits)
- Comments on the relevance of the results, relations to other results, and applications
- Review of the relevant literature; possibly supplemented with the opinion of the author on recent developments and future directions
- Extensive bibliography (several hundred items will not be exceptional)

The present

Volume 1 appeared in December 1995 (copyright 1996). Volume 3 is scheduled for 2000. Thereafter, we aim at one volume per year.

The future

Of course, ideally, a comprehensive series of books like this should be interactive and have a hypertext structure to make finding material and navigation through it immediate and intuitive. It should also incorporate the various algorithms in implemented form as well as permit a certain amount of dialogue with the reader. Plans for such an interactive, hypertext, CD-Rom-based version certainly exist but the realization is still a nontrivial number of years in the future.

Bussum, June 1999 Michiel Hazewinkel

Kaum nennt man die Dinge beim richtigen Namen,
so verlieren sie ihren gefährlichen Zauber

(You have but to know an object by its proper name
for it to lose its dangerous magic)

E. Canetti

This Page Intentionally Left Blank

Outline of the Series

(as of June 1999)

Philosophy and principles of the Handbook of Algebra

Compared to the outline in Volume 1 this version differs in several aspects.

First, there is a major shift in emphasis away from completeness as far as more elementary material is concerned and towards more emphasis on recent developments and active areas.

Second, the plan is now more dynamic in that there is no longer a fixed list of topics to be covered, determined long in advance. Instead there is a more flexible nonrigid list that can change in response to new developments and availability of authors.

The new policy is therefore to work with a dynamic list of topics that should be covered, to arrange these in sections and larger groups according to the major divisions into which algebra falls, and to publish collections of contributions as they become available from the invited authors.

The coding by style below is as follows.

- **Author(s) in bold,** followed by the article title: chapters (articles) that have been received and are published or ready for publication;
- *italic*: chapters (articles) that are being written.
- plain text: topics that should be covered but for which no author has yet been definitely contracted.
- chapters that are included in volume 1 or volume 2 have a (1; xx pp.) or (2; xx pp.) after them, where xx is the number of pages.

Compared to the earlier outline the section on "Representation and invariant theory" has been thoroughly revised.

Section 1. Linear algebra. Fields. Algebraic number theory

A. Linear Algebra

G.P. Egorychev, Van der Waerden conjecture and applications (1; 22 pp.)
V.L. Girko, Random matrices (1; 52 pp.)
A. N. Malyshev, Matrix equations. Factorization of matrices (1; 38 pp.)
L. Rodman, Matrix functions (1; 38 pp.)
Linear inequalities (also involving matrices)

Orderings (partial and total) on vectors and matrices
Positive matrices
Special kinds of matrices such as Toeplitz and Hankel
Integral matrices. Matrices over other rings and fields

B. Linear (In)dependence

J.P.S. Kung, Matroids (1; 28 pp.)

C. Algebras Arising from Vector Spaces

Clifford algebras, related algebras, and applications

D. Fields, Galois Theory, and Algebraic Number Theory

(There is also an article on ordered fields in Section 4)
J.K. Deveney and J.N. Mordeson, Higher derivation Galois theory of inseparable field extensions (1; 34 pp.)
I. Fesenko, Complete discrete valuation fields. Abelian local class field theories (1; 48 pp.)
M. Jarden, Infinite Galois theory (1; 52 pp.)
R. Lidl and H. Niederreiter, Finite fields and their applications (1; 44 pp.)
W. Narkiewicz, Global class field theory (1; 30 pp.)
H. van Tilborg, Finite fields and error correcting codes (1; 28 pp.)
Skew fields and division rings. Brauer group
Topological and valued fields. Valuation theory
Zeta and L-functions of fields and related topics
Structure of Galois modules
Constructive Galois theory (realizations of groups as Galois groups)

E. Nonabelian Class Field Theory and the Langlands Program

(To be arranged in several chapters by Y. Ihara)

F. Generalizations of Fields and Related Objects

U. Hebisch and H.J. Weinert, Semi-rings and semi-fields (1; 38 pp.)
G. Pilz, Near rings and near fields (1; 36 pp.)

Section 2. Category theory. Homological and homotopical algebra. Methods from logic

A. Category Theory

S. MacLane and I. Moerdijk, Topos theory (1; 28 pp.)
R. Street, Categorical structures (1; 50 pp.)
Algebraic theories
B. Plotkin, Algebra, categories and databases (2; 68 pp.)
P.J. Scott, Some aspects of categories in computer science (2; 73 pp.)

B. Homological Algebra. Cohomology. Cohomological Methods in Algebra. Homotopical Algebra

J.F. Carlson, The cohomology of groups (1; 30 pp.)
A. Generalov, Relative homological algebra. Cohomology of categories, posets, and coalgebras (1; 28 pp.)
J.F. Jardine, Homotopy and homotopical algebra (1; 32 pp.)
B. Keller, Derived categories and their uses (1; 32 pp.)
A.Ya. Helemskii, Homology for the algebras of analysis (2; 122 pp.)
Galois cohomology
Cohomology of commutative and associative algebras
Cohomology of Lie algebras
Cohomology of group schemes

C. Algebraic K-theory

Algebraic K-theory: the classical functors K_0, K_1, K_2
Algebraic K-theory: the higher K-functors
Grothendieck groups
K_2 *and symbols*
KK-theory and EXT
Hilbert C^**-modules*
Index theory for elliptic operators over C^* *algebras*
Algebraic K-theory (including the higher K_n*)*
Simplicial algebraic K-theory
Chern character in algebraic K-theory
Noncommutative differential geometry
K-theory of noncommutative rings
Algebraic L-theory
Cyclic cohomology

D. Model Theoretic Algebra

Methods of logic in algebra (general)
Logical properties of fields and applications
Recursive algebras
Logical properties of Boolean algebras
F. Wagner, Stable groups (2; 40 pp.)

E. Rings up to Homotopy

Rings up to homotopy

Section 3. Commutative and associative rings and algebras

A. Commutative Rings and Algebras

J.P. Lafon, Ideals and modules (1; 24 pp.)

General theory. Radicals, prime ideals etc. Local rings (general). Finiteness and chain conditions
Extensions. Galois theory of rings
Modules with quadratic form
Homological algebra and commutative rings. Ext, Tor, etc. Special properties (p.i.d., factorial, Gorenstein, Cohen–Macauley, Bezout, Fatou, Japanese, excellent, Ore, Prüfer, Dedekind, ... and their interrelations)
D. Popescu, Artin approximation (2; 34 pp.)
Finite commutative rings and algebras. (See also Section 3B)
Localization. Local–global theory
Rings associated to combinatorial and partial order structures (straightening laws, Hodge algebras, shellability, ...)
Witt rings, real spectra

B. Associative Rings and Algebras

P.M. Cohn, Polynomial and power series rings. Free algebras, firs and semifirs (1; 30 pp.)
Classification of Artinian algebras and rings
V.K. Kharchenko, Simple, prime, and semi-prime rings (1; 52 pp.)
A. van den Essen, Algebraic microlocalization and modules with regular singularities over filtered rings (1; 28 pp.)
F. Van Oystaeyen, Separable algebras (2; 43 pp.)
K. Yamagata, Frobenius rings (1; 48 pp.)
V.K. Kharchenko, Fixed rings and noncommutative invariant theory (2; 38 pp.)
General theory of associative rings and algebras
Rings of quotients. Noncommutative localization. Torsion theories
von Neumann regular rings
Semi-regular and pi-regular rings
Lattices of submodules
A.A. Tuganbaev, Modules with distributive submodule lattice (2; 16 pp.)
A.A. Tuganbaev, Serial and distributive modules and rings (2; 19 pp.)
PI rings
Generalized identities
Endomorphism rings, rings of linear transformations, matrix rings
Homological classification of (noncommutative) rings
Group rings and algebras
Dimension theory
Duality. Morita-duality
Commutants of differential operators
Rings of differential operators
Graded and filtered rings and modules (also commutative)
Goldie's theorem, Noetherian rings and related rings
Sheaves in ring theory
A.A. Tuganbaev, Modules with the exchange property and exchange rings (2; 19 pp.)

Finite associative rings (see also Section 3A)

C. Co-algebras

D. Deformation Theory of Rings and Algebras (Including Lie Algebras)

Deformation theory of rings and algebras (general)
Yu. Khakimdjanov, Varieties of Lie algebras (2; 31 pp.)

Section 4. Other algebraic structures. Nonassociative rings and algebras. Commutative and associative algebras with extra structure

A. Lattices and Partially Ordered Sets

Lattices and partially ordered sets
Frames, locales, quantales

B. Boolean Algebras

C. Universal Algebra

D. Varieties of Algebras, Groups, ... (See also Section 3D)

V.A. Artamonov, Varieties of algebras (2; 29 pp.)
Varieties of groups
Quasi-varieties
Varieties of semigroups

E. Lie Algebras

Yu.A. Bahturin, A.A. Mikhalev and M. Zaicev, Infinite-dimensional Lie superalgebras (2; 34 pp.)
General structure theory
Free Lie algebras
Classification theory of semisimple Lie algebras over **R** and **C**
The exceptional Lie algebras
M. Goze and Yu. Khakimdjanov, Nilpotent and solvable Lie algebras (2; 47 pp.)
Universal envelopping algebras
Modular (ss) Lie algebras (including classification)
Infinite-dimensional Lie algebras (general)
Kac–Moody Lie algebras

F. Jordan Algebras (Finite and infinite dimensional and including their cohomology theory)

G. Other Nonassociative Algebras (Malcev, alternative, Lie admissable, ...)

Mal'tsev algebras
Alternative algebras

H. Rings and Algebras with Additional Structure

Graded and super algebras (commutative, associative; for Lie superalgebras, see Section 4E)
Topological rings
Hopf algebras
Quantum groups
Formal groups
λ-rings, γ-rings, ...
Ordered and lattice-ordered groups, rings and algebras
Rings and algebras with involution. C^*-algebras
Difference and differential algebra. Abstract (and p-adic) differential equations. Differential extensions
Ordered fields

I. Witt Vectors

Witt vectors and symmetric functions. Leibniz Hopf algebra and quasi-symmetic functions

