

Basic Simple Type Theory

J. Roger Hindley

CAMBRIDGE TRACTS
IN THEORETICAL
COMPUTER SCIENCE

52

BASIC SIMPLE TYPE THEORY

Cambridge Tracts in Theoretical Computer Science

Editorial Board

S. Abramsky, *Department of Computing Science, Edinburgh University*
P.H. Aczel, *Department of Computer Science, University of Manchester*
J.W. de Bakker, *Centrum voor Wiskunde en Informatica, Amsterdam*
J.A. Goguen, *Programming Research Group, University of Oxford*
Y. Gurevich, *Department of Electrical Engineering and Computer Science, University of Michigan*
J.V. Tucker, *Department of Mathematics and Computer Science, University College of Swansea*

Titles in the series

1. G. Chaitin *Algorithmic Information Theory*
2. L.C. Paulson *Logic and Computation*
3. M. Spivey *Understanding Z*
4. G. Revesz *Lambda Calculus, Combinators and Functional Programming*
5. A. Ramsay *Formal Methods in Artificial Intelligence*
6. S. Vickers *Topology via Logic*
7. J.-Y. Girard, Y. Lafont & P. Taylor *Proofs and Types*
8. J. Clifford *Formal Semantics & Pragmatics for Natural Language Processing*
9. M. Winslett *Updating Logical Databases*
10. K. McEvoy & J.V. Tucker (eds) *Theoretical Foundations of VLSI Design*
11. T.H. Tse *A Unifying Framework for Structured Analysis and Design Models*
12. G. Brewka *Nonmonotonic Reasoning*
14. S.G. Hoggar *Mathematics for Computer Graphics*
15. S. Dasgupta *Design Theory and Computer Science*
17. J.C.M. Baeten (ed) *Applications of Process Algebra*
18. J.C.M. Baeten & W.P. Weijland *Process Algebra*
19. M. Manzano *Extensions of First Order Logic*
21. D.A. Wolfram *The Clausal Theory of Types*
22. V. Stoltenberg-Hansen, I. Lindström & E. Griffor *Mathematical Theory of Domains*
23. E.-R. Olderog *Nets, Terms and Formulas*
26. P.D. Mosses *Action Semantics*
27. W.H. Hesselink *Programs, Recursion and Unbounded Choice*
28. P. Padawitz *Deductive and Declarative Programming*
29. P. Gärdenfors (ed) *Belief Revision*
30. M. Anthony & N. Biggs *Computational Learning Theory*
31. T.F. Melham *Higher Order Logic and Hardware Verification*
32. R.L. Carpenter *The Logic of Typed Feature Structures*
33. E.G. Manes *Predicate Transformer Semantics*
34. F. Nielson & H.R. Nielson *Two Level Functional Languages*
35. L. Feijs & H. Jonkers *Formal Specification and Design*
36. S. Mauw & G.J. Veltink (eds) *Algebraic Specification of Communication Protocols*
37. V. Stavridou *Formal Methods in Circuit Design*
38. N. Shankar *Metamathematics, Machines and Gödel's Proof*
39. J.B. Paris *The Uncertain Reasoner's Companion*
40. J. Desel & J. Esparza *Free Choice Petri Nets*
41. J.-J. Ch. Meyer & W. van der Hoek *Epistemic Logic for AI and Computer Science*
42. J.R. Hindley *Basic Simple Type Theory*

BASIC SIMPLE TYPE THEORY

J. Roger Hindley

University of Wales, Swansea



PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge CB2 1RP, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, United Kingdom
40 West 20th Street, New York, NY 10011-4211, USA
10 Stamford Road, Oakleigh, Melbourne 3166, Australia

© Cambridge University Press 1997

This book is in copyright. Subject to statutory exception
and to the provisions of relevant collective licensing agreements,
no reproduction of any part may take place without
the written permission of Cambridge University Press.

First published 1997

Typeset in 10/13 point Times

A catalogue record for this book is available from the British Library

Library of Congress Cataloguing in Publication data

Hindley, J. Roger.

Basic simple type theory / J. Roger Hindley.

p. cm. – (Cambridge tracts in theoretical computer science ; 42)

Includes bibliographical references and index.

ISBN 0 521 46518 4

1. Programming languages (Electronic computers) 2. Type theory.

I. Title. II. Series.

QA76.7.H55 1996

005.13–dc20 95-9058 CIP

ISBN 0 521 46518 4 hardback

Transferred to digital printing 2002

To Carol

Contents

<i>Introduction</i>	<i>page</i>	<i>ix</i>
1 The type-free λ-calculus		1
1A λ -terms and their structure		1
1B β -reduction and β -normal forms		4
1C η - and $\beta\eta$ -reductions		7
1D Restricted λ -terms		10
2 Assigning types to terms		12
2A The system TA_λ		12
2B The subject-construction theorem		20
2C Subject reduction and expansion		24
2D The typable terms		27
3 The principal-type algorithm		30
3A Principal types and their history		31
3B Type-substitutions		34
3C Motivating the PT algorithm		38
3D Unification		40
3E The PT algorithm		44
4 Type assignment with equality		52
4A The equality rule		52
4B Semantics and completeness		57
5 A version using typed terms		63
5A Typed terms		63
5B Reducing typed terms		67
5C Normalization theorems		71
6 The correspondence with implication		74
6A Intuitionist implicational logic		74
6B The Curry-Howard isomorphism		79
6C Some weaker logics		85
6D Axiom-based versions		88
7 The converse principal-type algorithm		93
7A The converse PT theorems		93
7B Identifications		95

7C	The converse PT proof	96
7D	Condensed detachment	102
8	Counting a type's inhabitants	108
8A	Inhabitants	108
8B	Examples of the search strategy	114
8C	The search algorithm	118
8D	The Counting algorithm	124
8E	The structure of a nf-scheme	127
8F	Stretching, shrinking and completeness	132
9	Technical details	140
9A	The structure of a term	140
9B	Residuals	144
9C	The structure of a TA_λ -deduction	148
9D	The structure of a type	151
9E	The condensed structure of a type	153
9F	Imitating combinatory logic in λ -calculus	157
	<i>Answers to starred exercises</i>	161
	<i>Bibliography</i>	169
	<i>Table of principal types</i>	177
	<i>Index</i>	179

Introduction

This book is not about type theories in general but about one very neat and special system called “TA” for “type-assignment”. Its types contain type-variables and arrows but nothing else, and its terms are built by λ -abstraction and application from term-variables and nothing else. Its expressive power is close to that of the system called *simple type theory* that originated with Alonzo Church.

TA is polymorphic in the sense that a term can have more than one type, indeed an infinite number of types. On the other hand the system has no \forall -types and hence it is weaker than the strong polymorphic theories in current use in logic and programming. However, it lies at the core of nearly every one of them and its properties are so distinctive and even enjoyable that I believe the system is worth isolating and studying on its own. That is the aim of this book. In it I hope to try to pass on to the reader the pleasure the system’s properties have given me.

TA is also an excellent training ground for learning the techniques of type-theory as a whole. Its methods and algorithms are not trivial but the main lines of most of them become clear once the basic concepts have been understood. Many ideas that are complicated and tedious to formulate for stronger type-theories, and many complex techniques for analysing structures in these theories, appear in TA in a very clean and neat stripped-down form. This book will take advantage of this neatness to introduce some of the most important type-theoretic techniques with particular emphasis on explaining why things happen the way they do.

Thus the reader who learns the basic techniques of type-theory from TA will acquire a very good foundation for the study of other type-systems.

Type theories in general date back to the philosopher Bertrand Russell and beyond.¹ They were used in the early 1900’s for the very specific purpose of getting round the paradoxes that had shaken the foundations of mathematics at that time, but their use was later widened until they came to be part of the logicians’ standard bag of technical tools, especially in proof-theory. (Their use in combinatory logic dates back to Curry 1934 and in λ -calculus to Church 1940.) However, they remained a relatively specialist tool until around the 1970’s.

About that time the need for stronger programming languages brought type-theories to the attention of computer scientists, and several of the new languages developed in the 1970’s and 80’s were built on a type-theory base. These languages

¹ See for example Russell 1903 Appendix B and the comments in Gandy 1977 and Church 1976.

have proved themselves in many applications and have now become well established in the research community (and are even becoming known outside it!). The chief example is ML, developed at Edinburgh University by the group led by Robin Milner, but others include HOL (Cambridge University), Miranda (Regd trademark, Research Software Ltd.) and Nuprl (Cornell University).

The system TA is, with slight modifications, a common part of all of these. Indeed, in its early days it was studied mainly as a prelude to studies of stronger systems and this is the way it was treated in Curry and Feys 1958.¹ But from the 1960's onward it gradually became clear that TA was not as trivial as it had at first seemed and was worth isolating and studying in its own right. Natural questions about TA turned out to be much harder to answer than expected. Their answers are not completed even today, but from them have come some very interesting techniques that have had applications elsewhere, such as type-checking algorithms and filter λ -models.

In fact more is now known about TA than can fit into a book of reasonable length. The present book will therefore be very selective. Although covering all the main basic properties of TA it will focus on the following three algorithms. The first is well known but the other two are scarcely known at all and their consequences are still not nearly fully understood, though they were first discovered well over fifteen years ago.

(1) *Type-checking or principal-type algorithm.* This algorithm is the core of the type-checking algorithm used in ML. It takes a λ -term and decides whether a type can be assigned to it and, if so, outputs the most general such type (the *principal type* of the term). The method behind the algorithm was sketched in Curry and Feys 1958 and versions of the algorithm itself have appeared in Morris 1968, Curry 1969, Hindley 1969 and Milner 1978.

(2) *Converse principal-type algorithm.* This algorithm takes a closed λ -term M and any type τ which can be assigned to M , and outputs a closed λ -term M^* such that τ is the principal type of M^* . Via the formulae-as-types correspondence with propositional logic it leads to a completeness proof for a variant of the Resolution rule called the *rule of condensed detachment* in a system of implicational logic. In fact several converse principal-type algorithms are known, each producing an M^* with slightly different properties and giving completeness for a slightly different logic. (Hindley 1969, Meyer and Bunder 1988, Mints and Tammet 1991.)

(3) *Inhabitant-counting algorithm.* A *normal inhabitant* of a type τ is a closed λ -term M in β -normal form to which τ can be assigned. The counting-algorithm takes a type τ and outputs the number of its normal inhabitants (0, 1, 2, ... or infinity, modulo changes of bound variables), and then lists these one by one; in particular it decides in a finite time whether this list will be infinite or not. It is like the known algorithm for deciding whether a regular language is infinite but with extra procedures to deal with bound variables. It is also like known algorithms for deciding provability in Intuitionist propositional logic but with extra procedures

¹ Barendregt 1992 is a good survey of type-theories which shows the relations between TA_λ and others; see especially §3.1 where TA_λ is called $\lambda \rightarrow$ -Curry. Other modern introductions to type theories in general are Andrews 1986, Girard et al. 1989, Krivine 1990, Mitchell 1990 and 1996, Constable 1991, Gallier 1993, Nerode and Odifreddi 199-, Scedrov 1990.

added to do the counting. It originated in Ben-Yelles 1979 though part of the proof that it works is due to Hirokawa 1993c.

Each of the above algorithms will be presented in full with a proof of correctness included.

Acknowledgements

I am very grateful to all who have helped in producing this book, especially the following.

For invitations to give the lecture-courses from which this book has (very slowly) developed: Institut d' Informatique, Université de Tizi-Ouzou, Algeria (1988), and the U.K. Science and Engineering Research Council's "Logic for I.T." Initiative (1990).

For a period of reduced teaching duties without which the book would not have been possible: my colleagues in the Mathematics Department, University of Wales Swansea.

For finance and accommodation which facilitated some of the meetings and discussions involved: the European Community's "Esprit" Basic Research Actions 3230 and 7232, the Australian Research Grants Scheme, and the Mathematics Department of the University of Wollongong.

For useful criticisms, discussions, and helpful suggestions: Yohji Akama, Choukri-Bey Ben-Yelles, Martin Bunder, Naim Çağman, Felice Cardone, Mariangiola Dezani-Ciancaglini, Roy Dyckhoff, Fritz Henglein, Sachio Hirokawa, Hans Leiss, Mohamed Mezghiche, Seref Mirasyedioglu, Gordon Plotkin, Adrian Rezus, Jon Seldin, Masako Takahashi, Anne Troelstra, Werner Wolff, Marek Zaionc and anonymous referees. (Any errors remaining in the work are my own responsibility, however.)

For perseverance: the staff of Cambridge University Press, especially David Tranah.

And last but not least, for a healthy combination of support and cynicism, my wife Carol.

Notes on the text

Chapter 1 contains a very short summary of all the basic facts about λ -calculus needed in this book, though the reader is assumed to have met λ -calculus before. Further information can be found in standard text-books; for example (in English) Barendregt 1984, Curry and Feys 1958, Hindley and Seldin 1986, Révész 1988, Hankin 1994, or (in French or English) Krivine 1990, or (in Japanese) Takahashi 1991.

Chapter 9 at the end of the book collects together some technical details that are needed in the correctness-proofs of the main algorithms: readers who prefer to omit these proofs can omit this chapter too.

Some exercises are provided in the text. Answers to starred ones are at the end of the book.

Throughout this book "HS 86" refers to Hindley and Seldin 1986.

The type-free λ -calculus

The λ -calculus is a family of prototype programming languages invented by a logician, Alonzo Church, in the 1930's. Their main feature is that they are *higher-order*; that is, they give a systematic notation for operators whose input and output values may be other operators. Also they are *functional*, that is they are based on the notion of *function* or *operator* and include notation for function-application and abstraction.

This book will be about the simplest of these languages, the *pure* λ -calculus, in which λ -terms are formed by application and abstraction from variables only. No atomic constants will be allowed.

1A λ -terms and their structure

1A1 Definition (λ -terms) An infinite sequence of *term-variables* is assumed to be given. Then linguistic expressions called λ -*terms* are defined thus:

- (i) each term-variable is a λ -term, called an *atom* or *atomic term*;
- (ii) if M and N are λ -terms then (MN) is a λ -term called an *application*;
- (iii) if x is a term-variable and M is a λ -term then $(\lambda x \cdot M)$ is a λ -term called an *abstract* or a λ -*abstract*.

A *composite* λ -term is a λ -term that is not an atom.

1A1.1 Notation *Term-variables* are denoted by " u ", " v ", " w ", " x ", " y ", " z ", with or without number-subscripts. Distinct letters denote distinct variables unless otherwise stated.

Arbitrary λ -terms are denoted by " L ", " M ", " N ", " P ", " Q ", " R ", " S ", " T ", with or without number-subscripts. For " λ -term" we shall usually say just "*term*".

Syntactic identity: " $M \equiv N$ " will mean that M is the same expression as N (if M and N are terms or other expressions). But for identity of numbers, sets, etc. we shall say " $=$ " as usual.

Parentheses and repeated λ 's will often be omitted in such a way that, for example,

$$\lambda xyz \cdot M \equiv (\lambda x \cdot (\lambda y \cdot (\lambda z \cdot M))), \quad MNPQ \equiv (((MN)P)Q).$$

(The rule for restoring parentheses omitted from $MNPQ$ is called *association to the left*.)

1A2 Definition The *length*, $|M|$, of a λ -term M is the number of occurrences of variables in M ; in detail, define

$$|x| = 1, \quad |MN| = |M| + |N|, \quad |\lambda x \cdot M| = 1 + |M|.$$

1A2.1 *Example* $|(\lambda x \cdot yx)(\lambda z \cdot x)| = 5$.

1A3 Definition (Subterms) The *subterms* of a term M are defined by induction on $|M|$ as follows:

- (i) an atom is a subterm of itself;
- (ii) if $M \equiv \lambda x \cdot P$, its subterms are M and all subterms of P ;
- (iii) if $M \equiv P_1 P_2$, its subterms are all the subterms of P_1 , all those of P_2 , and M itself.

1A3.1 *Example* If $M \equiv (\lambda x \cdot yx)(\lambda z \cdot x(yx))$ its subterms are x , y , yx , $\lambda x \cdot yx$, $x(yx)$, $\lambda z \cdot x(yx)$ and M itself. (But not z .)

1A4 Notation (Occurrences, components) A subterm of a term M may have more than one occurrence in M ; for example the term

$$(\lambda x \cdot yx)(\lambda z \cdot x(yx))$$

contains two occurrences of yx and three of x . The precise definition of “occurrence” is written out in 9A2, but the reader who already has a good intuitive idea of the occurrence-concept will go a long way without needing to look at this definition.

In this book occurrences will be underlined to distinguish them from subterms; for example we may say

“Let \underline{P} be any occurrence of P in M ”.

An occurrence of λx will be called an *abstractor*, and the occurrence of x in it will be called a *binding occurrence* of x .

All the occurrences of terms in M , other than binding occurrences of variables, will be called *components* of M .

1A5 Definition (Body, scope, covering abstractors) Let $\underline{\lambda x \cdot P}$ be a component of a term M . The displayed component \underline{P} is called the *body* of $\underline{\lambda x \cdot P}$ or the *scope* of the *abstractor* $\underline{\lambda x}$.

The *covering abstractors* of a component \underline{R} of M are the abstractors in M whose scopes contain \underline{R} .

1A6 Definition (Free, bound) A non-binding variable-occurrence \underline{x} in a term M is said to be *bound* in M iff it is in the scope of an occurrence of λx in M , otherwise it is *free* in M .

A variable x is said to be *bound* in M iff M contains an occurrence of λx ; and x is said to be *free* in M iff M contains a free occurrence of x . The set of all variables free in M is called

$$FV(M).$$

1A6.1 Warning Two distinct concepts have been defined here, free/bound occurrences and free/bound variables. A variable x may be both free and bound in M , for example if $M \equiv x(\lambda x \cdot x)$, but a particular occurrence of x in M cannot be both free and bound.

Also note that x is said to be bound in $\lambda x \cdot y$ even though its only occurrence there is a binding one.

1A7 Definition (Substitution) Define $[N/x]M$ to be the result of substituting N for each free occurrence of x in M and making any changes of bound variables needed to prevent variables free in N from becoming bound in $[N/x]M$. More precisely, define for all N, x, P, Q and all $y \neq x$

- (i) $[N/x]x \equiv N$,
- (ii) $[N/x]y \equiv y$,
- (iii) $[N/x](P, Q) \equiv ([N/x]P)([N/x]Q)$,
- (iv) $[N/x](\lambda x \cdot P) \equiv \lambda x \cdot P$,
- (v) $[N/x](\lambda y \cdot P) \equiv \lambda y \cdot P$ if $x \notin FV(P)$,
- (vi) $[N/x](\lambda y \cdot P) \equiv \lambda y \cdot [N/x]P$ if $x \in FV(P)$ and $y \notin FV(N)$,
- (vii) $[N/x](\lambda y \cdot P) \equiv \lambda z \cdot [N/x][z/y]P$ if $x \in FV(P)$ and $y \in FV(N)$.

(In (vii) z is the first variable in the sequence given in 1A1 which does not occur free in NP .)

1A7.1 Notation (Simultaneous substitution) For any N_1, \dots, N_n and any distinct x_1, \dots, x_n , the result of simultaneously substituting N_1 for x_1, N_2 for x_2, \dots in M , and changing bound variables to avoid clashes, is defined similarly to $[N/x]M$. (For a neat definition see Stoughton 1988 §2.) It is called

$$[N_1/x_1, \dots, N_n/x_n]M.$$

1A8 Definition (Changing bound variables, α -conversion) Let $y \notin FV(M)$; then we say

$$(\alpha) \quad \lambda x \cdot M \equiv_\alpha \lambda y \cdot [y/x]M,$$

and the act of replacing an occurrence of $\lambda x \cdot M$ in a term by $\lambda y \cdot [y/x]M$ is called a **change of bound variables**. If P changes to Q by a finite (perhaps empty) series of changes of bound variables we say P **α -converts to** Q or

$$P \equiv_\alpha Q.$$

1A8.1 Note Some basic lemmas about α -conversion and substitution are given in HS 86 §1B. Two simple properties that will be needed here are

- (i) $P \equiv_\alpha Q \implies |P| = |Q|$,
- (ii) $P \equiv_\alpha Q \implies FV(P) = FV(Q)$.

1A9 Definition A term M has a **bound-variable clash** iff M contains an abstractor λx and a (free, bound or binding) occurrence of x that is not in its scope.

Examples of terms with bound-variable clashes are

$$x(\lambda x \cdot N), \quad \lambda x \cdot \lambda y \cdot \lambda x \cdot N, \quad (\lambda x \cdot P)(\lambda x \cdot Q).$$

We shall be mainly interested in terms *without* such clashes.

1A9.1 Lemma Every term can be α -converted to a term without bound-variable clashes.

Proof By the lemmas in HS 86 §1B. □

1A10 Definition (Closed terms) A *closed term* or *combinator* is a term in which no variable occurs free.

1A10.1 Example The following closed terms will be used in examples and results throughout this book.

$$\begin{array}{ll} \mathbf{B} \equiv \lambda xyz \cdot x(yz), & \mathbf{B}' \equiv \lambda xyz \cdot y(xz), \\ \mathbf{C} \equiv \lambda xyz \cdot xzy, & \mathbf{I} \equiv \lambda x \cdot x, \\ \mathbf{K} \equiv \lambda xy \cdot x, & \mathbf{S} \equiv \lambda xyz \cdot xz(yz), \\ \mathbf{W} \equiv \lambda xy \cdot xyy, & \mathbf{Y} \equiv \lambda x \cdot (\lambda y \cdot x(yy))(\lambda y \cdot x(yy)), \\ \bar{0} \equiv \lambda xy \cdot y, & \bar{1} \equiv \lambda xy \cdot xy, \\ \bar{n} \equiv \lambda xy \cdot x^n y \equiv \lambda xy \cdot x(x(\dots(xy)\dots)) \quad (n \text{ x's applied to } y). \end{array}$$

(\mathbf{Y} is Curry's *fixed-point combinator*, see HS 86 Ch.3 §3B for background; the terms \bar{n} are the *Church numerals* for $n = 0, 1, 2, \dots$, see HS 86 Def. 4.2.)

1B β -reduction and β -normal forms

This section outlines the definition and main properties of the term-rewriting procedure called β -reduction. Further details can be found in many other books, for example HS 86 Chs. 1–6 and Barendregt 1984 Chs. 3 and 11–14.

1B1 Definition (β -contraction) A β -*redex* is any term $(\lambda x \cdot M)N$; its *contractum* is $[N/x]M$ and its *re-write rule* is

$$(\lambda x \cdot M)N \triangleright_{1\beta} [N/x]M.$$

Iff P contains a β -redex-occurrence $\underline{R} \equiv (\lambda x \cdot M)N$ and Q is the result of replacing this by $[N/x]M$, we say P β -*contracts to* Q ($P \triangleright_{1\beta} Q$) and we call the triple $\langle P, \underline{R}, Q \rangle$ a β -*contraction of* P .

1B1.1 Lemma $P \triangleright_{1\beta} Q \implies FV(P) \supseteq FV(Q)$.

1B2 Definition (β -reduction) A β -*reduction* of a term P is a finite or infinite sequence of β -contractions with form

$$(i) \quad \langle P_1, \underline{R}_1, Q_1 \rangle, \quad \langle P_2, \underline{R}_2, Q_2 \rangle, \quad \dots$$

where $P_1 \equiv_\alpha P$ and $Q_i \equiv_\alpha P_{i+1}$ for $i = 1, 2, \dots$ (The empty sequence is allowed.) We say a finite reduction is *from* P *to* Q iff either it has $n \geq 1$ contractions and $Q_n \equiv_\alpha Q$

or it is empty and $P \equiv_\alpha Q$. A reduction from P to Q is said to *terminate* or *end* at Q . If there is a reduction from P to Q we say P β -reduces to Q , or

$$P \triangleright_\beta Q.$$

Note that α -conversions are allowed in a β -reduction.

1B3 Definition The *length* of a β -reduction is the number of its β -contractions (finite or ∞). A reduction with *maximal length* is one that continues as long as there are redexes to contract (i.e. one that either is infinite or ends at a term containing no redexes).

1B4 Definition (β -conversion) If we can change P to Q by a finite sequence of β -reductions and reversed β -reductions, we say P β -converts to Q , or P is β -equal to Q , or

$$P =_\beta Q.$$

A reversed β -reduction is called a β -expansion.

1B4.1 Exercise For every term F let $X_F \equiv \mathbf{Y}F$ where \mathbf{Y} is the fixed-point combinator defined in 1A10.1; show that

$$FX_F =_\beta X_F.$$

1B5 Church-Rosser Theorem for β (i) If $M \triangleright_\beta P$ and $M \triangleright_\beta Q$ (see Fig. 1B5a) then there exists T such that

$$P \triangleright_\beta T, \quad Q \triangleright_\beta T.$$

(ii) If $P =_\beta Q$ (see Fig. 1B5b) then there exists T such that

$$P \triangleright_\beta T, \quad Q \triangleright_\beta T.$$

Proof of 1B5 (i) See HS 86 Appendix 1 or Barendregt 1984 §3.2. (ii) This is deduced from (i) as suggested in Fig. 1B5b. \square

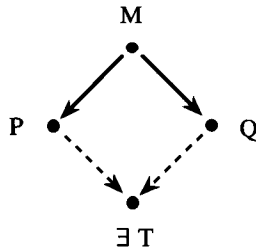


Fig. 1B5a.

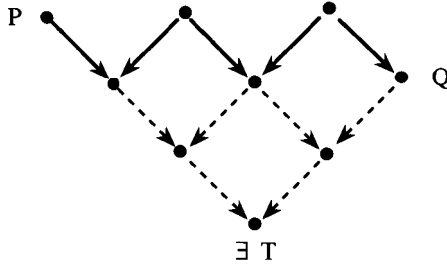


Fig. 1B5b.

1B6 Definition (β -normal forms) A β -normal form is a term that contains no β -redexes. The class of all β -nf's is called β -nf. We say a term M has β -nf N iff

$$M \triangleright_{\beta} N \text{ and } N \in \beta\text{-nf}.$$

1B6.1 Note Roughly speaking, a reduction can be thought of as a computation and a β -nf as its result. One main aim when designing a type-theory is to give it the property that every computation can be pursued to a result if the operator wishes, i.e. that every term with a type has a β -nf. This gives normal forms even more significance in a type-theory than they already have in a type-free theory.

(Terms in general do not necessarily have β -nf's of course. The simplest term without one is $(\lambda x \cdot xx)(\lambda x \cdot xx)$.)

1B7 NF-Uniqueness Lemma Modulo α -conversion, a term M has at most one β -nf.

Proof An easy application of the Church-Rosser theorem. □

1B7.1 Notation If M has a β -nf it will be called $M_{\star\beta}$.

1B8 Definition (Leftmost reductions) The *leftmost β -redex-occurrence* in a term P is the β -redex-occurrence whose leftmost parenthesis is to the left of all the parentheses in all the other β -redex-occurrences in P .

The *leftmost β -reduction* of a term P is a β -reduction of P with maximal length, say

$$\langle P_1, \underline{R}_1, Q_1 \rangle, \quad \langle P_2, \underline{R}_2, Q_2 \rangle, \quad \dots,$$

such that \underline{R}_i is the leftmost β -redex-occurrence in P_i for all $i \geq 1$ (and P_i α -converts to P and P_{i+1} α -converts to Q_i for all $i \geq 1$).

1B9 Leftmost-reduction Theorem A term M has a β -nf $M_{\star\beta}$ iff the leftmost β -reduction of M is finite and ends at $M_{\star\beta}$.

Proof See Curry and Feys 1958 §4E Cor. 1.1. (In fact this result is an immediate corollary of a slightly deeper result called the *standardization theorem*; for the latter see Curry and Feys 1958 §4E Thm. 1 or Barendregt 1984 Thm. 11.4.7, or the particularly clear proof in Mitschke 1979 Thm. 7.) \square

1B9.1 Example The leftmost reduction of the fixed-point combinator \mathbf{Y} in 1A10.1 is easily seen to be infinite, so \mathbf{Y} has no β -nf.

1B9.2 Note (Seeking β -normal forms) The leftmost reduction of a term M is completely determined by M , so by 1B9 it gives an algorithm for seeking $M_{*\beta}$: if $M_{*\beta}$ exists the leftmost reduction of M will end at $M_{*\beta}$, and if not, this reduction will be infinite. Of course this algorithm does not decide in finite time whether M has a β -nf; and in fact this cannot be done, as the set of terms with normal forms is not recursive. (See e.g. HS 86 Cor 5.6.2 or Barendregt 1984 Thm. 6.6.5.)

1B10 Lemma (Structure of a β -normal form) Every β -nf N can be expressed uniquely in the form

$$(i) \quad N \equiv \lambda x_1 \dots x_m \cdot y N_1 \dots N_n \quad (m \geq 0, n \geq 0),$$

where N_1, \dots, N_n are β -nf's. And if N is closed then $y \in \{x_1, \dots, x_m\}$.

Proof Easy induction on $|N|$. Note the uniqueness. \square

1B10.1 Note The following special cases of 1B10 are worth mention:

$$\begin{array}{lll} m = n = 0: & N \equiv y & \text{(an atom);} \\ m = 0, n \geq 1: & N \equiv y N_1 \dots N_n & \text{(an application);} \\ m \geq 1: & N \equiv \lambda x_1 \dots x_m \cdot P & \text{(an abstract);} \\ m \geq 1, n = 0: & N \equiv \lambda x_1 \dots x_m \cdot y & \text{(called an *abstracted atom*).} \end{array}$$

1B10.2 Exercise Prove that β -nf is the smallest class of terms satisfying (i) and (ii) below:

- (i) all variables are in β -nf;
- (ii) for all $m, n \geq 0$ with $m + n \geq 1$, and all x_1, \dots, x_m, y ,

$$N_1, \dots, N_n \in \beta\text{-nf} \implies \lambda x_1 \dots x_m \cdot y N_1 \dots N_n \in \beta\text{-nf}.$$

1C η - and $\beta\eta$ -reductions

This section sketches the most basic properties of η - and $\beta\eta$ -reductions. For more details see HS 86 Ch. 7 and Barendregt 1984 §15.1.