Section 5. Groups and semigroups

A. Groups

A.V. Mikhalev and A.P. Mishina, Infinite Abelian groups: Methods and results (2; 36 pp.)
Simple groups, sporadic groups
Abstract (finite) groups. Structure theory. Special subgroups. Extensions and decompositions
Solvable groups, nilpotent groups, p-groups
Infinite soluble groups
Word problems
Burnside problem
Combinatorial group theory
Free groups (including actions on trees)
Formations
Infinite groups. Local properties
Algebraic groups. The classical groups. Chevalley groups
Chevalley groups over rings
The infinite dimensional classical groups
Other groups of matrices. Discrete subgroups
Reflection groups. Coxeter groups
Groups with BN-pair, Tits buildings, ...
Groups and (finite combinatorial) geometry
"Additive" group theory
Probabilistic techniques and results in group theory
Braid groups

B. Semigroups

Semigroup theory. Ideals, radicals, structure theory
Semigroups and automata theory and linguistics

C. Algebraic Formal Language Theory. Combinatorics of Words

D. Loops, Quasigroups, Heaps, ...

E. Combinatorial Group Theory and Topology

Section 6. Representation and invariant theory

A. Representation Theory. General

Representation theory of rings, groups, algebras (general)
Modular representation theory (general)
Representations of Lie groups and Lie algebras. General

B. Representation Theory of Finite and Discrete Groups and Algebras

Representation theory of finite groups in characteristic zero
Modular representation theory of finite groups. Blocks
Representation theory of the symmetric groups (both in characteristic zero and modular)
Representation theory of the finite Chevalley groups (both in characteristic zero and modular
Modular representation theory of Lie algebras

C. Representation Theory of 'Continuous Groups' (Linear Algebraic Groups, Lie Groups, Loop Groups, ...) and the Corresponding Algebras

Representation theory of compact topolgical groups
Representation theory of locally compact topological groups
Representation theory of $SL_2(\mathbf{R}), \ldots$
Representation theory of the classical groups. Classical invariant theory
Classical and transcendental invariant theory
Reductive groups and their representation theory
Unitary representation theory of Lie groups
Finite-dimensional representation theory of the ss Lie algebras (in characteristic zero); structure theory of semi-simple Lie algebras
Infinite dimensional representation theory of ss Lie algebras. Verma modules
Representation of Lie algebras. Analytic methods
Representations of solvable and nilpotent Lie algebras. The Kirillov orbit method
Orbit method, Dixmier map, ... for ss Lie algebras
Representation theory of the exceptional Lie groups and Lie algebras
Representation theory of 'classical' quantum groups
A.U. Klimyk, Infinite-dimensional representations of quantum algebras (2; 27 pp.)
Duality in representation theory

Representation theory of loop groups and higher dimensional analogues, gauge groups, and current algebras
Representation theory of Kac–Moody algebras
Invariants of nonlinear representations of Lie groups
Representation theory of infinite-dimensional groups like GL_∞
Metaplectic representation theory

D. Representation Theory of Algebras

Representations of rings and algebras by sections of sheafs
Representation theory of algebras (Quivers, Auslander–Reiten sequences, almost split sequences, ...)

E. Abstract and Functorial Representation Theory

Abstract representation theory
S. Bouc, Burnside rings (2; 64 pp.)
P. Webb, A guide to Mackey functors (2; 30 pp.)

F. Representation Theory and Combinatorics

G. Representations of Semigroups

Representation of discrete semigroups
Representations of Lie semigroups

Section 7. Machine computation. Algorithms. Tables

Some notes on this volume: Besides some general article(s) on machine computation in algebra, this volume should contain specific articles on the computational aspects of the various larger topics occurring in the main volume, as well as the basic corresponding tables. There should also be a general survey on the various available symbolic algebra computation packages.

The CoCoA computer algebra system

Section 8. Applied algebra

Section 9. History of algebra

Contents

List of Contributors

Artamonov, V.A., *Moscow State University, Moscow*
Bahturin, Yu., *Memorial University of Newfoundland, St. John's, NF, and Moscow State University, Moscow*
Bouc, S., *Université Paris 7-Denis Diderot, Paris*
Goze, M., *Université de Haute Alsace, Mulhouse*
Helemskii, A.Ya., *Moscow State University, Moscow*
Khakimdjanov, Yu., *Université de Haute Alsace, Mulhouse*
Kharchenko, V.K., *Universidad Nacional Autónoma de México, México, and Sobolev Institute of Mathematics, Novosibirsk*
Klimyk, A.U., *Institute for Theoretical Physics, Kiev*
Mikhalev, A.A., *The University of Hong Kong, Pokfulam Road*
Mikhalev, A.V., *Moscow State University, Moscow*
Mishina, A.P., *Moscow State University, Moscow*
Plotkin, B., *Hebrew University, Jerusalem*
Popescu, D., *University of Bucharest, Bucharest*
Scott, P.J., *University of Ottawa, Ottawa, ON*
Tuganbaev, A.A., *Moscow State University, Moscow*
Van Oystaeyen, F., *University of Antwerp, Wilrijk*
Wagner, F., *University of Oxford, Oxford*
Webb, P., *University of Minnesota, Minneapolis, MN*
Zaicev, M., *Moscow State University, Moscow*

This Page Intentionally Left Blank

Section 2A
Category Theory

This Page Intentionally Left Blank

Some Aspects of Categories in Computer Science

P.J. Scott

Department of Mathematics, University of Ottawa, Ottawa, Ontario, Canada

Contents

HANDBOOK OF ALGEBRA, VOL. 2
Edited by M. Hazewinkel

This Page Intentionally Left Blank

1. Introduction

Over the past 25 years, category theory has become an increasingly significant conceptual and practical tool in many areas of computer science. There are major conferences and journals devoted wholly or partially to applying categorical methods to computing. At the same time, the close connections of computer science to logic have seen categorical logic (developed in the 1970's) fruitfully applied in significant ways in both theory and practice.

Given the rapid and enormous development of the subject and the availability of suitable graduate texts and specialized survey articles, we shall only examine a few of the areas that appear to the author to have conceptual and mathematical interest to the readers of this Handbook. Along with the many references in the text, the reader is urged to examine the final section (Literature Notes) where we reference omitted important areas, as well as the Bibliography.

We shall begin by discussing the close connections of certain closed categories with typed lambda calculi on the one hand, and with the proof theory of various logics on the other. It cannot be overemphasized that modern computer science heavily uses formal syntax but we shall try to tread lightly. The so-called Curry–Howard isomorphism (which identifies formal proofs with lambda terms, hence with arrows in certain free categories) is the cornerstone of modern programming language semantics and simply cannot be overlooked.

NOTATION. We often elide composition symbols, writing $gf : A \to C$ for $g \circ f : A \to C$, whenever $f : A \to B$ and $g : B \to C$. To save some space, we have omitted large numbers of routine diagrams, which the reader can find in the sources referenced.

2. Categories, lambda calculi, and formulas-as-types

2.1. *Cartesian closed categories*

Cartesian closed categories (ccc's) were developed in the 1960's by F.W. Lawvere [Law66, Law69]. Both Lawvere and Lambek [L74] stressed their connections to Church's lambda calculus, as well as to intuitionistic proof theory. In the 1970's, work of Dana Scott and Gordon Plotkin established their fundamental role in the semantics of programming languages. A precise equivalence between these three notions (ccc's, typed lambda calculi, and intuitionistic proof theory) was published in Lambek and Scott [LS86]. We recall the appropriate definitions:

DEFINITION 2.1.

(i) A *Cartesian category* C is a category with distinguished finite products (equivalently, binary products and a terminal object $\mathbf{1}$). This says there are isomorphisms (natural in A, B, C)

$$Hom_C(A, \mathbf{1}) \cong \{*\}, \tag{1}$$

$$Hom_C(C, A \times B) \cong Hom_C(C, A) \times Hom_C(C, B). \tag{2}$$

(ii) A *Cartesian closed category* $\mathcal{C}$ is a Cartesian category $\mathcal{C}$ such that, for each object $A \in \mathcal{C}$, the functor $(-) \times A : \mathcal{C} \to \mathcal{C}$ has a specified right adjoint, denoted $(-)^A$. That is, there is an isomorphism (natural in B and C)

$$Hom_{\mathcal{C}}(C \times A, B) \cong Hom_{\mathcal{C}}\big(C, B^A\big). \tag{3}$$

For many purposes in computer science, it is often useful to have categories with explicitly given *strict* structure along with *strict* functors that preserve everything on the nose. We may present such ccc's equationally, in the spirit of multisorted universal algebra. The arrows and equations are summarized in Figure 1. These equations determine the isomorphisms (1), (2), and (3). In this presentation we say the structure is *strict*, meaning there is only one object representing each of the above constructs $\mathbf{1}$, $A \times B$, B^A. The exponential object B^A is often called the *function space* of A and B. In the computer science literature, the function space is often denoted $A \Rightarrow B$, while the arrow $C \xrightarrow{f^*} B^A$ is often called *currying* of f.

REMARK 2.2. Following most categorical logic and computer science literature, we do not assume ccc's have finite limits [Law69,LS86,AC98,Mit96]), in order to keep the correspondence with simply typed lambda calculi, cf. Theorem 2.20 below. Earlier books (cf. [Mac71]) do not always follow this convention.

Let us list some useful examples of Cartesian closed categories: for details see [LS86, Mit96,Mac71]

Objects	Distinguished Arrow(s)	Equations
Terminal $\mathbf{1}$	$A \xrightarrow{!_A} \mathbf{1}$	$!_A = f$, $f : A \to \mathbf{1}$
Products $A \times B$	$\pi_1^{A,B} : A \times B \to A$ $\pi_2^{A,B} : A \times B \to B$ $\dfrac{C \xrightarrow{f} A \; C \xrightarrow{g} B}{C \xrightarrow{\langle f,g\rangle} A \times B}$	$\pi_1 \circ \langle f, g\rangle = f$ $\pi_2 \circ \langle f, g\rangle = g$ $\langle \pi_1 \circ h, \pi_2 \circ h\rangle = h$, $h : C \to A \times B$
Exponentials B^A	$ev_{A,B} : B^A \times A \to B$ $\dfrac{C \times A \xrightarrow{f} B}{C \xrightarrow{f^*} B^A}$	$ev \circ \langle f^* \circ \pi_1, \pi_2\rangle = f$ $(ev \circ \langle g \circ \pi_1, \pi_2\rangle)^* = g$, $g : C \to B^A$

Fig. 1. CCC's equationally.

EXAMPLE 2.3. The category **Set** of sets and functions. Here $A \times B$ is a chosen Cartesian product and B^A is the set of functions from A to B. The map $B^A \times A \xrightarrow{ev} B$ is the usual evaluation map, while currying $C \xrightarrow{f^*} B^A$ is the map $c \mapsto (a \mapsto f(c, a))$.

An important subfamily of examples are *Henkin models* which are ccc's in which the terminal object **1** is a generator ([Mit96], Theorem 7.2.41). More concretely, for a lambda calculus signature with freely generated types (cf. Section 2.13 below), a *Henkin model* $\mathcal{A}$ is a type-indexed family of sets $\mathcal{A} = \{A_\sigma \mid \sigma \text{ a type}\}$ where $A_1 = \{*\}$, $A_{\sigma\times\tau} = A_\sigma \times A_\tau$, $A_{\sigma\Rightarrow\tau} \subseteq A_\tau^{A_\sigma}$ which forms a ccc with respect to restriction of the usual ccc structure of **Set**. In the case of atomic base sorts b, $\mathcal{A}_b$ is some fixed but arbitrary set. A *full type hierarchy* is a Henkin model with full function spaces, i.e. $A_{\sigma\Rightarrow\tau} = A_\tau^{A_\sigma}$.

EXAMPLE 2.4. More generally, the functor category $\mathbf{Set}^{\mathcal{C}^{op}}$ of presheaves on $\mathcal{C}$ is Cartesian closed. Its objects are (contravariant) functors from $\mathcal{C}$ to **Set**, and its arrows are natural transformations between them. We sketch the ccc structure: given $F, G \in \mathbf{Set}^{\mathcal{C}^{op}}$, define $F \times G$ pointwise on objects and arrows. Motivated by Yoneda's Lemma, define $G^F(A) = Nat(h^A \times F, G)$, where $h^A = Hom(A, -)$. This easily extends to a functor. Finally if $H \times F \xrightarrow{\theta} G$, define $H \xrightarrow{\theta^*} G^F$ by: $\theta^*_A(a)_C(h, c) = \theta_C(H(h)(a), c)$.

Functor categories have been used in studying problematic semantical issues in Algol-like languages [Rey81,Ol85,OHT92,Ten94], as well as recently in concurrency theory and models of π-calculus [CSW,CaWi]. Special cases of presheaves have been studied extensively [Mit96,LS86]:

- Let $\mathcal{C}$ be a poset (qua trivial category). Then $\mathbf{Set}^{\mathcal{C}^{op}}$, the category of Kripke models over $\mathcal{C}$, may be identified with sets indexed (or graded) by the poset $\mathcal{C}$. Such models are fundamental in intuitionistic logic [LS86,TrvD88] and also arise in Kripke Logical Relations, an important tool in semantics of programming languages [Mit96,OHT93, OHRi].
- Let $\mathcal{C} = \mathcal{O}(X)$, the poset of opens of the topological space X. The subcategory $Sh(X)$ of sheaves on X is Cartesian closed.
- Let $\mathcal{C}$ be a monoid M (qua category with one object). Then $\mathbf{Set}^{\mathcal{C}^{op}}$ is the category of *M-sets*, i.e. sets X equipped with a left action; equivalently, a monoid homomorphism $M \to End(X)$, where $End(X)$ is the monoid of endomaps of X. Morphisms of M-sets X and Y are equivariant maps (i.e. functions commuting with the action.) A special case of this is when M is actually a group G (qua category with one object, where all maps are isos). In that case $\mathbf{Set}^{\mathcal{C}^{op}}$ is the category of G-sets, the category of permutational representations of G. Its objects are sets X equipped with left actions $G \to Sym(X)$ and whose morphisms are equivariant maps. We shall return to these examples when we speak of Laüchli semantics and Full Completeness, Section 5.2

EXAMPLE 2.5. ω-**CPO**. Objects are posets P such that countable ascending chains $a_0 \leqslant a_1 \leqslant a_2 \leqslant \cdots$ have suprema. Morphisms are maps which preserve suprema of countable ascending chains (in particular, are order preserving). This category is a ccc, with products $P \times Q$ ordered pointwise and $Q^P = Hom(P, Q)$, ordered pointwise. In this case, the categories are ω-**CPO**-enriched – i.e. the hom-sets themselves form an ω-**CPO**, com-

patible with composition. An important subccc is ω-$\mathbf{CPO}_\perp$, in which all objects have a distinguished minimal element $\perp$ (but morphisms need not preserve it).

The category ω-**CPO** is the most basic example in a vast research area, *domain theory*, which has arisen since 1970. This area concerns the denotational semantics of programming languages and models of untyped lambda calculi (cf. Section 2.5 below). See also the survey article [AbJu94].

EXAMPLE 2.6. *Coherent spaces and stable maps.* A *Coherent Space* $\mathcal{A}$ is a family of sets satisfying: (i) $a \in \mathcal{A}$ and $b \subseteq a$ implies $b \in \mathcal{A}$, and (ii) if $B \subseteq \mathcal{A}$ and if $\forall c, c' \in B(c \cup c' \in \mathcal{A})$ then $\bigcup B \in \mathcal{A}$. In particular, $\emptyset \in \mathcal{A}$. Morphisms are *stable maps*, i.e. monotone maps preserving pullbacks and filtered colimits. That is, $f : \mathcal{A} \to \mathcal{B}$ is a stable map if

(i) $b \subseteq a \in \mathcal{A}$ implies $f(b) \subseteq f(a)$,
(ii) $f(\bigcup_{i \in I} a_i) = \bigcup_{i \in I} f(a_i)$, for I directed, and
(iii) $a \cup b \in \mathcal{A}$ implies $f(a \cap b) = f(a) \cap f(b)$.

This gives a category **Stab**. Every coherent space $\mathcal{A}$ yields a reflexive-symmetric (undirected) graph $(|\mathcal{A}|, \frown)$ where $|\mathcal{A}| = \{a \mid \{a\} \in \mathcal{A}\}$ and $a \frown b$ iff $\{a, b\} \in \mathcal{A}$. Moreover, there is a bijective correspondence between such graphs and coherent spaces. Given two coherent spaces $\mathcal{A}, \mathcal{B}$ their product $\mathcal{A} \times \mathcal{B}$ is defined via the associated graphs as follows: $(|\mathcal{A} \times \mathcal{B}|, \frown_{\mathcal{A}\times\mathcal{B}})$, with $|\mathcal{A} \times \mathcal{B}| = |\mathcal{A}| \uplus |\mathcal{B}| = (\{1\} \times |\mathcal{A}|) \cup (\{2\} \times |\mathcal{B}|)$ where $(1, a) \frown_{\mathcal{A}\times\mathcal{B}} (1, a')$ iff $a \frown_{\mathcal{A}} a'$, $(2, b) \frown_{\mathcal{A}\times\mathcal{B}} (2, b')$ iff $b \frown_{\mathcal{B}} b'$, and $(1, a) \frown_{\mathcal{A}\times\mathcal{B}} (2, b)$ for all $a \in |\mathcal{A}|$, $b \in |\mathcal{B}|$. The function space $\mathcal{B}^{\mathcal{A}} = \mathit{Stab}(\mathcal{A}, \mathcal{B})$ of stable maps can be given the structure of a coherent space, ordered by Berry's order: $f \preceq g$ iff for all $a, a' \in \mathcal{A}$, $a' \subseteq a$ implies $f(a') = f(a) \cap g(a')$. For details, see [GLT,Tr92]. This class of domains led to the discovery of linear logic (Section 4.2).

EXAMPLE 2.7. *Per models.* A *partial equivalence relation* (per) is a symmetric, transitive relation $\sim_A \subseteq A^2$. Thus $\sim_A$ is an equivalence relation on the subset $\mathit{Dom}_A = \{x \in A \mid x \sim_A x\}$. A $\mathcal{P}$-set is a pair $(A, \sim_A)$ where A is a set and $\sim_A$ is a per on A. Given two $\mathcal{P}$-sets $(A, \sim_A)$ and $(B, \sim_B)$ a *morphism of $\mathcal{P}$-sets* is a function $f : A \to B$ such that $a \sim_A a'$ implies $f(a) \sim_B f(a')$ for all $a, a' \in A$. That is, f induces a map of quotients $\mathit{Dom}_A / \sim_A \to \mathit{Dom}_B / \sim_B$ which preserves the associated partitions.

$\mathcal{P}$**Set**, the category of $\mathcal{P}$-sets and morphisms is a ccc, with structure induced from **Set**: we define $(A \times B, \sim_{A\times B})$, where $(a, b) \sim_{A\times B} (a', b')$ iff $a \sim_A a'$ and $b \sim_B b'$ and $(B^A, \sim_{B^A})$, where $f \sim_{B^A} g$ iff for all $a, a' \in A$, $a \sim_A a'$ implies $f(a) \sim_B g(a')$. We shall discuss variants of the ccc structure of $\mathcal{P}$**Set** in Section 2.7 below, with respect to reduction-free normalization.

Other classes of *Per* models are obtained by considering pers on a fixed (functionally complete) partial combinatory algebra, for example built over a model of untyped lambda calculus (cf. Section 2.5 below). The prototypical example is the following category $\mathit{Per}(\mathbf{N})$ of pers on the natural numbers. The objects are pers on **N**. Morphisms $R \xrightarrow{f} S$ are (equivalence classes of) partial recursive functions (= Turing-machine computable partial functions) $\mathbf{N} \rightharpoonup \mathbf{N}$ which induce a total map on the induced partitions, i.e. for all $m, n \in \mathbf{N}$, mRn implies $f(m), f(n)$ are defined and $f(m) S f(n)$. Here we define equivalence of maps $f, g : R \to S$ by: $f \sim g$ iff $\forall m, n$, mRn implies $f(m), g(n)$ are de-

fined and $f(m)Sg(n)$. The fact that $Per(\mathbf{N})$ is a ccc uses some elementary recursion theory [BFSS90,Mit96,AL91]. (See also Section 2.4.1.)