1C1 Definition (η -reduction, η -conversion) An η -redex is any term $\lambda x \cdot Mx$ with $x \notin FV(M)$; its re-write rule is

$$\lambda x \cdot Mx \triangleright_{1\eta} M.$$

Its *contractum* is M . The definitions of η -contracts, η -reduces (\triangleright_η), η -converts ($=_\eta$), etc. are like those of the corresponding β -concepts in 1B.

1C2 Lemma All η -reductions are finite; in fact an η -reduction $P \triangleright_{\eta} Q$ must have length $\leq |P|/2$.

Proof Each η -contraction reduces $|P|$ to $|P| - 2$. □

1C3 Definition The η -family $\{P\}_{\eta}$ of a term P is the set of all terms Q such that $P \triangleright_{\eta} Q$.

1C3.1 *Note* By 1C2, $\{P\}_{\eta}$ is finite.

1C4 Church-Rosser Theorem for η If $P =_{\eta} Q$ then there exists T such that

$$P \triangleright_{\eta} T, \quad Q \triangleright_{\eta} T.$$

Proof Straightforward. (Barendregt 1984 Lemma 3.3.7.) □

1C5 Definition ($\beta\eta$ -reduction, $\beta\eta$ -conversion) A $\beta\eta$ -redex is any β - or η -redex. The definitions of $\beta\eta$ -contracts, $\beta\eta$ -reduces ($\triangleright_{\beta\eta}$), $\beta\eta$ -converts ($=_{\beta\eta}$) are like those of the corresponding β -concepts in 1B.

1C5.1 *Lemma* $P \triangleright_{\beta\eta} Q \implies FV(P) \supseteq FV(Q)$.

1C5.2 *Note* A $\beta\eta$ -reduction may have α -steps as well as β and η . The following theorem says that all its η -steps can be postponed to the end of the reduction.

1C6 η -Postponement Theorem If $M \triangleright_{\beta\eta} N$ then there exists a term P such that

$$M \triangleright_{\beta} P \triangleright_{\eta} N.$$

Proof Nederpelt 1973 Thm. 7.28 or Barendregt 1984 Cor. 15.1.6. □

1C7 Commuting Lemma If $M \triangleright_{\beta} P$ and $M \triangleright_{\eta} Q$ (see Fig. 1C7a) then there exists a term T such that

$$P \triangleright_{\eta} T, \quad Q \triangleright_{\beta} T.$$

Proof Barendregt 1984 Lemma 3.3.8. □

1C7.1 *Corollary* If $M \triangleright_{\beta\eta} P$ and $M \triangleright_{\beta} Q$ then there exists a term T such that

$$P \triangleright_{\beta} T, \quad Q \triangleright_{\beta\eta} T.$$

Proof By 1B5, 1C4 and 1C7. □

1C8 Church-Rosser Theorem for $\beta\eta$ (i) If $M \triangleright_{\beta\eta} P$ and $M \triangleright_{\beta\eta} Q$ then there exists T such that

$$P \triangleright_{\beta\eta} T, \quad Q \triangleright_{\beta\eta} T.$$

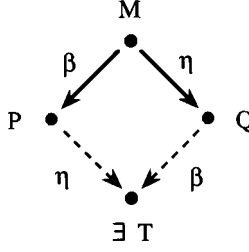


Fig. 1C7a.

(ii) If $P =_{\beta\eta} Q$ then there exists T such that

$$P \triangleright_{\beta\eta} T, \quad Q \triangleright_{\beta\eta} T.$$

Proof (i) From 1B5, 1C4, 1C6, 1C7. (ii) From (i) as in Fig. 1B5b. □

1C9 Definition ($\beta\eta$ - and η -normal forms) A $\beta\eta$ -normal form ($\beta\eta$ -nf) is a term without $\beta\eta$ -redexes. The class of all $\beta\eta$ -nf's is called $\beta\eta$ -nf. We say M has $\beta\eta$ -nf N iff

$$M \triangleright_{\beta\eta} N, \quad N \in \beta\eta\text{-nf}.$$

Similarly we define η -normal form, η -nf, and M has η -nf N .

1C9.1 Notation The $\beta\eta$ -nf and η -nf of a term M are unique modulo \equiv_α by the Church-Rosser theorems for $\beta\eta$ and η ; they will be called

$$M_{\star\beta\eta}, \quad M_{\star\eta}.$$

1C9.2 Lemma (i) An η -reduction of a β -nf cannot create new β -redexes; more precisely

$$M \in \beta\text{-nf and } M \triangleright_\eta N \implies N \in \beta\text{-nf}.$$

(ii) For every M , $M_{\star\beta\eta}$ is the η -nf of $M_{\star\beta}$; i.e. $M_{\star\beta\eta} \equiv (M_{\star\beta})_{\star\eta}$.

Proof (i) It is easy to check all possible cases. (ii) By 1C2, $M_{\star\beta}$ has an η -nf $(M_{\star\beta})_{\star\eta}$, and this is a $\beta\eta$ -nf by (i). □

1C9.3 Corollary If N is a β -nf then all the members of its η -family are β -nf's and exactly one of them is a $\beta\eta$ -nf, namely $N_{\star\eta}$.

1C9.4 Lemma A term has a $\beta\eta$ -nf iff it has a β -nf.

Proof For “only if”, see Curry et al. 1972 §11E Lemma 13.1 or Barendregt 1984 Cor. 15.1.5. For “if”, see 1C9.2. (By the way, do not confuse the present lemma with a claim that a term is in β -nf iff it is in $\beta\eta$ -nf, which is of course false!) □

1C9.5 Note (Seeking $\beta\eta$ -normal-forms) To seek for $M_{*\beta\eta}$, reduce M by its leftmost β -reduction. If this is finite, it must end at $M_{*\beta}$ and then the leftmost η -reduction will reach an η -nf in $\leq |M_{*\beta}|/2$ steps, by 1C2. If the leftmost β -reduction of M is infinite, $M_{*\beta}$ does not exist and hence by 1C9.4 neither does $M_{*\beta\eta}$. Of course this procedure does not decide in finite time whether $M_{*\beta\eta}$ exists; see the comment in 1B9.2.

1D Restricted λ -terms

The following restricted classes of λ -terms will play a role later in the correspondence between type-assignment and propositional logic.

1D1 Definition (λ I-terms) A λ -term P is called a **λ I-term** iff, for each subterm with form $\lambda x \cdot M$ in P , x occurs free in M at least once.

1D1.1 Note The λ I-terms are the terms that were originally studied by Church. They have the property that if a λ I-term has a normal form, so have all its subterms (Church 1941 §7, Thm. 7 XXXII). Church restricted his system to λ I-terms because he regarded terms without normal forms as meaningless and preferred that meaningful terms did not have meaningless subterms. The λ I-terms are discussed in detail in Barendregt 1984 Ch. 9.

The standard example of a non- λ I-term is $\mathbf{K} \equiv \lambda xy \cdot x \equiv \lambda x \cdot (\lambda y \cdot x)$.

1D1.2 Notation Sometimes unrestricted λ -terms are called **λ K-terms**, and the unrestricted λ -calculus the **λ K-calculus**, to contrast with λ I-terms and to emphasise the absence of restriction.

1D2 Definition (BCK λ -terms) A **BCK λ -term** is a λ -term P such that

- (i) for each subterm $\lambda x \cdot M$ of P , x occurs free in M at most once,
- (ii) each free variable of P has just one occurrence free in P .

1D2.1 Examples Of the terms in the list in 1A10.1 the following are BCK λ -terms:

$$\begin{array}{lll} \mathbf{B} \equiv \lambda xyz \cdot x(yz), & \mathbf{B}' \equiv \lambda xyz \cdot y(xz), & \mathbf{C} \equiv \lambda xyz \cdot xzy, \\ \mathbf{I} \equiv \lambda x \cdot x, & \mathbf{K} \equiv \lambda xy \cdot x, & \bar{n} \equiv \lambda xy \cdot x^n y \ (n = 0 \text{ or } 1). \end{array}$$

And the following are not:

$$\begin{array}{ll} \mathbf{S} \equiv \lambda xyz \cdot xz(yz), & \mathbf{W} \equiv \lambda xy \cdot xyy, \\ \mathbf{Y} \equiv \lambda x \cdot (\lambda y \cdot x(yy))(\lambda y \cdot x(yy)), & \bar{n} \equiv \lambda xy \cdot x^n y \ (n \geq 2). \end{array}$$

1D2.2 Lemma The class of all BCK λ -terms is closed under abstraction, i.e. if M is a BCK λ -term then so is $\lambda x \cdot M$ for every variable x .

Proof By 1D2(ii), x occurs free at most once in M . □

1D2.3 Notes (i) In contrast to the above lemma the class of all λ I-terms is only closed under abstractions $\lambda x \cdot M$ such that x occurs free in M .

(ii) The BCK λ -terms are so called because the closed terms in this class correspond to combinations of three combinators called “**B**”, “**C**” and “**K**” in combinatory logic (see 9F for details). They have also sometimes been called *linear λ -terms* but this name is nowadays usually applied to the following class.

1D3 Definition (BCI λ -terms) A *BCI λ -term* or *linear λ -term* is a λ -term P such that

- (i) for each subterm $\lambda x.M$ of P , x occurs free in M exactly once,
- (ii) each free variable of P has just one occurrence free in P .

Clearly every BCI λ -term is a BCK λ -term, but the BCK λ -term **K** is not a BCI λ -term; in fact a term is a BCI λ -term iff it is both a λ I-term and a BCK λ -term. The closed BCI λ -terms correspond to combinations of the combinators called **B**, **C** and **I** in combinatory logic; details are in 9F.

1D4 Lemma *Each of the three classes (λ I-terms, BCK λ -terms and BCI λ -terms) is closed under $\beta\eta$ -reduction, i.e. every term obtained by $\beta\eta$ -reducing a member of the class is also in the class.*

Proof Straightforward. □

1D5 Definition A β -contraction $(\lambda x.M)N \triangleright_{1\beta} [N/x]M$ is said to *cancel* N iff x does not occur free in M ; it is said to *duplicate* N iff x has at least two free occurrences in M .

A β -reduction is *non-duplicating* iff none of its contractions duplicates; it is *non-cancelling* iff none cancels.

1D6 Lemma *Every β -reduction of a λ I-term is non-cancelling; every one of a BCK λ -term is non-duplicating, and every one of a BCI λ -term is both.*

Assigning types to terms

The topic of this book is one of the simplest current type-theories. It was called TA in the Introduction but in fact it comes in two forms, TA_C for combinatory logic and TA_λ for λ -calculus. Since most readers probably know λ -calculus better than combinatory logic, only TA_λ will be described here. (The reader who wishes to see an outline of TA_C can find one in HS 86 Ch.14; most of its properties are parallel to those of TA_λ .)

The present chapter consists of a definition and description of TA_λ . It is close to the treatment in HS 86 Ch. 15 but differs in some technical details.

2A The system TA_λ

2A1 Definition (Types) An infinite sequence of *type-variables* is assumed to be given, distinct from the term-variables. *Types* are linguistic expressions defined thus:

- (i) each type-variable is a type (called an *atom*);
- (ii) if σ and τ are types then $(\sigma \rightarrow \tau)$ is a type (called a *composite type*).

2A1.1 Notation *Type-variables* are denoted by “a”, “b”, “c”, “d”, “e”, “f”, “g”, with or without number-subscripts, and distinct letters denote distinct variables unless otherwise stated.

Arbitrary types are denoted by lower-case Greek letters except “ λ ”.

Parentheses will often (but not always) be omitted from types, and the reader should restore omitted ones in such a way that, for example,

$$\rho \rightarrow \sigma \rightarrow \tau \equiv (\rho \rightarrow (\sigma \rightarrow \tau)).$$

This restoration rule is called *association to the right*.¹

2A1.2 Informal interpretation To interpret types we think of each type-variable as a set and $\sigma \rightarrow \tau$ as a set of functions from σ into τ . The precise nature of this set of functions (all functions, all functions definable in some given system, etc.) will depend on the particular interpretation we may have in mind.

¹ It is the opposite of the rule for terms! The reason originates in the fact that terms follow the common notation convention in which an operator's input is written on the right, but in a type $\sigma \rightarrow \tau$ the type of the input is on the left of that of the output.

2A2 Definition The total number of occurrences of type-variables in a type τ will be called $|\tau|$ or the *length* of τ ; more precisely, define

$$|a| \equiv 1, \quad |\rho \rightarrow \sigma| \equiv |\rho| + |\sigma|.$$

The number of distinct type-variables occurring in τ will be called

$$\|\tau\|$$

and the set of all these variables will be called

$$Vars(\tau).$$

2A2.1 Example If $\tau \equiv (a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$, then

$$|\tau| = 7, \quad \|\tau\| = 3, \quad Vars(\tau) = \{a, b, c\}.$$

The structure of an arbitrary type is analysed in detail in 9D–E. The lemmas there will be used in some later chapters but not in this one.

2A3 Discussion (The Church and Curry approaches) In current use there are two main ways of introducing types into λ -calculus, one attributable to Alonzo Church and the other to Haskell Curry.

The former goes back to a type-system introduced in Church 1940. In it, the definition of “ λ -term” is restricted by giving each term a unique type as part of its structure and saying that an application PQ is only defined when P has a function-type $\sigma \rightarrow \tau$ and Q the appropriate argument-type σ .¹

The effect of Church’s restriction can be seen on $\lambda x \cdot xx$, which is a well-formed term in type-free λ -calculus but represents the abstract concept of self-application, a concept whose meaningfulness may well be questioned. Self-application was involved in most of the paradoxes that were discovered in mathematics in the early 1900’s, and Bertrand Russell devised the first of all type-theories specifically as a language in which these paradoxes could not be expressed. In Church’s typed λ -calculus each variable has a unique type, so if x has a function-type $\sigma \rightarrow \tau$ it cannot also have type σ , and so the application xx cannot be defined as a typed term. Hence also $\lambda x \cdot xx$ cannot be a typed term.

Curry took a different approach. He pointed out that if we wish to ask questions about the meaningfulness of $\lambda x \cdot xx$, then we need a language in which these questions can be expressed. And Church’s type-theory by itself is not adequate for this, because we have just seen that $\lambda x \cdot xx$ is excluded from it. Curry proposed a language which would include all the type-free λ -terms, and a type-theory which would contain rules assigning types to some of these terms but not to others. The term $\lambda x \cdot xx$ would not be given a type by these rules, but would still remain in the system and hence be discussable. (Curry and Feys 1958 §0B, p.5.)