EXAMPLE 2.8. *Free CCC's.* Given a set of basic objects $\mathcal{X}$, we can form $\mathcal{F}_{\mathcal{X}}$, the *free ccc generated by* $\mathcal{X}$. Its objects are freely generated from $\mathcal{X}$ and $\mathbf{1}$ using $\times$ and $(-)^{(-)}$, its arrows are freely generated using identities and composition plus the structure in Figure 1, and we impose the minimal equations required to have a ccc. More generally, we may build $\mathcal{F}_{\mathcal{G}}$, the free ccc generated by a directed multigraph (or even a small category) $\mathcal{G}$, by freely generating from the vertices (resp. objects) and edges (resp. arrows) of $\mathcal{G}$ and then – in the case of categories $\mathcal{G}$ – imposing the appropriate equations. The sense that this is free is related to Definition 2.9 and discussed in Example 2.23.

Cartesian closed categories can themselves be made into a category in many ways. This depends, to some extent, on how much 2- , bi-, enriched- , etc. structure one wishes to impose. The following elementary notions have proved useful. We shall mention a comparison between strict and nonstrict ccc's with coproducts in Remark 2.28. More general notions of monoidal functors, etc. will be mentioned in Section 4.1.

DEFINITION 2.9. $\mathbf{CART}_{st}$ is the category of strictly structured Cartesian closed categories with functors that preserve the structure on the nose. $\mathbf{2\text{-}CART}_{st}$ is the 2-category whose 0-cells are Cartesian closed categories, whose 1-cells are strict Cartesian closed functors, and whose 2-cells are natural isomorphisms [Cu93].

As pointed out by Lambek [L74,LS86], given a ccc $\mathcal{A}$, we may adjoin an indeterminate arrow $\mathbf{1} \xrightarrow{x} A$ to $\mathcal{A}$ to form a *polynomial* Cartesian closed category $\mathcal{A}[x]$ over $\mathcal{A}$, with the expected universal property in $\mathbf{CART}_{st}$. The objects of $\mathcal{A}[x]$ are the same as those of $\mathcal{A}$, while the arrows are "polynomials", i.e. formal expressions built from the symbol x using the arrow-forming operations of $\mathcal{A}$. The key fact about such polynomial expressions is a normal form theorem, stated here for ccc's, although it applies more generally (see [LS86], p. 61):

PROPOSITION 2.10 (Functional completeness). *For every polynomial* $\varphi(x)$ *in an indeterminate* $\mathbf{1} \xrightarrow{x} A$ *over a ccc* $\mathcal{A}$, *there is a unique arrow* $\mathbf{1} \xrightarrow{h} C^A \in \mathcal{A}$ *such that* $ev \circ \langle h, x\rangle \underset{x}{=} \varphi(x)$, *where* $\underset{x}{=}$ *is equality in* $\mathcal{A}[x]$.

Looking ahead to lambda calculus notation in the next section, we write $h \equiv \lambda_{x:A}.\varphi(x)$, so the equation above becomes $ev \circ \langle \lambda_{x:A}.\varphi(x), x\rangle \underset{x}{=} \varphi(x)$. The universal property of polynomial algebras guarantees a notion of *substitution of constants* $\mathbf{1} \xrightarrow{a} A \in \mathcal{A}$ for indeterminates x in $\varphi(x)$. We obtain the following:

COROLLARY 2.11 (The β rule). *In the situation above, for any arrow* $\mathbf{1} \xrightarrow{a} A \in \mathcal{A}$

$$ev \circ \langle \lambda_{x:A} \cdot \varphi(x), a\rangle = \varphi(a) \tag{4}$$

holds in $\mathcal{A}$.

The β-rule is the foundation of the *lambda calculus*, fundamental in programming language theory. It says the following: we think of $\lambda_{x:A}.\varphi(x)$ as the function $x \mapsto \varphi(x)$. Equation 4 says: evaluating the function $\lambda_{x:A}.\varphi(x)$ at argument a is just substitution of the constant a for each occurrence of x in $\varphi(x)$. However this process is far more sophisticated than simple polynomial substitution in algebra. In our situation, the argument a may itself be a lambda term, which in turn may contain other lambda terms applied to various arguments, etc. After substitution, the right hand side $\varphi(a)$ of Eq. (4) may be far more complex than the left hand side, with many new possibilities for evaluations created by the substitution. Thus, if we think of *computation* as oriented rewriting from the LHS to the RHS, it is not at all obvious the process ever halts. The fact that it does is a basic theorem in the so-called Operational Semantics of typed lambda calculus. Indeed, the Strong Normalization Theorem (cf. [LS86], p. 81) says *every* sequence of ordered rewrites (from left to right) eventually halts at an irreducible term (cf. Remark 2.49 and Section 2.7 below).

REMARK 2.12. We may also form polynomial ccc's $\mathcal{A}[x_1, \ldots, x_n]$ by adjoining a finite set of indeterminates $\mathbf{1} \xrightarrow{x_i} A_i$. Using product types, one may show $\mathcal{A}[x_1, \ldots, x_n] \cong \mathcal{A}[z]$, for an indeterminate $\mathbf{1} \xrightarrow{z} A_1 \times \cdots \times A_n$.

Polynomial Cartesian or Cartesian closed categories $\mathcal{A}[x]$ may be constructed directly, showing they are the Kleisli category of an appropriate comonad on $\mathcal{A}$ (see [LS86], p. 56). Extensions of this technique to allow adjoining indeterminates to fibrations, using 2-categorical machinery are considered in [HJ95].

2.2. *Simply typed lambda calculi*

Lambda Calculus is an abstract theory of functions developed by Alonzo Church in the 1930's. Originally arising in the foundations of logic and computability theory, more recently it has become an essential tool in the mathematical foundations of programming languages [Mit96]. The calculus itself, to be described below, encompasses the process of building functions from variables and constants, using application and functional abstraction.

Actually, there are many "lambda calculi" – typed and untyped – with various elaborate structures of types, terms, and equations. Let us give the basic typed one. We shall follow an algebraic syntax as in [LS86].

DEFINITION 2.13 (*Typed λ-calculus*). Let Sorts be a set of sorts (or atomic types). The *typed λ-calculus generated by* Sorts is a formal system consisting of three classes: Types, Terms and Equations between terms. We write $a : A$ for "a is a term of type A".

Types: This is the set obtained from the set of Sorts using the following rules: Sorts are types, $\mathbf{1}$ is a type, and if A and B are types then so are $A \times B$ and B^A. We allow the possibility of other types or type-forming operations and possible identifications between types. (Set theorists may even use "classes" instead of "sets".)

Terms: To every type A we assign a denumerable set of typed variables $x_i^A : A$, $i = 0, 1, 2, \ldots$. We write $x : A$ or x^A for a typical variable x of type A. Terms are freely

generated from variables, constants, and term-forming operations. We require at least the following distinguished generators:

(1) $*:\mathbf{1}$,
(2) If $a:A$, $b:B$, $c:A\times B$, then $\langle a,b\rangle:A\times B$, $\pi_1^{A,B}(c):A$, $\pi_2^{A,B}(c):B$,
(3) If $a:A$, $f:B^A$, $\varphi:B$ then $ev_{A,B}(f,a):B$, $\lambda_{x:A}.\varphi:B^A$.

There may be additional constants and term-forming operations besides those specified.

We shall abbreviate $ev_{A,B}(f,a)$ by $f\text{`}a$, read "f *of* a", omitting types when clear. Intuitively, $ev_{A,B}$ denotes evaluation, $\langle -,-\rangle$ denotes pairing, and $\lambda_{x:A}.\varphi$ denotes the function $x\mapsto\varphi$, where φ is some term expression possibly containing x. The operator $\lambda_{x:A}$ acts like a quantifier, so the variable x in $\lambda_{x:A}\cdot\varphi$ is a bound (or dummy) variable, just like the x in $\forall_{x:A}\varphi$ or in $\int f(x)\,dx$. We inductively define the sets of free and bound variables in a term t, denoted $FV(t)$, $BV(t)$, resp. (cf. [Bar84], p. 24). We shall always identify terms up to renaming of bound variables. The expression $\varphi[a/x]$ denotes the result of substituting the term $a:A$ for each occurrence of $x:A$ in φ, if necessary renaming bound variables in φ so that no clashes occur (cf. [Bar84]). Terms without free variables are called *closed*; otherwise, *open*.

Equations between terms: A *context* Γ is a finite set of (typed) variables. An *equation in context* Γ is an expression $a\underset{\Gamma}{=}a'$, where a,a' are terms of the same type A whose free variables are contained in Γ.

The equality relation between terms (in context) of the same type is generated using (at least) the following axioms and closure under the following rules:

(i) $\underset{\Gamma}{=}$ is an equivalence relation.

(ii) $\dfrac{a\underset{\Gamma}{=}b}{a\underset{\Delta}{=}b}$, whenever $\Gamma\subseteq\Delta$.

(iii) $\underset{\Gamma}{=}$ must be a congruence relation with respect to all term-forming operations. It suffices to consider closure under the following two rules (cf. [LS86])

$$\frac{a\underset{\Gamma}{=}b}{f\text{`}a\underset{\Gamma}{=}f\text{`}b}\qquad\frac{\varphi\underset{\Gamma\cup\{x^A\}}{=}\varphi'}{\lambda_{x:A}\cdot\varphi\underset{\Gamma}{=}\lambda_{x:A}\cdot\varphi'}.$$

(iv) The following specific axioms (we omit subscripts on terms, when the types are obvious):
Products
(a) $a\underset{\Gamma}{=}*$ for all $a:\mathbf{1}$,
(b) $\pi_1(\langle a,b\rangle)\underset{\Gamma}{=}a$ for all $a:A$, $b:B$,
(c) $\pi_2(\langle a,b\rangle)\underset{\Gamma}{=}b$ for all $a:A$, $b:B$,
(d) $\langle\pi_1(c),\pi_2(c)\rangle\underset{\Gamma}{=}c$ for all $c:C$,

Lambda Calculus

β-Rule $(\lambda_{x:A} \cdot \varphi)\text{`}a \underset{\Gamma}{=} \varphi[a/x]$,

η-Rule $\lambda_{x:A} \cdot (f\text{`}x) \underset{\Gamma}{=} f$, where $f : B^A$ and x is not a free variable of f.