Along with this change Curry proposed another, which is best understood by looking at the identity-combinator $\lambda x \cdot x$ as an example.

In Church’s type-theory there is no term $\lambda x \cdot x$. Instead, for each type σ there is

¹ For a definition of typed λ -term and a few examples see HS 86 §13A; for another version, with motivation and more details, see Barendregt 1992 §3.2.

a variable x^σ with type σ and a term $\lambda x^\sigma \cdot x^\sigma$ with type $\sigma \rightarrow \sigma$. Informally, this term denotes the identity function on whatever set S may be denoted by σ . Call this function I_S ; the only objects it accepts as inputs are members of S , and $I_S(x) = x$ for all $x \in S$. Thus Church's theory has an infinite number of identity functions, one for each set S . This agrees with the view of functions taken by most mathematicians: each function is seen as a set of ordered pairs with a domain and range built into its definition, and the identity functions I_S and I_T on two distinct sets S and T are viewed as different functions.

But this view is not entirely satisfying; an alternative and perhaps more natural view is to see all the separate identity-functions I_S , I_T , etc. as special cases of one intuitive concept, the operation of doing nothing. If we admit that such a concept exists, even though only in an imprecise sense, then a type-theory that tries to make it precise by splitting it into an infinite number of different special cases at the beginning will seem at the very best inefficient.

Curry's aim was a type-theory in which the identity-concept would be expressed by just one term $\lambda x \cdot x$, to which an infinite number of types would be assigned by suitable formal rules. Types would contain variables, and if a term M received a type τ it would also receive all substitution-instances of τ . This kind of theory will be called here a *type-assignment* theory or a *Curry-style* type-theory. (It is the ancestor of polymorphic type-theories.) In contrast, a theory in which each term has a unique built-in type will be called a *typed-term* theory or a *Church-style* theory.¹

TA_λ will be a type-assignment theory.

2A4 Definition A *type-assignment* is any expression

$$M : \tau$$

where M is a λ -term and τ is a type; we call M its *subject* and τ its *predicate*.

(" $M : \tau$ " should be read informally as "assign to M the type τ " or " M has type τ " or " M denotes a member of whatever set τ denotes".)

2A5 Definition A *type-context* Γ is any finite, perhaps empty, set of type-assignments

$$\Gamma = \{x_1 : \rho_1, \dots, x_m : \rho_m\}$$

whose subjects are term-variables and which is *monovalent* or *consistent* in the sense that no variable is the subject of more than one assignment. For any such Γ define

$$\text{Subjects}(\Gamma) = \{x_1, \dots, x_m\}.$$

¹ Curry's and Church's lines of thought were not really as distinct as the above seems to imply. In particular Church did not ignore the possibility that a single identity-concept might be formalizable instead of a multitude of particular identity-functions. Indeed his first systems of λ -calculus in the 1930's were part of an attempt to formalize exactly this single-identity view of functions in a type-free theory, and one of the best available expositions of this view is in the introduction to his book Church 1941. Only after his attempt to do this in an extremely general setting proved inconsistent did Church turn to type-theory and a more restricted approach to functions. Also Curry's type-theories began their development in some of his earliest work and were not simply a response to Church's; see Curry 1934.

2A5.1 Notation The result of removing from Γ the assignment whose subject is x (if Γ has one) is called

$$\Gamma - x.$$

(If $x \notin \text{Subjects}(\Gamma)$ we define $\Gamma - x = \Gamma$.) The result of removing from Γ all assignments $x_i:\rho_i$ with $x_i \notin FV(M)$ (where M is a given term) is called

$$\Gamma \upharpoonright M$$

or “ Γ restricted to M ”. And Γ is called an “ M -context” (for a given M) iff

$$\text{Subjects}(\Gamma) = FV(M).$$

2A5.2 Note A type-context Γ is a set, not a sequence. Hence it does not change when its members are permuted or repeated. To implement TA_λ as a practical system we would have to represent Γ by an expression in some language and include rewrite-rules to permute Γ 's members and make and remove repetitions. Such rules would obscure the main themes of this book so they have been avoided here by simply assuming that contexts are sets.¹

2A6 Definition We say Γ_1 is *consistent with* Γ_2 iff $\Gamma_1 \cup \Gamma_2$ is consistent; and $\Gamma_1, \dots, \Gamma_n$ are *mutually consistent* iff their union is consistent.

2A7 Definition (TA_λ -formulae) For any Γ , M and τ the triple $\langle \Gamma, M, \tau \rangle$ is called a TA_λ -*formula* and is written as

$$\Gamma \mapsto M:\tau$$

(or just $\mapsto M:\tau$ when Γ is empty). We shall call M the *subject* of this formula and τ its *predicate* (despite the fact that in general it contains other subjects and predicates too, namely those of the assignments in Γ).

2A7.1 Notation The following abbreviations will often be used:

$$\begin{array}{ll} x_1:\sigma_1, \dots, x_n:\sigma_n \vdash M:\tau & \text{for } \{x_1:\sigma_1, \dots, x_n:\sigma_n\} \vdash M:\tau, \\ \Gamma, y_1:\sigma_1, \dots, y_n:\sigma_n \vdash M:\tau & \text{for } \Gamma \cup \{y_1:\sigma_1, \dots, y_n:\sigma_n\} \vdash M:\tau. \end{array}$$

2A8 Definition (The system TA_λ) TA_λ has an infinite set of axioms and two deduction-rules (called $(\rightarrow E)$ or \rightarrow -*elimination* and $(\rightarrow I)$ or \rightarrow -*introduction*), as follows.

Axioms of TA_λ : for every term-variable x and every type τ , TA_λ has an axiom

$$x:\tau \mapsto x:\tau.$$

¹ Type-contexts are also called *environments* in the literature. They play a different role from the sets called *bases* in HS 86 Chs.14–15: there a basis was a set of axioms for a theory, whereas here a context will be used as a set of assumptions for a particular deduction in a theory.

Deduction-rules of TA_λ :

$$\begin{array}{ll}
(\rightarrow E) \quad \frac{\Gamma_1 \mapsto P:(\sigma \rightarrow \tau) \quad \Gamma_2 \mapsto Q:\sigma}{\Gamma_1 \cup \Gamma_2 \mapsto (PQ):\tau}, & \text{[if } \Gamma_1 \cup \Gamma_2 \text{ is consistent]} \\
(\rightarrow I) \quad \frac{\Gamma \mapsto P:\tau}{\Gamma - x \mapsto (\lambda x.P):(\sigma \rightarrow \tau)}, & \text{[if } \Gamma \text{ is consistent with } x:\sigma\text{]}
\end{array}$$

A TA_λ -*deduction* Δ is a tree of TA_λ -formulae, those at the tops of branches being axioms and those below being deduced from those immediately above them by a rule. (A detailed definition of such deductions is given in 9C1.) The bottom formula in Δ is called its *conclusion*; if it is

$$\Gamma \mapsto M:\tau$$

we call Δ a *deduction of $\Gamma \mapsto M:\tau$* or a *deduction of $M:\tau$ from Γ* , and say that $\Gamma \mapsto M:\tau$ is *TA_λ -deductible*. In the special case $\Gamma = \emptyset$, Δ may be called a *proof* of the assignment $M:\tau$.

2A8.1 *Note* (Rule $(\rightarrow I)$) The condition in $(\rightarrow I)$ that Γ be consistent with $x:\sigma$ means that either Γ contains $x:\sigma$ or Γ contains no assignment at all whose subject is x . In the first case the rule is said to *discharge* or *cancel* x from Γ . In the second case it is said to *discharge x vacuously*.

In these two cases the rule takes two slightly different forms which may be displayed as follows (using “ Γ_1 ” below to correspond to “ $\Gamma - x$ ” above).

$$\begin{array}{ll}
(\rightarrow I)_{\text{main}} \quad \frac{\Gamma_1, x:\sigma \mapsto P:\tau}{\Gamma_1 \mapsto (\lambda x.P):(\sigma \rightarrow \tau)}, & \text{[if } x \notin \text{Subjects}(\Gamma_1)\text{]} \\
(\rightarrow I)_{\text{vac}} \quad \frac{\Gamma_1 \mapsto P:\tau}{\Gamma_1 \mapsto (\lambda x.P):(\sigma \rightarrow \tau)}, & \text{[if } x \notin \text{Subjects}(\Gamma_1)\text{]}
\end{array}$$

2A8.2 *Example* Let $\mathbf{B} \equiv \lambda xyz.x(yz)$ as in the list in 1A10.1; the following is a deduction of

$$\mapsto \mathbf{B}:(a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b.$$

(In it, “ Γ ” will denote the set $\{x:a \rightarrow b, y:c \rightarrow a, z:c\}$.)

$$\begin{array}{c}
\frac{y:c \rightarrow a \mapsto y:c \rightarrow a \quad z:c \mapsto z:c}{\quad} (\rightarrow E) \\
\frac{x:a \rightarrow b \mapsto x:a \rightarrow b \quad y:c \rightarrow a, z:c \mapsto yz:a}{\quad} (\rightarrow E) \\
\frac{\quad}{\Gamma \mapsto (x(yz)):b} (\rightarrow I)_{\text{main}} \\
\frac{\quad}{\Gamma - z \mapsto (\lambda z.x(yz)):c \rightarrow b} (\rightarrow I)_{\text{main}} \\
\frac{\quad}{\Gamma - z - y \mapsto (\lambda yz.x(yz)): (c \rightarrow a) \rightarrow c \rightarrow b} (\rightarrow I)_{\text{main}} \\
\frac{\quad}{\mapsto (\lambda xyz.x(yz)): (a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b} (\rightarrow I)_{\text{main}}
\end{array}$$

2A8.3 *Example* Let $\mathbf{I} \equiv \lambda x \cdot x$; the following is a deduction of

$$\begin{array}{c} \vdash \mathbf{I} : a \rightarrow a. \\ \frac{x : a \vdash x : a}{\vdash (\lambda x \cdot x) : a \rightarrow a} (\rightarrow I)_{\text{main}} \end{array}$$

2A8.4 *Example* Let $\mathbf{K} \equiv \lambda xy \cdot x$; the following is a deduction of

$$\begin{array}{c} \vdash \mathbf{K} : a \rightarrow b \rightarrow a. \\ \frac{\frac{x : a \vdash x : a}{x : a \vdash (\lambda y \cdot x) : b \rightarrow a} (\rightarrow I)_{\text{vac}}}{\vdash (\lambda xy \cdot x) : a \rightarrow b \rightarrow a} (\rightarrow I)_{\text{main}} \end{array}$$

2A8.5 *Example* The following is a deduction of

$$\begin{array}{c} \vdash \mathbf{II} : a \rightarrow a. \\ \frac{\frac{x : a \rightarrow a \vdash x : a \rightarrow a}{\vdash (\lambda x \cdot x) : (a \rightarrow a) \rightarrow a \rightarrow a} (\rightarrow I) \quad \frac{x : a \vdash x : a}{\vdash (\lambda x \cdot x) : (a \rightarrow a) \rightarrow a} (\rightarrow I)}{\vdash (\lambda x \cdot x)(\lambda x \cdot x) : a \rightarrow a} (\rightarrow E) \end{array}$$

2A8.6 *Remark* (Self-application) The above example gave a type to a term involving self-application, namely \mathbf{II} . This was done by giving a different type to each of the two occurrences of \mathbf{I} , and to do this we had to give two different types to the one variable x ; but there was no inconsistency problem when $(\rightarrow E)$ was applied because the two applications of $(\rightarrow I)$ above $(\rightarrow E)$ removed x from the contexts on the left of “ \vdash ”. Similarly it is possible to give types to several other self-applications in TA_λ , for example \mathbf{KK} and \mathbf{BB} .

This may seem surprising, in view of the claim in 2A3 that the original aim of a type-theory was to avoid self-application. But in fact the “dangerous” self-application to be avoided is not any one simple particular case like \mathbf{II} , but the overall general concept of self-application as represented by the term $\lambda x \cdot xx$. And $\lambda x \cdot xx$ does not receive a type in TA_λ .

To see this, suppose there were a TA_λ -deduction of

$$\vdash (\lambda x \cdot xx) : \tau$$

for some τ . Then its last step would have to be an application of $(\rightarrow I)$ to a deduction of

$$x : \rho \vdash xx : \sigma$$

for some ρ and σ such that $\tau \equiv \rho \rightarrow \sigma$; and the last step in this deduction would have to be an application of $(\rightarrow E)$ to two deductions of

$$x : \sigma_1 \rightarrow \sigma \vdash x : \sigma_1 \rightarrow \sigma, \quad x : \sigma_1 \vdash x : \sigma_1$$

for some σ_1 . But $\sigma_1 \rightarrow \sigma \neq \sigma_1$ so the consistency condition in $(\rightarrow E)$ would be violated by these deductions.

Thus the consistency condition in $(\rightarrow E)$ prevents $\lambda x \cdot xx$ from having a type. This is in fact its main purpose.¹

2A8.7 Exercise* Deduce the following in TA_λ , where $B' \equiv \lambda xyz \cdot y(xz)$, $C \equiv \lambda xyz \cdot xzy$, $S \equiv \lambda xyz \cdot xz(yz)$ and $W \equiv \lambda xy \cdot xyy$ as in 1A10.1.

- (i) $\mapsto B' : (a \rightarrow b) \rightarrow (b \rightarrow c) \rightarrow a \rightarrow c$,
- (ii) $\mapsto C : (a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c$,
- (iii) $\mapsto S : (a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$,
- (iv) $\mapsto W : (a \rightarrow a \rightarrow b) \rightarrow a \rightarrow b$.

2A8.8 Exercise* Deduce the following in TA_λ , where

$$P \equiv (\lambda vxyz \cdot v(y(vxz)))I, \quad Q \equiv \lambda xyz \cdot I(y(Ixz));$$

- (i) $\mapsto P : (a \rightarrow b) \rightarrow (b \rightarrow a \rightarrow b) \rightarrow a \rightarrow a \rightarrow b$;
- (ii) $\mapsto Q : (a \rightarrow b) \rightarrow (b \rightarrow c) \rightarrow a \rightarrow c$.

2A8.9 Note (Comparison with HS 86) The format of TA_λ is what is known as the “*Natural Deduction*” style and was originated by Gerhard Gentzen in his thesis Gentzen 1935. The system called “ TA_λ ” in HS 86 §15B is another variant of the same style; its main differences from the above system TA_λ are as follows.

(i) In HS 86 the discharging of assumptions by rule $(\rightarrow I)$ was shown by enclosing the assumption in brackets at the top of the deduction-tree. But here the set of undischarged assumptions at each stage of the deduction is displayed on the left of the “ \mapsto ” symbol and when rule $(\rightarrow I)$ is used this set is simply reduced. This notation is perhaps more explicit than that in HS 86 and is in common use in recent literature. In both notations deductions have the same tree-structure.

(ii) The version in HS 86 included an α -rule that is not in the present version. This was to ensure that the set of provable formulae would be closed under α -conversion even when the basis of axioms was not. But there are no axioms here in the sense of HS 86 so α -closure will turn out to be provable without adding an α -rule; see 2B6.

2A9 Definition Let Γ be a type-context. Iff there is a TA_λ -deduction of a formula $\Gamma' \mapsto M : \tau$ for some $\Gamma' \subseteq \Gamma$ we shall say

$$\Gamma \vdash_\lambda M : \tau.$$

In the special case $\Gamma = \emptyset$ we shall say *M has type τ in TA_λ* , or *τ is a type of M in TA_λ* , or

$$\vdash_\lambda M : \tau.$$

The phrase “in TA_λ ” may be omitted when no confusion is likely.

2A9.1 Lemma (Weakening) $\Gamma \vdash_\lambda M : \tau, \Gamma^+ \supseteq \Gamma \implies \Gamma^+ \vdash_\lambda M : \tau.$

¹ There is at least one interesting type-theory in which this consistency condition is relaxed, the theory of intersection-types that originated in Coppo and Dezani 1978 and Sallé 1978. In this theory xx receives a type and types play a significantly more complex role than in TA_λ , see for example the comment in Hindley 1992 §1.1.

Proof Trivial from 2A9. □

2A9.2 Warning Do not confuse “ \mapsto ” with “ \vdash ”. The former is part of the language of TA_λ and serves merely to separate two parts of a formula. But “ \vdash_λ ” is part of the meta-language in which TA_λ is described, and asserts the existence of a TA_λ -deduction; it is the traditional deducibility symbol.¹ In particular do not confuse the two statements

- (a) the formula $\Gamma \mapsto M:\tau$ is deducible,
- (b) $\Gamma \vdash_\lambda M:\tau$.

Statement (b) has the weakening property, as we have just seen in 2A9.1. But (a) does not; the rules of TA_λ have been formulated so that if (a) holds then the subjects of Γ coincide exactly with the free variables of M , and we cannot modify a deduction of $\Gamma \mapsto M:\tau$ to make a deduction of $\Gamma^+ \mapsto M:\tau$ if $\Gamma^+ \supsetneq \Gamma$.²

The next two lemmas express the above comments more formally.

2A10 Lemma If $\Gamma \mapsto M:\tau$ is deducible in TA_λ then $\text{Subjects}(\Gamma) = FV(M)$.

Proof By an easy induction on lengths of deductions. □

2A11 Lemma (i) $\Gamma \vdash_\lambda M:\tau$ iff $\text{Subjects}(\Gamma) \supseteq FV(M)$ and there exists a TA_λ -deduction of the formula $\Gamma \upharpoonright M \mapsto M:\tau$.

(ii) $(\exists \Gamma)(\Gamma \vdash_\lambda M:\tau) \iff (\exists \Gamma)\{\Gamma \text{ is an } M\text{-context and } \Gamma \vdash_\lambda M:\tau\}$.

(iii) For closed terms M ,

$$(\exists \Gamma)(\Gamma \vdash_\lambda M:\tau) \iff \vdash_\lambda M:\tau.$$

Proof By 2A9 and 2A10. (The definition of $\Gamma \upharpoonright M$ is in 2A5.1.) □

2A12 Historical Comment Although this book focuses on λ -calculus, type-assignment in fact began in its sister-theory, combinatory logic. The first systems appeared in Curry 1934 and were developed further in Curry and Feys 1958 Chs. 8–10, Seldin 1968 and Curry et al. 1972 Ch. 14, though in his earlier work Curry was aiming at building type-theories with the greatest possible generality and strength and simple type-assignment formed only a small part of each of these. However, when he found his strongest system inconsistent in 1954 he turned to the study of weaker ones and gradually realized that their \rightarrow -fragment formed a very neat core system that was worth studying on its own and stood a good chance of having some practical value.

This system was essentially a combinatory-logic analogue of TA_λ : Curry called it “*modified basic functionality*” though later in HS86 it was called “ TA_C ”. Its basic properties first appeared in Curry and Feys 1958 §§8C and 9A–F, though even then those authors’ main interest was in a slightly stronger system obtained by adding to

¹ Many works use “ \vdash ” where this book uses “ \mapsto ”, and introduce no special notation for deducibility.

² Many other versions of Natural Deduction in the literature do not have this restriction but I believe its use slightly simplifies the proofs of some properties that depend on analysing the structure of a deduction.

TA_C an equality-invariance rule (see Chapter 4 below). The first papers to feature TA_C exclusively were Curry 1969 and Hindley 1969.

As for TA_λ itself, its first appearance was in Curry and Feys 1958 §9F1 under the name “ $F_1(\lambda)^I$ ”, though none of its properties were stated at that date other than a theorem relating it to TA_C . Its properties as an independent system were not described until ten years later in the theses Morris 1968 and Seldin 1968. Seldin’s results on TA_λ appeared in Curry et al. 1972 §14D, but Morris’ thesis was never published. The next study devoted to TA_λ was another unpublished thesis, Ben-Yelles 1979; some of its material will be the subject of Chapter 8 below.

For subsequent work on TA_λ see the references in Chapters 3–9 below.

2B The subject-construction theorem

Deductions in TA_λ have one very important property that is not shared by deductions in many more complex type-theories; the tree-structure of a deduction of $\Gamma \mapsto M:\tau$ follows the tree-structure of M exactly. To make this correspondence precise we really need the detailed definition of construction-tree of a term given in 9A4 and that of a deduction given in 9C1; but the following example gives a very good idea of what it means.

2B1 Example Let $B \equiv \lambda xyz \cdot x(yz)$. A deduction of a type-assignment for B was shown in 2A8.2. If all but the subject is erased from each formula in this deduction the result is the tree shown in Fig. 2B1a. This is the same as the construction-tree of B (with certain details called position-labels omitted, for these see the full definition of construction-tree in 9A4).

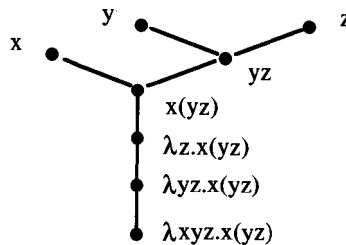


Fig. 2B1a.

The following theorem describes the deductions-to-terms correspondence formally.

2B2 Subject-construction Theorem (Seldin 1968 §3D Thm. 1, Curry et al. 1972 §14D Thm. 1.) *Let Δ be a TA_λ -deduction of a formula $\Gamma \mapsto M:\tau$.*

(i) *If we remove from each formula in Δ everything except its subject, Δ changes to a tree of terms which is exactly the construction-tree for M .*

(ii) If M is an atom, say $M \equiv x$, then $\Gamma = \{x:\tau\}$ and Δ contains only one formula, namely the axiom

$$x:\tau \mapsto x:\tau.$$

(iii) If $M \equiv PQ$ the last step in Δ must be an application of $(\rightarrow E)$ to two formulae with form

$$\Gamma \vdash P \mapsto P:\sigma \rightarrow \tau, \quad \Gamma \vdash Q \mapsto Q:\sigma,$$

for some σ .

(iv) If $M \equiv \lambda x.P$ then τ must have form $\rho \rightarrow \sigma$; further, if $x \in FV(P)$ the last step in Δ must be an application of $(\rightarrow I)_{\text{main}}$ to

$$\Gamma, x:\rho \mapsto P:\sigma,$$

and if $x \notin FV(P)$ the last step in Δ must be an application of $(\rightarrow I)_{\text{vac}}$ to

$$\Gamma \mapsto P:\sigma.$$

Proof Induction on $|M|$. Parts (i)–(iii) follow immediately from the full definition of deduction in 9C1. For (iv): if $M \equiv \lambda x.P$ then by 2A8.1 the last step in Δ must have one of the forms

$$(\rightarrow I)_{\text{main}} \frac{\Gamma, x:\rho \mapsto P:\sigma}{\Gamma \mapsto \lambda x.P:\rho \rightarrow \sigma}, \quad (\rightarrow I)_{\text{vac}} \frac{\Gamma \mapsto P:\sigma}{\Gamma \mapsto \lambda x.P:\rho \rightarrow \sigma}$$

and by 2A10 $(\rightarrow I)_{\text{main}}$ is used when $x \in FV(P)$ and $(\rightarrow I)_{\text{vac}}$ is used otherwise. Hence result. \square

2B2.1 Warning (Deductions are not unique) Given M , let Δ be a deduction of $\Gamma \mapsto M:\tau$. By the subject-construction theorem the structure of M determines both the tree-structure of Δ and the terms at all the nodes in Δ . But this does not mean that the whole of Δ is completely determined by its conclusion, because there is some freedom of choice of the types assigned to terms at non-bottom nodes in Δ . For example, let $\Gamma = \emptyset$ and

$$M \equiv (\lambda xy.y)(\lambda z.z), \quad \tau \equiv a \rightarrow a,$$

and consider the deduction in Fig. 2B2.1a; the type σ in that figure can be arbitrary.

$$\begin{array}{c} \frac{y:a \mapsto y:a}{\mapsto (\lambda xy.y):a \rightarrow a} (\rightarrow I) \\ \frac{\mapsto (\lambda xy.y):a \rightarrow a}{\mapsto (\lambda xy.y):(\sigma \rightarrow \sigma) \rightarrow a \rightarrow a} (\rightarrow I) \quad \frac{z:\sigma \mapsto z:\sigma}{\mapsto (\lambda z.z):\sigma \rightarrow \sigma} (\rightarrow I) \\ \hline \mapsto (\lambda xy.y)(\lambda z.z):a \rightarrow a. (\rightarrow E) \end{array}$$

Fig. 2B2.1a.

However, if M is a normal form or a λ I-term this freedom will disappear and Δ will be completely determined by M , as we shall see in the next lemma and the exercise below it.

2B3 Lemma (Uniqueness of deductions for nf's) (Ben-Yelles 1979 Cor. 3.2.) *Let M be a β -nf and Δ a TA_λ -deduction of $\Gamma \mapsto M:\tau$. Then*

- (i) *every type in Δ has an occurrence in τ or in a type in Γ ,*
- (ii) *Δ is unique, i.e. if Δ' is also a deduction of $\Gamma \mapsto M:\tau$ then $\Delta' \equiv \Delta$.*

Proof Use induction on $|M|$. The cases $M \equiv y$ and $M \equiv \lambda x.P$ are easy. Since M is a β -nf, by 1B10 the only other possible case is

$$M \equiv yP_1 \dots P_n \quad (n \geq 1).$$

In this case any deduction Δ of $\Gamma \mapsto M:\tau$ must contain an axiom

$$y : (\rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow \tau) \mapsto y : (\rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow \tau),$$

as well as n deductions $\Delta_1, \dots, \Delta_n$ giving

$$\Gamma_1 \mapsto P_1:\rho_1, \dots, \Gamma_n \mapsto P_n:\rho_n$$

followed by n applications of $(\rightarrow E)$ to deduce

$$\{y : (\rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow \tau)\} \cup \Gamma_1 \cup \dots \cup \Gamma_n \mapsto (yP_1 \dots P_n) : \tau.$$

And Γ must be

$$\{y : (\rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow \tau)\} \cup \Gamma_1 \cup \dots \cup \Gamma_n.$$

To prove (i): by part (i) of the induction hypothesis every type in a Δ_i occurs in ρ_i or Γ_i and hence in Γ ; also the type of y occurs in Γ . Hence (i) holds.

To prove (ii): the argument above shows that Δ' must use the same rules at the same positions as in Δ . And the type assigned to y in Δ' is determined by Γ and the assumption that type-contexts are consistent; then the types of P_1, \dots, P_n are determined by the type of y . \square

2B3.1 Note (Subformula property) Part (i) of 2B3 corresponds to what is usually called in logic the **subformula property**; this says that in a Natural Deduction system every formula in an irreducible deduction occurs in either the conclusion or an undischarged assumption. (The correspondence between types and propositional logic will be fully described in Chapter 6.)

In contrast the TA_λ -deduction in Fig. 2B2.1a contains a type σ that does not occur in an undischarged assumption or the conclusion.

2B3.2 Exercise* (Uniqueness of deductions for λ I-terms) Show that if Δ is a TA_λ -deduction of $\Gamma \mapsto M:\tau$ and M is a λ I-term, then Δ is unique; i.e. if Δ' is also a TA_λ -deduction of $\Gamma \mapsto M:\tau$ then $\Delta' \equiv \Delta$. (Hint (Thierry Coquand): use 2B3 and some facts from 2C and 2D below, and the leftmost-reduction theorem (1B9), plus some thought on the form that a leftmost reduction must have; see the Answers for details.)

The following three lemmas will be needed in the next section. The first is a special case of the third but is stated separately because it is needed in the proof of the third.

2B4 First Substitution Lemma for Deductions *Let $\Gamma \vdash_{\lambda} M:\tau$ and let $[y/x]\Gamma$ be the result of substituting y for a term-variable x in Γ . If either of the following holds:*

- (i) $y \notin \text{Subjects}(\Gamma)$,
- (ii) y and x receive the same type in Γ ,

then

$$[y/x]\Gamma \vdash_{\lambda} ([y/x]M) : \tau.$$

Proof First, in both cases (i) and (ii) $[y/x]\Gamma$ satisfies the consistency condition for contexts. Next, by 2A11 there is a deduction of

$$\Gamma^- \mapsto M : \tau$$

for some $\Gamma^- \subseteq \Gamma$ with $\text{Subjects}(\Gamma^-) = FV(M)$. Then $[y/x]\Gamma^-$ is consistent. An induction on $|M|$ then shows that

$$[y/x]\Gamma^- \vdash_{\lambda} ([y/x]M) : \tau.^1$$

Then the weakening lemma (2A9.1) gives the result. □

2B5 α -Invariance Lemma *If $\Gamma \vdash_{\lambda} P:\tau$ and $P \equiv_{\alpha} Q$ then $\Gamma \vdash_{\lambda} Q:\tau$.*

Proof [Depends on Section 9C] It is enough to prove the result for one change of bound variable, say the replacement of a component $\lambda x.M$ of P by $\lambda y.[y/x]M$ with $y \notin FV(M)$. If $\lambda x.M \equiv \underline{P}$ the result follows using 2B4. If $\lambda x.M$ is a proper part of \underline{P} , use 9C5 (a replacement lemma). □

2B6 Second Substitution Lemma for Deductions *Let Γ_1 be consistent with Γ_2 and let*

$$\Gamma_1, x:\sigma \vdash_{\lambda} M:\tau, \quad \Gamma_2 \vdash_{\lambda} N:\sigma$$

Then

$$\Gamma_1 \cup \Gamma_2 \vdash_{\lambda} [N/x]M : \tau.$$

Proof Assume $x \in FV(M)$. (If not, the result holds trivially.) By 2B5 we can assume no variable bound in M is free in xN . In this case $[N/x]M$ is simply the result of replacing each free x in M by N with no accompanying changes of bound variables. And by 2A11 we can assume that

$$\begin{aligned} \text{Subjects}(\Gamma_1) \cup \{x\} &= FV(M), \\ \text{Subjects}(\Gamma_2) &= FV(N). \end{aligned}$$

The result is then proved by a straightforward induction on $|M|$. □

¹ By the way, y might be bound in M . To deal with the case that $M \equiv \lambda y.P$, it is necessary to use the induction hypothesis twice and use the fact that $[[z/y]P] = [P]$.