REMARK 2.14. There may be additional types, terms, or equations. Following standard conventions, we equate terms which only differ by change of bound variables – this is called α-conversion in the literature [Bar84]. Equations are *in context* – i.e. occur within a declared set of free variables. This allows the possibility of *empty types*, i.e. types without closed terms (of that type). This view is fundamental in recent approaches to functional languages [Mit96] and necessary for interpreting such theories in presheaf categories, for example. However, if there happen to be closed terms $a : A$ of each type, we may omit the subscript Γ on equations, because of the following derivable rule (cf. [LS86], Prop. 10.1, p. 75): for $x \notin \Gamma$ and if all free variables of a are contained in Γ,

$$\frac{\varphi(x) \underset{\Gamma \cup \{x\}}{=} \psi(x)}{\varphi[a/x] \underset{\Gamma}{=} \psi[a/x]}.$$

EXAMPLE 2.15. *Freely generated simply typed lambda calculi.* These are freely generated from specified sorts, terms, and/or equations. In the minimal case (no additional assumptions) we obtain the simply typed lambda calculus with finite products freely generated by Sorts. Typically, however, we assume that among the Sorts are distinguished *datatypes* and associated terms, possibly with specified equations. For example, basic universal algebra would be modelled by sorts A with distinguished n-ary operations given by terms $t : A^n \Rightarrow A$ and constants $c : \mathbf{1} \to A$. Any specified term equations are added to the theory as (nonlogical) axioms.

EXAMPLE 2.16. *The internal language of a ccc* $\mathcal{A}$. Here the types are the objects of $\mathcal{A}$, where $\times, (-)^{(-)}, \mathbf{1}$ have the obvious meanings. Terms with free variables $x_1 : A_1, \ldots, x_n : A_n$ are polynomials in $\mathcal{A}[x_1, \ldots, x_n]$, where $\mathbf{1} \xrightarrow{x_i} A_i$ is an indeterminate, lambda abstraction is given by functional completeness, as in Proposition 2.10, and we define $a \underset{X}{=} b$ to hold iff $a \underset{X}{=} b$ as polynomials in $\mathcal{A}[X]$, where $X = \{x_1, \ldots, x_n\}$.

REMARK 2.17.

(i) Historically, typed lambda calculi were often presented with *only* exponential types B^A (no products) and the associated machinery [Bar84,Bar92]. This permits certain simplifications in inductive arguments, athough it is categorically less "natural" (cf. also Remark 2.24).

(ii) It is a fundamental property that lambda calculus is a *higher-order* functional language: terms of type B^A can use an arbitrary term of type A as an argument, and A and B themselves may be very complex. Thus, typed lambda calculus is often referred to as a theory of *functionals of higher type*.

2.3. *Formulas-as-types: The fundamental equivalence*

Let us describe the third component of the trio: Cartesian closed categories, typed lambda calculi, and formulas-as-types. The *Formulas-as-Types* view, sometimes called the Curry–Howard isomorphism, is playing an increasingly influential role in the logical foundations of computing, especially in the foundations of functional programming languages. Its historical roots lie in the so-called Brouwer–Heyting–Kolmogorov (BHK) interpretation of intuitionistic logic from the 1920's [GLT,TrvD88]. The idea is based on modelling proofs (which are programs) by functions, i.e. lambda terms. Since proofs can be modelled by lambda terms and the latter are themselves arrows in certain free categories, it follows that functional programs can be modelled categorically.

In modern guise, the Curry–Howard analysis says the following. Proofs in a constructive logic $\mathcal{L}$ may be identified as terms of an appropriate typed lambda calculus $\lambda_{\mathcal{L}}$, where:

- types = formulas of $\mathcal{L}$,
- lambda terms = proofs (i.e. annotations of Natural Deduction proof trees),
- provable equality of lambda terms corresponds to the equivalence relation on proofs generated by Gentzen's normalization algorithm.

Often researchers impose additional equations between lambda terms, motivated from categorical considerations (e.g., to force traditional datatypes to have a strong universal mapping property).

REMARK 2.18 (*formulas = specifications*). More generally, the Curry–Howard view identifies types of a programming language with formulas of some logic, and programs of type A as proofs within the logic of formula A. Constructing proofs of formula A may then be interpreted as building programs that meet the specification A.

For example, consider the intuitionistic $\{\top, \wedge, \Rightarrow\}$-fragment of propositional calculus, as in Figure 2. This logic closely follows the presentation of ccc's in Definition 2.1 and Figure 1. We now identify (= Formulas-as-Types) the propositional symbols $\top, \wedge, \Rightarrow$ with the type constructors $\mathbf{1}, \times, \Rightarrow$, respectively. We assign lambda terms inductively. To a proof

Formulas	$A ::= \top \mid \mathit{Atoms} \mid A_1 \wedge A_2 \mid A_1 \Rightarrow A_2$
Provability	$\vdash$ is a reflexive, transitive relation such that, for arbitrary formulas A, B, C
	$A \vdash \top$, $A \wedge B \vdash A$, $A \wedge B \vdash B$
	$C \vdash A \wedge B$ iff $C \vdash A$ and $C \vdash B$
	$C \wedge A \vdash B$ iff $C \vdash A \Rightarrow B$

Fig. 2. Intuitionistic $\top, \wedge, \Rightarrow$ logic.

of $A \vdash B$ we assign λ-terms $x : A \vdash t(x) : B$, where $t(x)$ is a term of type B with at most the free variable $x : A$ (i.e. in context $\{x : A\}$) as follows:

$$x : A \vdash x : A, \quad \frac{x : A \vdash s(x) : B \; y : B \vdash t(y) : C}{x : A \vdash t[s(x)/y] : C},$$

$$x : A \vdash * : \top, \quad x : A \wedge B \vdash \pi_1(x) : A, \quad x : A \wedge B \vdash \pi_2(x) : B,$$

$$\frac{x : C \vdash a : A \; x : C \vdash b : B}{x : C \vdash \langle a, b \rangle : A \wedge B}, \quad \frac{z : C \wedge A \vdash t(z) : B}{y : C \vdash \lambda_{x:A} \cdot t[\langle y, x \rangle / z] : A \Rightarrow B},$$

$$\frac{y : C \vdash t(y) : A \Rightarrow B}{z : C \wedge A \vdash t[\pi_1(z)/y]\,\pi_2(z) : B}.$$

We can now refer to entire proof trees by the associated lambda terms. We wish to put an equivalence relation on proofs, according to the equations of typed lambda calculus. Given two proofs of an entailment $A \vdash B$, say $x : A \vdash s(x) : B$ and $x : A \vdash t(x) : B$, we say they are *equivalent* if we can derive $s =_{\{x\}} t$ in the appropriate typed lambda calculus.

DEFINITION 2.19. Let λ-**Calc** denote the category whose objects are typed lambda calculi and whose morphisms are *translations*, i.e. maps Φ which send types to types, terms to terms (including mapping the ith variable of type A to the ith variable of type $\Phi(A)$), preserve all the specified operations on types and terms on the nose, and preserve equations.

THEOREM 2.20. *There are a pair of functors* $\mathbf{C} : \lambda\text{-}\mathbf{Calc} \to \mathbf{Cart}_{st}$ *and* $\mathbf{L} : \mathbf{Cart}_{st} \to \lambda\text{-}\mathbf{Calc}$ *which set up an equivalence of categories* $\mathbf{Cart}_{st} \cong \lambda\text{-}\mathbf{Calc}$.

The functor L associates to ccc $\mathcal{A}$ its internal language, while the functor $\mathbf{C}$ associates to any lambda calculus $\mathcal{L}$, a syntactically generated ccc $\mathbf{C}(\mathcal{L})$, whose objects are types of $\mathcal{L}$ and whose arrows $A \to B$ are denoted by (equivalence classes of) lambda terms $t(x)$ representing proofs $x : A \vdash t(x) : B$ as above (see [LS86]).

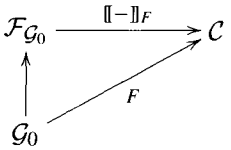
This leads to a kind of Soundness Theorem for diagrammatic reasoning which is important in categorical logic.

COROLLARY 2.21. *Verifying that a diagram commutes in a ccc* $\mathcal{C}$ *is equivalent to proving an equation in the internal language of* $\mathcal{C}$.

The above result includes allowing algebraic theories modelled in the Cartesian fragment [Mac82,Cr93], as well as extensions with categorical data types (like weak natural numbers objects, see Section 2.3.1). Theorem 2.20 also leads to concrete syntactic presentations of free ccc's [LS86,Tay98]. Let **Graph** be the category of directed multi-graphs [ST96].

COROLLARY 2.22. *The forgetful functor* $\mathcal{U} : \mathbf{Cart}_{st} \to \mathbf{Graph}$ *has a left adjoint* $\mathcal{F} : \mathbf{Graph} \to \mathbf{Cart}_{st}$. *Let* $\mathcal{F}\mathcal{G}$ *denote the image of graph* $\mathcal{G}$ *under* $\mathcal{F}$. *We call* $\mathcal{F}\mathcal{G}$ *the free ccc generated by* $\mathcal{G}$.

EXAMPLE 2.23. Given a discrete graph $\mathcal{G}_0$ considered as a a set, $\mathcal{F}_{\mathcal{G}_0}$ = the free ccc generated by the set of sorts $\mathcal{G}_0$. It has the following universal property: for any ccc $\mathcal{C}$ and for any graph morphism $F : \mathcal{G}_0 \to \mathcal{C}$, there is a unique extension to a (strict) ccc-functor $[\![-]\!]_F : \mathcal{F}_{\mathcal{G}_0} \to \mathcal{C}$.



This says: given any interpretation F of basic atomic types (= nodes of $\mathcal{G}_0$) as objects of $\mathcal{C}$, there is a unique extension to an interpretation $[\![-]\!]_F$ in $\mathcal{C}$ of the entire simply typed lambda calculus generated by $\mathcal{G}_0$ (identifying the free ccc $\mathcal{F}_{\mathcal{G}_0}$ with this lambda calculus).

REMARK 2.24. A Pitts [Pi9?] has shown how to construct free ccc's syntactically, using lambda calculi without product types. The idea is to take objects to be *sequences* of types and arrows to be sequences of terms. The terminal object is the empty sequence, while products are given by concatenation of sequences. For a full discussion, see [CDS97]. This is useful in reduction-free normalization (see Section 2.7 below).

REMARK 2.25. There are more advanced 2- and bi-categorical versions of the above results. We shall mention more structure in the case of Cartesian closed categories with coproducts, in the next section.