2C Subject reduction and expansion

Besides avoiding logical paradoxes another main purpose of type-theories is to avoid errors of mis-matching in programming. If a term P has a type τ we can think of P as being in some sense “safe”. If P represents a stage in some computation which continues by $\beta\eta$ -reducing P , we would like to know that all later stages in the computation are just as safe as P . The following theorem guarantees this.

2C1 Subject-reduction Theorem (Morris 1968 §4D Thm. 1, Seldin 1968 §3D Thm. 2.)
If $\Gamma \vdash_\lambda P:\tau$ and $P \triangleright_{\beta\eta} Q$ then

$$\Gamma \vdash_\lambda Q:\tau.$$

Proof [Depends on lemmas in 9C] First, by 2A11 there exists a deduction Δ of $\Gamma^- \mapsto P:\tau$ for some $\Gamma^- \subseteq \Gamma$ with *Subjects* $(\Gamma^-) = FV(P)$. By 1C5.1, $FV(P) \supseteq FV(Q)$. Hence by 9C5 it is enough to prove the theorem when P is a β - or η -redex and Q is its contractum.

Case 1: $P \equiv (\lambda x.M)N$, $Q \equiv [N/x]M$. If $x \in FV(M)$ then by the subject-construction theorem (2B2) the lower steps of Δ must have form

$$\frac{\frac{\Gamma_1, x:\sigma \mapsto M:\tau}{\Gamma_1 \mapsto (\lambda x.M):(\sigma \rightarrow \tau)} (\rightarrow I)_{\text{main}} \quad \Gamma_2 \mapsto N:\sigma}{\Gamma_1 \cup \Gamma_2 \mapsto ((\lambda x.M)N):\tau} (\rightarrow E)$$

where $\Gamma_1 \cup \Gamma_2 = \Gamma^-$. Then 2B6 applied to the deductions for M and N gives

$$\Gamma_1 \cup \Gamma_2 \vdash [N/x]M:\tau$$

If $x \notin FV(M)$ the proof is similar.

Case 2: $P \equiv \lambda x.Mx$, $Q \equiv M$, $x \notin FV(M)$. Then $\tau \equiv \rho \rightarrow \sigma$ for some ρ and σ , and by 2B2 the last steps in Δ must have form

$$\frac{\frac{\Gamma^- \mapsto M:\rho \rightarrow \sigma \quad x:\rho \mapsto x:\rho}{\Gamma^-, x:\rho \mapsto Mx:\sigma} (\rightarrow E)}{\Gamma^- \mapsto (\lambda x.Mx):\rho \rightarrow \sigma} (\rightarrow I)_{\text{main}}$$

Thus Δ contains a deduction of $\Gamma^- \mapsto M:\tau$ as required. □

The subject-reduction theorem has a partial analogue for expansion as follows.

2C2 Subject-expansion Theorem If $\Gamma \vdash_\lambda Q:\tau$ and $P \triangleright_\beta Q$ by non-duplicating and non-cancelling contractions, then

$$\Gamma \vdash_\lambda P:\tau.$$

Proof Exercise. This theorem is a special case of Curry et al. 1972 p.315, §14D Thm. 3 (= Seldin 1968 §3D Thm. 3). □

2C2.1 Corollary If P is a closed BCI λ -term and $P \triangleright_\beta Q$ then

$$\Gamma \vdash_\lambda P : \tau \iff \Gamma \vdash_\lambda Q : \tau$$

Proof For “ \Leftarrow ” use 1D6 and 2C2; for “ \Rightarrow ” use 2C1. □

The subject-expansion theorem can be extended to some cancelling contractions under suitable restrictions. (For example see Curry et al. 1972 §14D Thm. 3 or Hindley 1989 Thm. 3.3.) But it cannot be extended to arbitrary contractions, as the following examples show.

2C2.2 Example $P \triangleright_{1\beta} Q$ by a cancelling contraction and Q has a type but P has no type:

$$P \equiv (\lambda uv.v)(\lambda x.xx), \quad Q \equiv \lambda v.v.$$

We have $\vdash_\lambda Q : a \rightarrow a$ by 2A8.3. But no TA_λ -deduction has a conclusion with form $\mapsto P : \tau$. Because such a deduction would have to contain a deduction of $\mapsto (\lambda x.xx) : \sigma$ for some σ and this is impossible by 2A8.6.

2C2.3 Example $P \triangleright_{1\beta} Q$ by a duplicating contraction and Q has a type but P has none:

$$P \equiv (\lambda x.xx)\mathbf{I}, \quad Q \equiv \mathbf{II}.$$

We have $\vdash_\lambda Q : a \rightarrow a$ by 2A8.5. But P has no type because $\lambda x.xx$ has none (by 2A8.6).

2C2.4 Example $P \triangleright_{1\beta} Q$ by a cancellation, P and Q both have types, but Q has more types than P :

$$P \equiv \lambda xyz.(\lambda u.y)(xz), \quad Q \equiv \lambda xyz.y.$$

It is easy to prove that

$$\vdash_\lambda P : (c \rightarrow d) \rightarrow b \rightarrow c \rightarrow b, \quad \vdash_\lambda Q : a \rightarrow b \rightarrow c \rightarrow b;$$

and an application of the principal-type algorithm (3E1) will show that the types possessed by P are exactly the substitution-instances of the one shown above, and similarly for Q . Hence P cannot have the type displayed for Q . (Roughly speaking, the underlying reason is that x has a function position in P and must therefore be assumed to have a function-type $c \rightarrow d$; since x does not occur at all in Q the type of Q has no such limitation.)

2C2.5 Example $P \triangleright_{1\beta} Q$ by a duplication, P and Q both have types, but Q has more types than P :

$$P \equiv (\lambda vxyz.v(y(vxz)))\mathbf{I}, \quad Q \equiv \lambda xyz.\mathbf{I}(y(\mathbf{I}xz)).$$

By 2A8.8 we have

$$\vdash_\lambda P : (a \rightarrow b) \rightarrow (b \rightarrow a \rightarrow b) \rightarrow a \rightarrow a \rightarrow b,$$

$$\vdash_\lambda Q : (a \rightarrow b) \rightarrow (b \rightarrow c) \rightarrow a \rightarrow c;$$

and an application of the principal-type algorithm (3E1) will show that P cannot have the type displayed for Q . (The underlying reason is that the two v 's in P must receive the same type whereas the two l 's in Q are not so limited.)

2C2.6 Example $P \eta$ -contracts to Q , P and Q both have types, but Q has more types than P :

$$P \equiv \lambda xy. xy, \quad Q \equiv \lambda y. y.$$

It is easy to see that

$$\vdash_{\lambda} P : (a \rightarrow b) \rightarrow a \rightarrow b, \quad \vdash_{\lambda} Q : a \rightarrow a,$$

and that a TA_{λ} -deduction of $\mapsto P : a \rightarrow a$ is impossible (because x is in a function position in P).

2C3 Definition (Types(M)) If M is closed, define $\text{Types}(M)$ to be the set of all τ such that $\vdash_{\lambda} M : \tau$.

We shall see in Chapter 3 that if $\text{Types}(M)$ is not empty its members are exactly the substitution-instances of one type, the *principal type* of M ; hence $\text{Types}(M)$ is either empty or infinite.

2C3.1 Lemma Let P be closed. Then

- (i) $P \triangleright_{\beta} Q \implies \text{Types}(P) \subseteq \text{Types}(Q)$,
- (ii) if $P \triangleright_{\beta} Q$ by a non-cancelling and non-duplicating reduction, then

$$\text{Types}(P) = \text{Types}(Q).$$

Proof By 2C1 and 2C2. □

2C3.2 Note (Conversion-invariance) Examples 2C2.2–2C2.6 show that we do not always have

$$M =_{\beta} N \implies \text{Types}(M) = \text{Types}(N).$$

Even worse, we shall see an example in 7A2.1 where $M =_{\beta} N$ but

$$\text{Types}(M) \cap \text{Types}(N) = \emptyset.$$

Thus $\text{Types}(M)$ is very definitely not invariant under conversion.

From a theoretical point of view this seems unsatisfactory. In fact, continuing the Church-versus-Curry discussion from 2A3, it must be admitted that conversion-sensitivity of $\text{Types}(M)$ is the main disadvantage of a Curry-style type system. In a Church system the type of M does not change with conversion (because all terms are typed, including β -redexes, and this fact restricts the β -reduction rule and prevents type-changes). But when we move to a Curry-style system to get the extra expressive power provided by its polymorphism, we do not get it for free, and the price we pay is that $\text{Types}(M)$ can change with conversion.

However, Chapter 4 will describe the effect of adding a new rule to TA_λ to overcome this defect, and it will give theoretical evidence to suggest that perhaps the price is not so high after all.

In practice too the conversion-sensitivity of $Types(M)$ has turned out to be a very small problem. Indeed, if one views an assignment $M:\sigma\rightarrow\tau$ as saying that the application of M to every term with type σ is “safe” in some sense, then the most important practical property of a type-system is the subject-reduction theorem, which says that if M has type $\sigma\rightarrow\tau$ it will not lose this safety-feature during a reduction. If $Types(M)$ happens to increase as M is reduced this is not a drawback but simply means that M is becoming safer. In particular, practical programming languages like ML and its relatives operate very successfully without conversion-invariance.

2D The typable terms

The system TA_λ divides the λ -terms in a natural way into two complementary classes: those which can receive types, such as $\lambda xyz \cdot x(yz)$, and those which cannot, such as $\lambda x \cdot xx$. The former may be regarded as “safe” in the sense that if a term has a type we know there is a way of assigning types to all its components that avoids mis-matches of types. The following is a precise definition of this class.

2D1 Definition A term M is called (TA_λ) -*typable* or *stratified* iff there exist Γ and τ such that

$$\Gamma \vdash_\lambda M : \tau.$$

2D2 Lemma The class of all TA_λ -typable terms is closed under the following operations:

- (i) taking subterms (i.e. all subterms of a typable term are typable);
- (ii) $\beta\eta$ -reduction;
- (iii) non-cancelling and non-duplicating β -expansion;
- (iv) λ -abstraction (i.e. if M is typable so is $\lambda x \cdot M$).

Proof (i) by 2B2. (ii) by 2C1. (iii) by 2C2. (iv) by rule $(\rightarrow I)$. □

2D3 Theorem The class of all TA_λ -typable terms is decidable; that is, there is an algorithm which decides whether a given term is typable in TA_λ .

Proof The principal-type algorithm (3E1) will be a suitable decision-procedure. □

2D4 Remark (Normalization) A property that nearly every type-theory in the literature possesses is the *weak normalization (WN)* property, which says that every typable term can be reduced to a normal form. Many type-theories also have the *strong normalization (SN)* property, which says that all reductions of a typable term

are finite. Both WN and SN can be regarded as safety-features of the type-theory in question: if reductions are viewed as imitating the process of computing values, WN says that a computation can always be continued to a result if we wish and SN says that all computations terminate.

The next theorems state precisely the position for TA_λ .

2D5 Weak Normalization (WN) Theorem (Turing 1942, Curry and Feys 1958, etc.) *Every TA_λ -typable term has both a β -nf and a $\beta\eta$ -nf.*

Proof See 5C1 and 5C1.1 for a proof (from Turing 1942), and 5C1.2 for historical notes. □

2D5.1 Example By 2D5 the fixed-point combinator Y in 1A10.1 is not typable in TA_λ ; because by 1B9.1, Y has no β -nf.

2D6 Strong Normalization (SN) Theorem (Sanchis 1967, Diller 1968, etc.) *If M is a TA_λ -typable term, every $\beta\eta$ -reduction that starts at M is finite.*

Proof There are many proofs in the literature besides those of Sanchis and Diller; for example HS 86 Appendix 2 contains an accessible one for β in Thm. A2.3 and one for $\beta\eta$ in Thm. A2.4. For references to some others see 5C2.2. □

2D6.1 Note Since SN implies WN there is no real need for a separate treatment of WN. But the Turing proof of WN in 5C1 is both simpler and older than any proof of SN. Further, most applications of normalization turn out to be of WN rather than SN. The following are a couple of such applications.

2D7 Theorem *There is a decision-procedure for β -equality of TA_λ -typable terms; i.e. an algorithm which, given any typable terms P and Q , will decide whether $P =_\beta Q$. Similarly for $\beta\eta$ -equality.*

Proof Reduce P and Q to their β -nf's (which exist by WN, and can be found using leftmost reductions, by 1B9) and see whether they differ. □

2D7.1 Note The complexity of the above decision-procedure can be measured in terms of the Grzegorzczuk hierarchy $\mathcal{E}^0, \mathcal{E}^1, \mathcal{E}^2, \dots$ of sets of primitive recursive functions (Grzegorzczuk 1953 p.29): in fact Statman 1979b pp. 73–75 points out that the procedure can be programmed to operate on a Turing machine in \mathcal{E}^4 time but no decision-procedure for β -equality of typable terms can be made to operate in \mathcal{E}^3 time. (The members of \mathcal{E}^3 are known as *elementary functions*.)

2D8 Theorem *Every BCK λ -term (as defined in 1D2) is typable.*

Proof Hindley 1989 Thm. 4.1, depending on WN. □

2D8.1 *Note* The BCK λ -terms are terms without multiple occurrences of variables (except possibly for binding occurrences), so the above theorem connects untypability with multiple occurrences of variables. On the other hand not every term with multiple occurrences is untypable; consider $\mathbf{S} \equiv \lambda xyz.xz(yz)$ in 2A8.7(iii) for example.

The principal-type algorithm

A typable term has in general an infinite set of types in TA_λ . For example if $\mathbf{l} \equiv \lambda x.x$ it is possible to assign to \mathbf{l} every type with form $\sigma \rightarrow \sigma$, by the following deduction:

$$\frac{x:\sigma \mapsto x:\sigma}{\mapsto (\lambda x.x):\sigma \rightarrow \sigma} (\rightarrow\mathbf{I})$$

(like 2A8.3). But all the types in this infinite set are substitution-instances of the one type

$$a \rightarrow a,$$

and it is easy to see that \mathbf{l} has no other types than these. (In fact every deduction for \mathbf{l} must have the simple form shown above, by the subject-construction theorem.) The type $a \rightarrow a$ is called a *principal type* for \mathbf{l} .

The aim of the present chapter is to show that the existence of a principal type is a property of all typable terms, not just \mathbf{l} . In effect the principal type of a term is the most general type it can receive in TA_λ , and the *principal type theorem* will say that every typable term has one. Further, and most important in practice, an algorithm will be described for finding it.