2.3.1. *Some datatypes.* Computing requires datatypes, for example natural numbers, lists,
arrays, etc. The categorical development of such datatypes is an old and established area. The reader is referred to any of the standard texts for discussion of the basics, e.g., [MA86, BW95,Mit96,Ten94]. General categorical treatments of abstract datatypes abound in the literature. The standard treatment is to use initial T-algebras (cf. Section 2.4.2 below) or final T-coalgebras for "definable" or "polynomial" endofunctors T. There are interesting common generalizations to lambda calculi with functorial type constructors [Ha87, Wr89], categories with datatypes determined by strong monads [Mo91,CSp91], and using enriched categorical structures [K82]. There is recent discussion of datatypes in distributive categories [Co93,W92], and the use of the categorical theory of sketches [BW95,Bor94].

We shall merely illustrate a few elementary algebraic structures commonly added to a Cartesian or Cartesian closed category (or the associated term calculi).

DEFINITION 2.26. A category $\mathcal{C}$ *has finite coproducts* (equivalently, binary coproducts and an initial object $\mathbf{0}$) if for every $A, B \in \mathcal{C}$ there is a distinguished object $A + B$, together with isomorphisms (natural in $A, B, C \in \mathcal{C}$)

$$Hom_{\mathcal{C}}(\mathbf{0}, A) \cong \{*\}, \tag{5}$$

$$Hom_{\mathcal{C}}(A + B, C) \cong Hom_{\mathcal{C}}(A, C) \times Hom_{\mathcal{C}}(B, C). \tag{6}$$

Objects	Distinguished Arrow(s)	Equations
Initial $\mathbf{0}$	$\mathbf{0} \xrightarrow{O_A} A$	$O_A = f$, $f : \mathbf{0} \to A$
Coproducts $A + B$	$in_1^{A,B} : A \to A + B$ $in_2^{A,B} : B \to A + B$ $\frac{A \xrightarrow{f} C \quad B \xrightarrow{g} C}{A+B \xrightarrow{[f,g]} C}$	$[f, g] \circ in_1 = f$ $[f, g] \circ in_2 = g$ $[h \circ in_1, h \circ in_2] = h$, $h : A + B \to C$

Fig. 3. Coproducts.

We say $\mathcal{C}$ is *bi-Cartesian closed* (= bicce) if it is a ccc with finite coproducts.[1]

Just as in the case of products (cf. Figure 1), we may present coproducts equationally, as in Figure 3, and speak of *strict* structure, etc. In programming language semantics, coproducts correspond to *variant types*, set-theoretically they correspond to disjoint union, while from the logical viewpoint coproducts correspond to disjunction. Thus a bicce corresponds to intuitionistic $\{\bot, \top, \wedge, \vee, \Rightarrow\}$-logic. We add to the logic of Figure 2 formulas $\bot$ and $A_1 \vee A_2$, together with the rules

$$\begin{array}{lll} & \bot \vdash A & \\ A \vee B \vdash C & \text{iff} & A \vdash C \text{ and } B \vdash C \end{array}$$

corresponding to Eqs. (5), (6). The associated typed lambda calculus with coproducts is rather subtle to formulate [Mit96,GLT]. The problem is with the copairing operator

$$A + B \xrightarrow{[f,g]} C$$

which in **Sets** corresponds to a *definition-by-cases* operator:

$$[f, g](x) = \begin{cases} f(x) & \text{if } x \in A, \\ g(x) & \text{if } x \in B. \end{cases}$$

The correct lambda calculus formalism for coproduct types corresponds to the logicians' natural deduction rules for strong sums. The issue is not trivial, since the word problem for free bicce's (and the associated type isomorphism problem [DiCo95]) is among the most difficult of this type of question, and – at least for the current state of the art – depends heavily on technical subtleties of syntax for its solution (see [Gh96]).

Just as for ccc's, we may introduce various 2-categories of bicce's (cf. [Cu93]). For example

[1] Not to be confused with *bicategories*, cf. [Bor94].

DEFINITION 2.27. The 2-category **2-BiCART**$_{st}$ has 0-cells strict bi-Cartesian closed categories, 1-cells functors preserving the structure on the nose, and 2-cells natural isomorphisms.

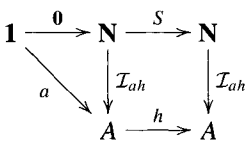
One may similarly define a non-strict version **2-BiCART**.

REMARK 2.28. Every bi-Cartesian closed category is equivalent to a strict one. Indeed, this is part of a general 2-categorical adjointness between the above 2-categories, from a theorem of Blackwell, Kelly, and Power. (See Čubrić [Cu93] for applications to lambda calculi.)

DEFINITION 2.29. In a bicce, define $\mathbf{Boole} = \mathbf{1} + \mathbf{1}$, the type of Booleans.

Boole's most salient feature is that it has two distinguished global elements (Boolean values) $\mathbf{T}, \mathbf{F} : \mathbf{1} \to \mathbf{Boole}$, corresponding to the two injections in_1, in_2, together with the universal property of coproducts. In **Set** we interpret **Boole** as a set of cardinality 2; similarly, in typed lambda calculus, it corresponds to a type with two distinguished constants $\mathbf{T}, \mathbf{F} : \mathbf{Boole}$ and an appropriate notion of definition by cases. In any bicce, we can define all of the classical n-ary propositional logic connectives as arrows $\mathbf{Boole}^n \to \mathbf{Boole}$ (see [LS86], I.8). A weaker notion of Booleans in the category ω-CPO$_\perp$ is illustrated in Figure 4.

DEFINITION 2.30. A *natural numbers object* in a ccc $\mathcal{C}$ is an object $\mathbf{N}$ with arrows $\mathbf{1} \xrightarrow{0} \mathbf{N} \xrightarrow{S} \mathbf{N}$ which is initial among diagrams of that shape. That is, for any object A and arrows $\mathbf{1} \xrightarrow{a} A \xrightarrow{h} A$, there is a unique *iterator* $\mathcal{I}_{ah} : \mathbf{N} \to A$ making the following diagram commute:



A *weak natural numbers object* is defined as above, but just assuming existence and not necessarily uniqueness of $\mathcal{I}_{ah}$.

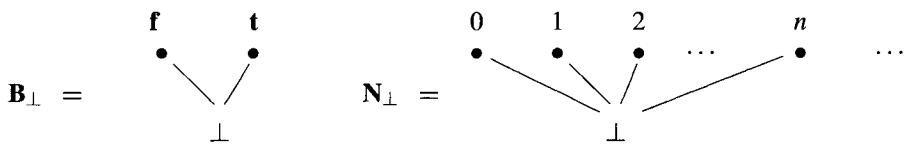


Fig. 4. Flat datatypes in ω-CPO$_\perp$.

In the category **Set**, the natural numbers $(\mathbf{N}, 0, S)$ is a natural numbers object, where $Sn = n + 1$.

In functor categories $\mathbf{Set}^{\mathcal{C}}$, a natural numbers object is given by the constant functor $K_{\mathbf{N}}$, where $K_{\mathbf{N}}(A) = \mathbf{N}$, and $K_{\mathbf{N}}(f) = id_{\mathbf{N}}$, with obvious natural transformations $\mathbf{1} \xrightarrow{0} K_{\mathbf{N}} \xrightarrow{S} K_{\mathbf{N}}$. In ω-**CPO** there are numerous *weak* natural numbers objects: for example the *flat pointed* natural numbers $\mathbf{N}_{\perp} = \mathbf{N} \uplus \{\perp\}$, ordered as follows: $a \leqslant b$ iff $a = b$ or $a = \perp$, where $S(n) = n + 1$ and $S(\perp) = \perp$, pictured in Figure 4.

Natural numbers objects – when they exist – are unique up to isomorphism; however weak ones are far from unique. Typical programming languages and typed lambda calculi in logic assume only weak natural numbers objects.

If a ccc $\mathcal{C}$ has a natural numbers object $\mathbf{N}$, we can construct parametrized versions of iteration, using products and exponentiation in $\mathcal{C}$ [LS86,FrSc]. For example, in **Set**: given functions $g : A \to B$ and $f : \mathbf{N} \times A \times B \to B$, there exists a unique *primitive recursor* $\mathcal{R}_{gf} : \mathbf{N} \times A \to B$ satisfying: (i) $\mathcal{R}_{gf}(0, a) = g(a)$ and (ii) $\mathcal{R}_{gf}(Sn, a) = f(n, a, \mathcal{R}_{gf}(n, a))$. These equations are easily represented in any ccc with $\mathbf{N}$, or in the associated typed lambda calculus (e.g., the number $n \in \mathbf{N}$ being identified with $S^n 0$). In the case $\mathcal{C}$ has only a weak natural numbers object, we may prove the existence but not necessarily the uniqueness of $\mathcal{R}_{gf}$.

An important datatype in Computer Science is the type of finite lists of elements of some type A. This is defined analogously to (weak) natural numbers objects:

DEFINITION 2.31. Given an object A in a ccc $\mathcal{C}$, we define the object $\ell ist(A)$ of *finite lists on* A with the following distinguished structure: arrows $\mathsf{nil} : \mathbf{1} \to \ell ist(A)$, $\mathsf{cons} : A \times \ell ist(A) \to \ell ist(A)$ satisfying the following (weak) universal property: for any object B and arrows $b : \mathbf{1} \to B$ and $h : A \times B \to B$, there exists an "iterator" $\mathcal{I}_{bh} : \ell ist(A) \to B$ satisfying (in the internal language):

$$\mathcal{I}_{bh}\,\mathsf{nil} = b, \qquad \mathcal{I}_{bh}\,\mathsf{cons}\langle a, w\rangle = h\langle a, \mathcal{I}_{bh} w\rangle.$$

Here nil corresponds to the empty list, and cons takes an element of A and a list and concatenates the element onto the head of the list.

Analogously to (weak) natural numbers objects $\mathbf{N}$, we can use product types and exponentiation to extend iteration on $\ell ist(A)$ to *primitive recursion with parameters* (cf. [GLT], p. 92).

What n-ary numerical **Set** functions are represented by arrows $\mathbf{N}^n \to \mathbf{N}$ in a ccc? The answer, of course, depends on the ccc. In general, the best we could expect is the following (cf. [LS86], Part III, Section 2):

PROPOSITION 2.32. *Let $\mathcal{F}_{\mathbf{N}}$ be the free ccc with weak natural numbers object. The class of numerical total functions representable therein is properly contained between the primitive recursive and the Turing-machine computable functions.*

In general, such fast-growing functions as the Ackermann function are representable in any ccc with weak natural numbers object (see [LS86]). Analogous results hold for symmetric monoidal and monoidal closed categories, [PR89].