This algorithm will decide whether a given term M is typable and, if the answer is “yes”, will output a principal type for M . Such algorithms are usually called *type-checking* or *principal-type* or *PT* algorithms. The existence of a PT algorithm is what gives TA_λ and its extensions such as ML their practical value, since if the typability of a program is regarded as a safety criterion the programmer will want to be able to decide effectively whether a newly created program satisfies this criterion.

The PT algorithm below will be easy to describe and even easier to apply in practice. But to prove the PT theorem a bald statement of the algorithm will not be enough; we shall need also a proof that the algorithm is correct, i.e. that it does what it claims to do. In the account below a correctness proof will be included with the algorithm in the form of comments to each of its steps, explaining the purpose and effect of each step as the reader meets it.

A little knowledge of substitution, unification and most general unifiers will be needed so introductions to these will be included before the statement of the algorithm.

3A Principal types and their history

3A1 Definition (Type-substitutions) A (*type-*) *substitution* s is any expression

$$[\sigma_1/a_1, \dots, \sigma_n/a_n],$$

where a_1, \dots, a_n are distinct type-variables and $\sigma_1, \dots, \sigma_n$ are any types. For any τ define $s(\tau)$ to be the type obtained by simultaneously substituting σ_1 for a_1, \dots, a_n for a_n throughout τ . In more detail, define

- (i) $s(a_i) \equiv \sigma_i$,
- (ii) $s(b) \equiv b$ if b is an atom $\notin \{a_1, \dots, a_n\}$,
- (iii) $s(\rho \rightarrow \sigma) \equiv s(\rho) \rightarrow s(\sigma)$.

We call $s(\tau)$ an *instance* of τ .

3A1.1 Notation Letters “ r ”, “ s ”, “ t ”, “ u ”, “ v ” will denote type-substitutions. If $s \equiv [\sigma_1/a_1, \dots, \sigma_n/a_n]$ a frequent alternative notation to $s(\tau)$ will be

$$[\sigma_1/a_1, \dots, \sigma_n/a_n]\tau.$$

Recall that the set of all variables occurring in a type τ is called $Vars(\tau)$. The sets of all type-variables occurring in a finite sequence $\langle \tau_1, \dots, \tau_n \rangle$ of types, or in a deduction Δ , are called respectively

$$Vars(\tau_1, \dots, \tau_n), \quad Vars(\Delta).$$

3A2 Definition The action of a substitution s is extended to finite sequences of types, to contexts and to TA_λ -formulae thus:

$$\begin{aligned} s(\langle \tau_1, \dots, \tau_n \rangle) &\equiv \langle s(\tau_1), \dots, s(\tau_n) \rangle, \\ s(\Gamma) &= \{x_1:s(\tau_1), \dots, x_m:s(\tau_m)\} \quad \text{if } \Gamma = \{x_1:\tau_1, \dots, x_m:\tau_m\}, \\ s(\Gamma \mapsto M:\tau) &= s(\Gamma) \mapsto M:s(\tau). \end{aligned}$$

We also extend s to act on deductions Δ by defining $s(\Delta)$ to be the result of applying s to every TA_λ -formula in Δ . We call $s(\Delta)$ an *instance* of Δ . (Similarly for instances of type-sequences, etc.)

3A2.1 Notes (i) The consistency of Γ implies that of $s(\Gamma)$.

(ii) If Δ is a TA_λ -deduction then so is $s(\Delta)$, because the side-conditions in rules $(\rightarrow E)$ and $(\rightarrow I)$ in 2A8 remain true after s has been applied. Hence

$$\Gamma \vdash_\lambda M:\tau \implies s(\Gamma) \vdash_\lambda M:s(\tau),$$

and the set of all types assigned to a term M is closed under substitution.

3A2.2 Warning Two distinct concepts of substitution into deductions have now been mentioned, for term-variables in 2B4 and for type-variables above. Note that $Vars(\Delta)$ is a set of type-variables not term-variables, and when s is applied to Δ the terms in Δ are completely unchanged.

3A3 Definition (Principal types) In TA_λ , a *principal type* or *PT* of a term M is a type τ such that

- (i) $\Gamma \vdash_\lambda M:\tau$ for some Γ ,
- (ii) if $\Gamma' \vdash_\lambda M:\sigma$ for some Γ' and σ then σ is an instance of τ .

3A3.1 Note By 3A2.1(ii) a type τ is a PT of M iff, for all types σ ,

$$(\exists \Gamma')(\Gamma' \vdash_\lambda M:\sigma) \iff \sigma \text{ is an instance of } \tau.$$

Thus a PT of M can be thought of as completely characterizing the set of all types assignable to M .

3A3.2 Notation It will be shown in 3B8.2 that a term's principal type is unique (modulo substitutions of distinct variables for distinct variables), so we shall often say “the principal type of M ” or

$$PT(M).$$

3A4 Definition (Principal pairs) A *principal pair* for a term M is a pair $\langle \Gamma, \tau \rangle$ such that the formula $\Gamma \mapsto M:\tau$ is TA_λ -deducible and every other TA_λ -deducible formula $\Gamma' \mapsto M:\sigma$ is an instance of $\Gamma \mapsto M:\tau$.

3A5 Definition (Principal deductions) A *principal deduction* for a term M is a deduction Δ of a formula $\Gamma \mapsto M:\tau$ such that every other deduction whose conclusion's subject is M is an instance of Δ . To abbreviate “there exists a principal deduction of $\Gamma \mapsto M:\tau$ ” we shall say

$$\Gamma \vdash_p M:\tau.$$

3A5.1 Notes (i) If Δ is a principal deduction of $\Gamma \mapsto M:\tau$ then clearly τ is a principal type of M and $\langle \Gamma, \tau \rangle$ is a principal pair for M . In fact the PT theorem will prove that every typable term has not only a principal type but a principal deduction, and the PT algorithm will be seen to construct principal deductions as well as types. This observation will slightly simplify the algorithm's correctness-proof. Thus a typable term M will be shown to have not only a most general type but a deduction whose every step is most general.

(ii) Although principal types have been studied since 1969, principal deductions were almost entirely neglected until about 1990 when the structure of a principal deduction was characterized by Sachio Hirokawa (Hirokawa 1991a Thms. 1 and 2). This work is beyond the scope of the present book but it is one of the more interesting recent developments in the study of TA_λ and has led to new results on principal types (for example those in Hirokawa 1991b–c, 1993a) and to simplified proofs of some old results.¹

¹ To be precise, Hirokawa 1991a uses a weaker definition of principal deduction than that in 3A5 above: Hirokawa calls a deduction of $\Gamma \mapsto M:\tau$ principal iff the formula $\Gamma \mapsto M:\tau$ is a principal pair in the sense of 3A4. But his characterization theorems can easily be modified to fit the definition in 3A5 by extending the conditions in them to apply to all type-variables in a deduction, not just those in its conclusion and undischarged assumptions.

3A6 Principal Type (PT) Theorem *Every typable term has a principal deduction and a principal type in TA_λ . Further, there is an algorithm that will decide whether a given λ -term M is typable in TA_λ , and if the answer is “yes” will output a principal deduction and principal type for M .*

Proof See the PT algorithm and correctness-proof in 3E. □

3A6.1 Exercise* Let $\mathbf{B} \equiv \lambda xyz \cdot x(yz)$. The type assigned to \mathbf{B} in Example 2A8.2 was

$$(a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b.$$

Using the subject-construction theorem (2B2), show that this type is principal for \mathbf{B} . That is, show that every type assigned to \mathbf{B} in TA_λ must have form

$$(\rho \rightarrow \tau) \rightarrow (\sigma \rightarrow \rho) \rightarrow \sigma \rightarrow \tau,$$

where ρ, σ, τ are arbitrary types.

3A7 Historical Comment The PT problem, that of deciding whether a term is typable and finding its principal type if it is, is one that is crucial to many type-systems, and several different PT algorithms have appeared in the literature over the years. One of the best known is probably *Algorithm W* of Damas and Milner 1982. This was devised originally by Robin Milner for the language ML which includes the present system and a version first appeared in print in Milner 1978.

But the history of PT algorithms goes back a long way before 1978. There seem to have been two mainly independent strands of development, one in λ -calculus with combinatory logic and the other in propositional logic. We shall leave the latter until Chapter 7, but it is almost certainly the older of the two; indeed what amounts to the core of a PT algorithm was formulated in propositional logic as long ago as 1957 and was actually run on a computer at that date, and the informal idea behind such algorithms may go back as far as the 1920's (see 7D3).

The first informal use of a PT algorithm in combinatory logic or λ -calculus was probably due to Haskell Curry in the 1950's. Particular examples of type-computations occur in Curry and Feys 1958 (§9B2, pp.284–293) and show very clearly the equation-solving method on which several later formal algorithms were based. But, as mentioned earlier, Curry was mainly interested in stronger systems at that time and only afterwards began to look at simple type-assignment on its own. When he did, he wrote up his earlier methods as a formal equation-solving PT algorithm for TA_C and added a proof of its correctness; these survive in his notes Curry 1966 and were published in Curry 1969.

A PT algorithm and a correctness-proof were also included in an account of TA_C in Hindley 1969, but these differed from Curry's in that the algorithm made explicit use of the unification algorithm of Robinson 1965 to save itself some work.¹

¹ The preparation of Curry 1969 and Hindley 1969 took place in about 1967 with the authors discussing matters frequently but deliberately keeping their approaches different.

On the λ -calculus side the first PT algorithm was due to James H. Morris and was written out with a correctness proof in his thesis Morris 1968. This was an equation-solving algorithm and like all such, including Curry's, it automatically did all the unification it needed without calling on an external algorithm.¹

For about ten years after 1968 type-assignment remained a very specialist topic, then the development of ML by Robin Milner and his group led to a surge of interest in assignment systems and PT algorithms. Milner's own algorithm was invented as a preliminary to this development, and was applied to \rightarrow -types and pair-types in a λ -calculus with the extra operator *let* that is characteristic of ML. It was written out formally in Milner 1978 and rewritten and extended in Damas and Milner 1982.² Milner's algorithm, like Hindley's, depended on Robinson's unification algorithm to save work.³

Since Milner 1978 many modified or extended PT algorithms have been published, some depending on external unification algorithms and others being self-sufficient. See Tiuryn 1990 for a survey.

3A.7.1 Note The PT algorithm in this chapter will use the method of Hindley 1969 and Milner 1978 rather than equation-solving.

As mentioned above this method depends on a unification algorithm given in advance, but unification algorithms are widely available as packages in practice, so this feature makes a PT algorithm easy to fit into an already given system in a practical implementation.

This method will also turn out to be well suited to deal with the case where the term whose PT is being computed is a combination of other terms P_1, \dots, P_n whose PT's are already known. This situation is common in practice, where a library of terms and their PT's can be built up and used in determining the PT's of new terms, and one of the original motivations for the method was a belief that it would probably use such accumulated information more efficiently than a straight equation-solving algorithm.

However, the account in this chapter will not be concerned with maximizing efficiency, but only with the (usually incompatible) aim of making the PT algorithm's structure and motivation as clear as possible.

3B Type-substitutions

To make the statement of the PT algorithm reasonably concise it will help to introduce some preliminary definitions and lemmas on type-substitutions.

¹ Morris 1968 was independent of Curry's 1966–67 work and must have been prepared at about the same time. Although Morris 1968 was never published there are later accounts of equation-solving λ -calculus algorithms available in the literature; for example there is one with a correctness-proof in Wand 1987.

² Milner's algorithm was discovered independently of earlier work. Milner 1978 did not contain a correctness-proof but there was one in the thesis Damas 1984 to which Damas and Milner 1982 referred.

³ The history of these early algorithms seems to be a tale of repeated re-discovery of very similar ideas. This is not really very surprising, as the ideas are very simple once they are approached in the right way; it is their applications that make them important, not any intrinsic subtlety.

3B1 Notation As defined in 3A1 a substitution is just a finite sequence of instructions $s \equiv [\sigma_1/a_1, \dots, \sigma_n/a_n]$ saying “simultaneously substitute σ_1 for a_1, \dots, σ_n for a_n ”. The following extra notation will be useful.¹

The case $n = 0$ will be allowed and called the *empty substitution*, e . So

$$e(\tau) \equiv \tau.$$

If $n = 1$, s will be called a *single substitution*.

Each part-expression σ_i/a_i of s will be called a *component* of s , and called *trivial* if $\sigma_i \equiv a_i$.

If all trivial components are deleted from s the resulting substitution will be called the *nontrivial kernel* of s .

The set $\{a_1, \dots, a_n\}$ will be called *Dom*(s) or the *variable-domain* of s .

And $Vars(\sigma_1, \dots, \sigma_n)$ will be called *Range*(s) or the *variable-range* of s .

3B2 Definition Substitutions s and t are *extensionally equivalent* ($s =_{\text{ext}} t$) iff $s(\tau) \equiv t(\tau)$ for all τ .

3B2.1 Lemma (i) $b \notin \text{Dom}(s) \implies s(b) \equiv b$.

(ii) $s =_{\text{ext}} t$ iff s and t have the same non-trivial kernel.

3B3 Definition (Restriction, $s \upharpoonright V$) If $s \equiv [\sigma_1/a_1, \dots, \sigma_n/a_n]$ and V is a given set of variables, the *restriction* $s \upharpoonright V$ of s to V is the substitution consisting of the components σ_i/a_i of s such that $a_i \in V$.

3B3.1 Lemma $(s \upharpoonright Vars(\tau))(\tau) \equiv s(\tau)$.

3B4 Definition (Union) If $s \equiv [\sigma_1/a_1, \dots, \sigma_n/a_n]$ and $t \equiv [\tau_1/b_1, \dots, \tau_p/b_p]$ and either $a_1, \dots, a_n, b_1, \dots, b_p$ are all distinct or $a_i \equiv b_j \implies \sigma_i \equiv \tau_j$, define

$$s \cup t \equiv [\sigma_1/a_1, \dots, \sigma_n/a_n, \tau_1/b_1, \dots, \tau_p/b_p]$$

(with repetitions omitted).

3B4.1 Lemma (i) $\text{Dom}(s \cup t) = \text{Dom}(s) \cup \text{Dom}(t)$.

(ii) If $s =_{\text{ext}} s'$ and $t =_{\text{ext}} t'$ and $s \cup t$ is defined, then so is $s' \cup t'$ and

$$s \cup t =_{\text{ext}} s' \cup t'.$$

The next definition will be the *composition* of two substitutions s and t , a simultaneous substitution that will have the same effect as applying t and s in succession. To motivate it, consider the case

$$s \equiv [(c \rightarrow d)/a, (b \rightarrow a)/b], \quad t \equiv [(b \rightarrow a)/b]$$

and let $\tau \equiv a \rightarrow b$. The result of applying first t then s is easily seen to be

$$s(t(\tau)) \equiv (c \rightarrow d) \rightarrow (b \rightarrow a) \rightarrow (c \rightarrow d);$$

¹ There seems to be no standard substitution notation in the literature.

the problem is to find a simultaneous substitution having the same effect. A naive attempt to use $s \cup t$ would fail, because

$$(s \cup t)(\tau) \equiv s(\tau) \equiv (c \rightarrow d) \rightarrow b \rightarrow a.$$

But there does exist a suitable substitution, namely

$$[(c \rightarrow d)/a, (s(b \rightarrow a))/b]$$

This special case is generalized as follows.