The question of *strong* versus *weak* datatypes is of some interest. For example, although we can define addition $+ : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ by primitive recursion on a weak natural numbers type, *commutativity* of addition follows from having a strong natural numbers object; a weak parametrized primitive recursor would only allow us to derive $x + n = n + x$ for each closed numeral n but we cannot then extend this to *variables* (cf. Gödel's incompleteness theorem, cf. [LS86], p. 263). Notice that, on the face of it, the definition of a natural numbers object appears not to be equational: informally, uniqueness of the arrow $\mathcal{I}_{ah}$ requires an *implication*: *for all* $f : \mathbf{N} \to A$ (*if* $f0 = a$ *and* $fS = hf$) *then* $f = \mathcal{I}_{ah}$.

Here we remark on a curious observation of Lambek [L88]. Let us recall from universal algebra that a Mal'cev operator on an algebra A is a function $m_A : A^3 \to A$ satisfying $m_A xxz = z$ and $m_A xzz = x$. For example, if A were a group, $m_A = xy^{-1}z$ is such an operator. Similarly, the definition of a Mal'cev operator on an object A makes sense in any ccc (e.g., as an arrow $A^3 \xrightarrow{m_A} A$ satisfying some diagrams) or, equivalently, in any typed lambda calculus (e.g., as a closed term $m_A : A^3 \Rightarrow A$ satisfying some equations).

THEOREM 2.33 (Lambek). *Let $\mathcal{C}$ be a ccc with weak natural numbers $(\mathbf{N}, 0, S)$ in which each object A has a Mal'cev operator m_A. Then the fact that $(\mathbf{N}, 0, S)$ is a natural numbers object is equationally definable using the family $\{m_A \mid A \in \mathcal{C}\}$. In particular, if $\mathcal{C} = \mathcal{F}_{\mathbf{N}}$, the free ccc with weak natural numbers object, there are a finite number of additional equations (as schema) that, when added to the original data, guarantee that every type has a Mal'cev operator and $\mathbf{N}$ is a natural numbers object.*

2.4. *Polymorphism*

"The perplexing subject of polymorphism."
C. Darwin, *Life & Lett*, 1887

Although Darwin was speaking of biology, he might very well have been discussing computer science 100 years later. Christopher Strachey in the 1960's introduced various notions of polymorphism into programming language design (see [Rey83,Mit96]). Perhaps the most influential was his notion of *parametric polymorphism*. Intuitively, a parametric polymorphic function is one which has a *uniformly given algorithm at all types*. Imagine a "generic" algorithm capable of being instantiated at any arbitrary type, but which is the "same algorithm" at each type instance. It is this idea of the "plurality of form" which inspired the biological metaphor.

EXAMPLE 2.34 (*Reverse*). Consider a simple algorithm that takes a finite list and reverses it. Here "lists" could mean: lists of natural numbers, lists of reals, lists of arrays, indeed lists of lists of .... The point is, the types do not matter: we have a uniform algorithm for all types. Let $list(\alpha)$ denote the type of finite lists of entities of type α. We thus might type this algorithm

$$rev_\alpha : list(\alpha) \Rightarrow list(\alpha) \quad \text{where } rev_\alpha(a_1, \ldots, a_n) = (a_n, \ldots, a_1).$$

A second example, discussed by Strachey, is

EXAMPLE 2.35 (*Map-list*). This algorithm begins with a function of type $\alpha \Rightarrow \beta$ and a finite α-list, applies the function to each element of the list, and then makes a β-list of the subsequent values. We might represent it as:

$$map_{\alpha,\beta} : (\alpha \Rightarrow \beta) \Rightarrow (list(\alpha) \Rightarrow list(\beta))$$

where $map_{\alpha,\beta}(f)(a_1, \ldots, a_n) = (f(a_1), \ldots, f(a_n))$.

Many recent programming languages (e.g., ML, Ada) support sophisticated uses of generic types and polymorphism. The mathematical foundations of such languages were a major challenge in the past decade and category theory played a fundamental role. We shall briefly recall the issues.

2.4.1. *Polymorphic lambda calculi.* The logician J.-Y. Girard [Gi71,Gi72] in a series of important works examined higher-order logic from the Curry–Howard viewpoint. He developed formal calculi of variable types, the so-called polymorphic lambda calculi, which correspond to proofs in higher-order logics. At the same time he developed the proof theory of such systems. J. Reynolds [Rey74] independently discovered the second-order fragment of Girard's system, and proposed it as a syntax representing Strachey's parametric polymorphism.

Let us briefly examine Girard's *System* $\mathcal{F}$, second order polymorphic lambda calculus. The underlying logical system is intuitionistic second order propositional calculus. The latter theory is similar to ordinary propositional calculus, except we can universally quantify over propositional variables.

The syntax of second order propositional calculus is presented in Figure 5. The usual notions of free and bound variables in formulas are assumed. For example, in $\forall\alpha(\alpha \Rightarrow \beta)$, α is a bound variable, while β is free. $A[B/\alpha]$ denotes A with formula B substituted for free α, changing bound variables if necessary to avoid clashes. Notice in the quantifier rules that when we instantiate a universally quantified formula to obtain, say, $\Gamma \vdash A[B/\alpha]$,

Formulas	$A ::= vbl \mid A_1 \Rightarrow A_2 \mid \forall\alpha.A$
Provability	$\vdash$ is a relation between finite sets of formulas and formulas
	$\Gamma \vdash A$ if $A \in \Gamma$
	$\dfrac{\Gamma \cup \{A\} \vdash B}{\Gamma \vdash A \Rightarrow B}$, $\dfrac{\Gamma \vdash A \quad \Delta \vdash A \Rightarrow B}{\Gamma \cup \Delta \vdash B}$
	$\dfrac{\Gamma \vdash A(\alpha)}{\Gamma \vdash \forall\alpha A(\alpha)}$, $\dfrac{\Gamma \vdash \forall\alpha A(\alpha)}{\Gamma \vdash A[B/\alpha]}$
	where $\alpha \notin FV(\Gamma)$ for any formula B.

Fig. 5. Second order intuitionistic propositional calculus.

the formula B may be of arbitrary logical complexity. Thus inductive proof techniques based on the complexity of subformulas are not available in higher-order logic. This is the essence of the problem of *impredicativity* in polymorphism.

We now introduce Girard's second order lambda calculus. We use the notation $FV(t)$ and $BV(t)$ for the set of free and bound variables of term t, respectively. We write $FTV(A)$ and $BTV(A)$ for the set of free type variables and bound type variables of formula A, respectively.

DEFINITION 2.36 (*Girard's System $\mathcal{F}$*).

Types: Freely generated from type variables $\alpha, \beta, \ldots$ by the rules: if A, B are types, so are $A \Rightarrow B$ and $\forall\alpha.A$.

Terms: Freely generated from variables x_i^A of every type A by

(1) First-order lambda calculus rules: if $f : A \Rightarrow B$, $a : A$, $\varphi : B$ then $f\,`a : B$ and $\lambda_{x:A}.\varphi : A \Rightarrow B$.

(2) Specifically second-order rules:
   (a) If $t : A(\alpha)$, then $\Lambda\alpha.t : \forall\alpha A(\alpha)$ where $\alpha \notin FTV(FV(t))$,
   (b) If $t : \forall\alpha A(\alpha)$ then $t[B] : A[B/\alpha]$ for any type B.

Equations: Equality is the smallest congruence relation closed under β and η for both lambdas, that is:

(3) $(\lambda_{x:A}.\varphi)\,`a =_{\beta^1} \varphi[a/x]$ and $\lambda_{x:A}(f\,`x) =_{\eta^1} f$, where $x \notin FV(f)$.

(4) $(\Lambda\alpha.\psi)[B] =_{\beta^2} \psi[B/\alpha]$ and $\Lambda\alpha.t[\alpha] =_{\eta^2} t$, where $\alpha \notin FTV(t)$.

Eqs. (3) are the first order $\beta\eta$ equations, while Eqs. (4) are second order $\beta\eta$.

From the Curry–Howard viewpoint, the types of $\mathcal{F}$ are precisely the formulas of second order propositional calculus (Fig. 5), while terms denote proofs. For example, to annotate second order rules we have:

$$\frac{\vec{x} : \Gamma \vdash t : A(\alpha)}{\vec{x} : \Gamma \vdash \Lambda\alpha \cdot t : \forall\alpha A(\alpha)}, \qquad \frac{\vec{x} : \Gamma \vdash t : \forall\alpha A(\alpha)}{\vec{x} : \Gamma \vdash t[B] : A[B/\alpha]}.$$

The $\beta\eta$ equations of course express equality of proof trees.

What about polymorphism? Suppose we think of a term $t : \forall\alpha A(\alpha)$ as an algorithm of type $A(\alpha)$ varying uniformly over all types α. Then $t[B] : A[B/\alpha]$ is the instantiation of t at the specific type B. Moreover, B may be arbitrarily complex. Thus the type variable acts as a parameter.

In System $\mathcal{F}$ we can internally represent common inductive data types within the syntax as *weak* T-algebras, for covariant definable functors T. Weakness refers to the categorical fact that these structures satisfy existence but not uniqueness of the mediating arrow in the universal mapping property. Thus, for any types A, B we are able to define the types $\mathbf{1}$, **Nat**, $List(A)$, $A \times B$, $A + B$, $\exists\alpha \cdot A$, etc. (see [GLT] for a full treatment).

Let us give two examples and at the same time illustrate polymorphic instantiation.

EXAMPLE 2.37. The type of Booleans is given by

$$\mathbf{Boole} = \forall\alpha.(\alpha \Rightarrow (\alpha \Rightarrow \alpha)).$$

It has two distinguished elements $\mathbf{T}, \mathbf{F}$: **Boole** given by $\mathbf{T} = \Lambda\alpha.\lambda_{x:\alpha}.\lambda_{y:\alpha}.x$ and $\mathbf{F} = \Lambda\alpha.\lambda_{x:\alpha}.\lambda_{y:\alpha}.y$, together with a *Definition by Cases* operator (for each type A) $D_A : A \Rightarrow (A \Rightarrow (\mathbf{Boole} \Rightarrow A))$ defined by $D_A uvt = (t[A]\text{`}u)\text{`}v$ where $u, v : A$, $t : \mathbf{Boole}$. One may easily verify that $D_A uv\mathbf{T} =_\beta u$ and $D_A uv\mathbf{F} =_\beta v$ (where β stands for $\beta^1 \cup \beta^2$).

EXAMPLE 2.38. The type of (Church) numerals

$$\mathbf{Nat} = \forall\alpha((\alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha)).$$

The numeral $\mathbf{n} : \mathbf{Nat}$ corresponds at each α to n-fold composition $f \mapsto f^n$, where $f^n = f \circ f \circ \cdots \circ f$ (n times) and $f^0 = Id_\alpha = \lambda x_\alpha \cdot x$. Formally, it is the closed term $\mathbf{n} = \Lambda\alpha.\lambda_{f:\alpha\Rightarrow\alpha}.f^n : \mathbf{Nat}$. Thus for *any* type B we have a uniform algorithm: $\mathbf{n}[B] = \lambda_{f:B\Rightarrow B}.f^n : (B \Rightarrow B) \Rightarrow (B \Rightarrow B)$. Successor is given by $\mathbf{S} = \lambda_{n:\mathbf{Nat}}.n + 1$, where $n + 1 = \Lambda\alpha.\lambda_{f:\alpha\Rightarrow\alpha}.f^{n+1} = \Lambda\alpha.\lambda_{f:\alpha\Rightarrow\alpha}.f \circ f^n = \Lambda\alpha.\lambda_{f:\alpha\Rightarrow\alpha} f \circ (n[\alpha]\text{`}f)$. Finally, iteration is given by: if $a : A$, $h : A \Rightarrow A$, $\mathcal{I}_{ah} = \lambda_{x:\mathbf{Nat}}.(x[A]\text{`}h)\text{`}a$: $\mathbf{Nat} \Rightarrow A$. The reader may easily calculate that $\mathcal{I}_{ah}\mathbf{0} =_\beta a$ and $\mathcal{I}_{ah}(\mathbf{n} + \mathbf{1}) =_\beta h\text{`}(\mathcal{I}_{ah}\mathbf{n})$ for numerals $\mathbf{n}$.

Let us illustrate the power of impredicativity in this situation. See the discussion of Church vs. Curry typing, Section 2.5.3. Notice that for any type B, $\mathbf{n}[B \Rightarrow B]\text{`}\mathbf{n}[B]$ makes perfectly good sense. In particular, let $B = \mathbf{Nat}$, the type of $\mathbf{n}$ itself. This is a well-defined term and if we erase all its types we obtain the *untyped* expression $\mathbf{n}\text{`}\mathbf{n} = \lambda f.f^n$. This latter untyped term is *not* typable in simply typed lambda calculus.

Formal systems describing far more powerful versions of polymorphism have been developed. For example, Girard's thesis described the typed lambda calculus corresponding to ω-order intuitionist type theory, so-called $\mathcal{F}_\omega$. Programming in the various levels of Girard's theories $\{\mathcal{F}_n\}$, $n = 1, 2, \ldots, \omega$, is described in [PDM89]. Other systems include Coquand–Huet's Calculus of Constructions and its extensions [Luo94]. These theories include not only Girard's $\mathcal{F}_\omega$ but also Martin-Löf's dependent type theories [H97a]. Indeed, these theories are among the most powerful logics known, yet form the basis of various proof-development systems (e.g., LEGO and Coq) [LP92,D+93].

2.4.2. *What is a model of System $\mathcal{F}$?* The problem of finding – and indeed defining precisely – a model of System $\mathcal{F}$ was difficult. Cartesian closedness is not the issue. The problem, of course, is the universal quantifier: clearly in $\forall\alpha.A$ the α is to range over all the objects of the model, and at the same time $\forall$ should be interpreted as some kind of product (over *all* objects). Such "large" products create havoc, as foreshadowed in the following theorem of Freyd (cf. [Mac71], Proposition 3, p. 110).

THEOREM 2.39 (Freyd). *A small category which is small complete is a preorder.*

Cartesian closed preorders (e.g., complete Heyting algebras) are of no interest for modelling proofs; we seek "nontrivial" categories.

Suppose instead we try to define a naive "set-theoretic" model of System $\mathcal{F}$, in which $\times, \Rightarrow$ have their usual meaning, and $\forall\alpha$ is interpreted as a "large" product. Such models are defined in detail in [RP,Pi87]. John Reynolds proved the following

THEOREM 2.40. *There is no* **Set** *model for System* $\mathcal{F}$.

There is an elegant categorical proof in Reynolds and Plotkin [RP]. Let us sketch the proof, which applies to somewhat more general categories than **Set**.

Let $\mathcal{C}$ be a category with an endofunctor $T : \mathcal{C} \to \mathcal{C}$. A *$T$-algebra* is an object A together with an arrow $TA \xrightarrow{a} A$. A morphism of T-algebras is a commutative square:

$$\begin{array}{ccc} TA & \xrightarrow{Tf} & TB \\ \downarrow{\scriptstyle a} & & \downarrow{\scriptstyle b} \\ A & \xrightarrow{f} & B \end{array}$$

An *initial T-algebra* (resp. *weakly initial T-algebra*) is one for which there exists a unique morphism (resp. there exists a morphism) to any other T-algebra.

We shall be interested in objects and arrows of the model category $\mathcal{C}$ which are "definable", i.e. denoted by types and terms of System $\mathcal{F}$. There are simple covariant endofunctors T on $\mathcal{C}$ whose action on objects is definable by types and whose actions on arrows is definable by terms (of System $\mathcal{F}$). For example, the identity functor $T(\alpha) = \alpha$ and the functor $T(\alpha) = (\alpha \Rightarrow B) \Rightarrow B$, for any fixed B, have this property.

Now it may be shown (see [RP]) that for any definable functor T, the System $\mathcal{F}$ expression $P = \forall\alpha.(T(\alpha) \Rightarrow \alpha) \Rightarrow \alpha$ is a weakly initial T-algebra. *Suppose the ambient model category $\mathcal{C}$ has equalizers of all subsets of arrows* (e.g., **Set** has this property). Essentially by taking a large equalizer (cf. the Solution Set Condition in Freyd's Adjoint Functor Theorem, [Mac71], p. 116) we could then construct a subalgebra of P which is an initial T-algebra. Call this initial T-algebra $\mathcal{I}$. We then use the following important observation of Lambek:

PROPOSITION 2.41 (Lambek). *If* $T(\mathcal{I}) \xrightarrow{f} \mathcal{I}$ *is an initial T-algebra, then f is an isomorphism.*

Applying this to the definable functor $T(\alpha) = (\alpha \Rightarrow B) \Rightarrow B$, we observe that $T(\mathcal{I}) \cong \mathcal{I}$. In particular, let $\mathcal{C} =$ **Set** and $B =$ **Boole**, and take the usual **Set** interpretation of $\times$ as Cartesian product and $\Rightarrow$ as the full function space. Notice $card(B) \geqslant 2$ (since there are always the two distinct closed terms **T** and **F**). Hence we obtain a bijection $B^{B^{\mathcal{I}}} \cong \mathcal{I}$, for some set $\mathcal{I}$, which is impossible for cardinality reasons.

The search for models of System $\mathcal{F}$ led to some extraordinary phenomena that had considerable influence in semantics of programming languages. Let us just briefly mention the history. Notice that the Reynolds–Plotkin proof depends on a simple cardinality argument, which itself depends on classical set theory. Similarly, the proof of Freyd's result, Theorem 2.39, depends on using classical (i.e. non-intuitionistic) logical reasoning in the

metalanguage. This suggests that it is really the non-constructive nature of the category **Sets** that is at fault; if we were to work within a non-classical universe – say within a model of intuitionistic set theory – there is still a chance that we could escape the above problems but still have a "set-theoretical" model of System $\mathcal{F}$. And, from one point of view, that is exactly what happened.

These ambient categories, called toposes [LS86,MM92], are in general models of intuitionistic higher-order logic (or set-theory), and include such categories as functor categories and sheaves on a topological space, as well as **Sets**. Moggi suggested constructing models of System $\mathcal{F}$ based on an internally complete internal full subcategory of a suitable ambient topos. This ambient topos would serve as our constructive set-theory, and function types would still be interpreted as the full "set-theoretical" space of total functions. M. Hyland [Hy88] proved that the Realizability (or Effective) Topos had (non-trivial) such internal category objects. The difficult development and clarification of these internal models was undertaken by many researchers, e.g., D. Scott, M. Hyland, E. Robinson, P. Rosolini, A. Carboni, P. Freyd, A. Scedrov, A. Pitts et al. (e.g., [HRR,Rob89,Ros90,CPS88,Pi87]).

In a separate development, R. Seely [See87] gave the first general categorical definition of a so-called *external model* of System $\mathcal{F}$, and more generally $\mathcal{F}_\omega$. The definition was based on the theory of *indexed* or *fibred* categories. This view of logic was pioneered by Lawvere [Law69] who emphasized that quantifiers were interpretable as adjoint functors. Pitts [Pi87] clarified the relationship between Seely's models and internal-category models within ambient toposes of presheaves. Moreover, he showed that there are enough such internal models for a Completeness Theorem. It is worth remarking that Pitts' work uses properties of Yoneda embeddings. For general expositions see [AL91]. Extensions of "set-theoretical" models to cases where function spaces include *partial functions* (i.e. non-termination) is in [RR90].

One can *externalize* these internal category models [Hy88,AL91] to obtain ordinary categories. And one such internal category in the Realizability Topos, the *modest sets*, when externalized is precisely the ccc category $Per(\mathbf{N})$ discussed in Section 2.1.

PROPOSITION 2.42. *$Per(\mathbf{N})$ is a model of System $\mathcal{F}$.*

The idea is that in addition to the ccc structure of $Per(\mathbf{N})$, we interpret $\forall$ as a large intersection (the intersection of an arbitrary family of pers is again a per). We shall return to this example in Section 3.2.

Ironically, in essence this model was already in Girard's original PhD thesis [Gi72]. Later, domain-theoretic models of System $\mathcal{F}$ were considered by Girard in [Gi86] and were instrumental in his development of linear logic

2.5. *The untyped world*

The advantages of types in programming languages are familiar and well-documented (e.g., [Mit96]). Nonetheless, there is an underlying *untyped* aspect of computation, already going back to the original work on lambda calculus and combinatory logic in the 1930's, which often underlies concrete machine implementations. In this early view, developed