3B5 Definition (Composition) If s and t are any substitutions, say

$$s \equiv [\sigma_1/a_1, \dots, \sigma_n/a_n], \quad t \equiv [\tau_1/b_1, \dots, \tau_p/b_p],$$

define

$$s \circ t \equiv [\sigma_{i_1}/a_{i_1}, \dots, \sigma_{i_h}/a_{i_h}, s(\tau_1)/b_1, \dots, s(\tau_p)/b_p]$$

where $\{a_{i_1}, \dots, a_{i_h}\} = \text{Dom}(s) - \text{Dom}(t)$ and $0 \leq h \leq n$.

3B5.1 Lemma (i) $\text{Dom}(s \circ t) = \text{Dom}(s) \cup \text{Dom}(t)$.

(ii) $(s \circ t)(\tau) \equiv s(t(\tau))$.

(iii) $r \circ (s \circ t) =_{\text{ext}} (r \circ s) \circ t$.

(iv) $s =_{\text{ext}} s', t =_{\text{ext}} t' \implies s \circ t =_{\text{ext}} s' \circ t'$.

(v) By (ii), an instance of an instance of τ is an instance of τ .

3B5.2 Exercise* (i) Write out $s \circ t$ in the special case that $s \equiv [a/b]$ and $t \equiv [b/a]$, and verify that $(s \circ t)(\tau) \equiv s(t(\tau))$ in the case $\tau \equiv a \rightarrow b$.

(ii) Show that the action of any s on a given type τ can be expressed as a composition $s_1 \circ \dots \circ s_k$ of single substitutions, in the sense that

$$(s_1 \circ \dots \circ s_k)(\tau) \equiv s(\tau).$$

The next lemma will play an important role in the correctness proof of the PT algorithm: it says that if a composition $s \circ t$ is “extended” to $r \cup (s \circ t)$, the extended substitution can also be expressed as a composition with t (under certain conditions on r to prevent clashes).

3B6 Composition-extension Lemma Let r, s, t be substitutions such that

(i) $\text{Dom}(r) \cap (\text{Dom}(s) \cup \text{Dom}(t)) = \emptyset$,

(ii) $\text{Dom}(r) \cap \text{Range}(t) = \emptyset$.

Then $r \cup (s \circ t)$ and $(r \cup s) \circ t$ are both defined and

$$r \cup (s \circ t) \equiv (r \cup s) \circ t$$

Proof Suppose r, s, t are, respectively,

$$[\rho_1/a_1, \dots, \rho_r/a_r], \quad [\sigma_1/b_1, \dots, \sigma_s/b_s], \quad [\tau_1/c_1, \dots, \tau_t/c_t]$$

with $r, s, t \geq 0$, and suppose

$$(1) \quad \text{Dom}(s) - \text{Dom}(t) = \{b_{i_1}, \dots, b_{i_h}\} \quad (h \geq 0).$$

Then $r \cup (s \circ t)$ is

$$[\rho_1/a_1, \dots, \rho_r/a_r, \sigma_{i_1}/b_{i_1}, \dots, \sigma_{i_h}/b_{i_h}, s(\tau_1)/c_1, \dots, s(\tau_t)/c_t].$$

(This is defined, since $Dom(s \circ t) = Dom(s) \cup Dom(t)$ by 3B5.1(i) and the latter is disjoint from $Dom(r)$ by assumption (i).) On the other hand $(r \cup s) \circ t$ is

$$[\rho_1/a_1, \dots, \rho_r/a_r, \sigma_1/b_1, \dots, \sigma_s/b_s] \circ [\tau_1/c_1, \dots, \tau_t/c_t].$$

(This is defined because $r \cup s$ is defined, by (i).) Now by (i) and (1),

$$Dom(r \cup s) - Dom(t) = \{a_1, \dots, a_r, b_{i_1}, \dots, b_{i_h}\}$$

so by the definition of composition, $(r \cup s) \circ t$ is

$$[\rho_1/a_1, \dots, \rho_r/a_r, \sigma_{i_1}/b_{i_1}, \dots, \sigma_{i_h}/b_{i_h}, (r \cup s)(\tau_1)/c_1, \dots, (r \cup s)(\tau_t)/c_t].$$

And $(r \cup s)(\tau_j) \equiv s(\tau_j)$ for $j = 1, \dots, t$ by assumption (ii). Hence result. \square

To end the section a proof will be given that the PT of a term is unique in a certain sense. The definition of PT carries with it an intuitive feeling that this should be so, and the following definitions will help to make this intuition precise.

3B7 Definition A *variables-for-variables* substitution is a substitution $s \equiv [b_1/a_1, \dots, b_n/a_n]$ where b_1, \dots, b_n are variables (not necessarily distinct).

If b_1, \dots, b_n are distinct, s is called *one-to-one*, and if also $\{a_1, \dots, a_n\} = Vars(\tau)$ for a given type τ , s is called a *renaming (of the variables) in τ* . (Renamings in deductions and finite type-sequences are defined similarly.)

If s is one-to-one its *inverse*, s^{-1} , is defined by

$$s^{-1} \equiv [a_1/b_1, \dots, a_n/b_n].$$

3B7.1 Lemma If s is a renaming in τ then s^{-1} is a renaming in $s(\tau)$ and

$$s^{-1}(s(\tau)) \equiv \tau.$$

3B7.2 Warning A one-to-one substitution may be a renaming in one type but not in another. For example $[b/a]$ is a renaming in $a \rightarrow a$ but not in $a \rightarrow b$.

3B8 Definition We say σ is an *alphabetic variant of τ* , or σ and τ are *identical modulo renaming*, iff $\sigma \equiv s(\tau)$ for some renaming s in τ . (Alphabetic variants of deductions and of finite type-sequences are defined similarly.)

3B8.1 Lemma (i) σ is an alphabetic variant of τ iff σ and τ are instances of each other.

(ii) Part (i) also holds for deductions and finite type-sequences.

Proof For “only if” in (i) use 3B7.1. For “if”, let $Vars(\tau) = \{a_1, \dots, a_n\}$ where a_1, \dots, a_n are distinct, and suppose $\sigma \equiv s(\tau)$ and $\tau \equiv t(\sigma)$, for substitutions s and t with

$$Dom(s) = Vars(\tau), \quad Dom(t) = Vars(\sigma).$$

Then $\mathfrak{t}(\mathfrak{s}(a_i)) \equiv a_i$ for $i = 1, \dots, n$. Hence $\mathfrak{s}(a_i)$ cannot be composite, otherwise $\mathfrak{t}(\mathfrak{s}(a_i))$ would be composite too. Thus \mathfrak{s} has form

$$\mathfrak{s} \equiv [b_1/a_1, \dots, b_n/a_n],$$

and $\text{Vars}(\sigma) = \{b_1, \dots, b_n\}$. Also b_1, \dots, b_n are distinct, because if $b_i \equiv b_j$ then

$$a_i \equiv \mathfrak{t}(\mathfrak{s}(a_i)) \equiv \mathfrak{t}(b_i) \equiv \mathfrak{t}(b_j) \equiv \mathfrak{t}(\mathfrak{s}(a_j)) \equiv a_j.$$

Thus σ is an alphabetic variant of τ . □

3B8.2 Corollary (Uniqueness) *Let τ be a principal type of a term M . Then τ is unique modulo renaming, in the sense that another type σ is a principal type of M iff σ is an alphabetic variant of τ . Principal deductions are unique in a similar sense.*

3B8.3 Lemma *For each finite set of type-variables a_1, \dots, a_m and each type τ there is an alphabetic variant of τ that contains none of a_1, \dots, a_m .*

3C Motivating the PT algorithm

The two main steps in the PT algorithm will be the computation of $PT(\lambda x.P)$ from $PT(P)$ and the computation of $PT(PQ)$ from $PT(P)$ and $PT(Q)$. The first will be no problem but the second will need a little work which in fact will be the core of the PT algorithm; the aim of the present section is to motivate the method used in this core.

3C1 Discussion Suppose we are trying to decide whether an application PQ is typable in TA_λ , and we already know that

$$PT(P) \equiv \rho \rightarrow \sigma, \quad PT(Q) \equiv \tau.$$

(And for simplicity suppose PQ has no free variables.) Then by 3A3.1 the types assignable to P are generated from $\rho \rightarrow \sigma$ by substitution and those assignable to Q are generated from τ . Hence, if we can find substitutions \mathfrak{s}_1 and \mathfrak{s}_2 such that

$$\mathfrak{s}_1(\rho) \equiv \mathfrak{s}_2(\tau),$$

we will be able to deduce a type for PQ by rule $(\rightarrow E)$, thus:

$$\frac{\mapsto P:\mathfrak{s}_1(\rho) \rightarrow \mathfrak{s}_1(\sigma) \quad \mapsto Q:\mathfrak{s}_2(\tau)}{\mapsto PQ:\mathfrak{s}_1(\sigma).} (\rightarrow E)$$

Conversely, by the subject-construction theorem (2B2(iii)), every type deduced for PQ must have been obtained from instances of $\rho \rightarrow \sigma$ and τ by $(\rightarrow E)$ in this way.

Thus the problem of deciding whether PQ is typable reduces to that of finding \mathfrak{s}_1 and \mathfrak{s}_2 such that $\mathfrak{s}_1(\rho) \equiv \mathfrak{s}_2(\tau)$. This suggests the next two definitions.

3C2 Definition (Common instances) (i) If $v \equiv \mathfrak{s}_1(\rho) \equiv \mathfrak{s}_2(\tau)$ we call v a *common instance (c.i.)* of the pair $\langle \rho, \tau \rangle$, and we call $\langle \mathfrak{s}_1, \mathfrak{s}_2 \rangle$ a pair of *converging substitutions* for $\langle \rho, \tau \rangle$.

(ii) Iff $\langle v_1, \dots, v_n \rangle \equiv s_1(\langle \rho_1, \dots, \rho_n \rangle) \equiv s_2(\langle \tau_1, \dots, \tau_n \rangle)$ we call $\langle v_1, \dots, v_n \rangle$ a **common instance** of $\langle \rho_1, \dots, \rho_n \rangle$ and $\langle \tau_1, \dots, \tau_n \rangle$. Common instances of pairs of deductions are defined similarly.

3C2.1 Example A common instance of the pair $\langle a \rightarrow (b \rightarrow c), (a \rightarrow b) \rightarrow a \rangle$ is the type $((\beta \rightarrow \gamma) \rightarrow \delta) \rightarrow (\beta \rightarrow \gamma)$ (where β, γ, δ are any given types), and the corresponding converging substitutions are

$$s_1 \equiv [((\beta \rightarrow \gamma) \rightarrow \delta)/a, \beta/b, \gamma/c], \quad s_2 \equiv [(\beta \rightarrow \gamma)/a, \delta/b].$$

3C2.2 Note Not every pair of types has a common instance. For example the pair $\langle a \rightarrow a, (b \rightarrow b) \rightarrow b \rangle$ has none, because if $s_1(a \rightarrow a) \equiv s_2((b \rightarrow b) \rightarrow b)$ we would have

$$s_1(a) \equiv s_2(b \rightarrow b), \quad s_1(a) \equiv s_2(b),$$

which would imply the impossible identity $s_2(b) \equiv s_2(b) \rightarrow s_2(b)$.

3C3 Definition (M.g.c.i.) (i) A **most general common instance (m.g.c.i.)** of $\langle \rho, \tau \rangle$ is a common instance v_0 such that every other common instance is an instance of v_0 . If v_0 is an m.g.c.i. of $\langle \rho, \tau \rangle$ we shall call any pair $\langle s_1, s_2 \rangle$ such that $s_1(\rho) \equiv s_2(\tau) \equiv v_0$ an **m.g.c.i.-generator** for $\langle \rho, \tau \rangle$.

(ii) M.g.c.i.'s of pairs of type-sequences and pairs of deductions are defined similarly.

3C3.1 Exercise* Show that the pair $\langle a \rightarrow (b \rightarrow c), (a \rightarrow b) \rightarrow a \rangle$ in 3C2.1 has the following as an m.g.c.i.:

$$v_0 \equiv ((b \rightarrow c) \rightarrow d) \rightarrow b \rightarrow c.$$

3C3.2 Lemma (i) *M.g.c.i.'s are unique modulo renaming. That is, if v is an m.g.c.i. of $\langle \rho, \tau \rangle$ the other m.g.c.i.'s of $\langle \rho, \tau \rangle$ are alphabetic variants of v .*

(ii) *If ρ' and τ' are alphabetic variants of ρ and τ respectively, then $\langle \rho', \tau' \rangle$ has the same common instances and the same m.g.c.i.'s as $\langle \rho, \tau \rangle$.*

(iii) *Similarly for m.g.c.i.'s of deductions and finite type-sequences.*

3C4 Discussion It will be shown later that every pair $\langle \rho, \tau \rangle$ with a common instance has an m.g.c.i. Given this fact, the discussion in 3C1 suggests that if we know $PT(P) \equiv \rho \rightarrow \sigma$ and $PT(Q) \equiv \tau$ and we know somehow that $\langle \rho, \tau \rangle$ has a common instance, we can compute $PT(PQ)$ by just constructing the m.g.c.i. of $\langle \rho, \tau \rangle$, say

$$v \equiv s_1(\rho) \equiv s_2(\tau),$$

and then letting $PT(PQ) \equiv s_1(\sigma)$.

And this is indeed true, provided we avoid one small snag. Suppose σ contains some variables b_1, \dots, b_k that do not occur in ρ , and by bad luck the m.g.c.i. v we have constructed also contains some of these variables. Then $s_1(\sigma)$ might contain two occurrences of one variable b_i , one originally in σ and the other introduced into it by s_1 . In this case $s_1(\sigma)$ would not be the most general type assignable to PQ , because we could change v to an alphabetic variant $v' \equiv s'_1(\rho)$ with no variables in

common with σ , and then the corresponding $s'_1(\sigma)$ would be a type of PQ that was not an instance of $s_1(\sigma)$.

As a concrete example, let $P \equiv \lambda xy.x$, $Q \equiv \lambda x.x$; it will be shown in 3E that

$$PT(P) \equiv a \rightarrow (b \rightarrow a), \quad PT(Q) \equiv b \rightarrow b.$$

Thus in this case $\rho \equiv a$, $\sigma \equiv b \rightarrow a$, $\tau \equiv b \rightarrow b$. Clearly an m.g.c.i. of $\langle \rho, \tau \rangle$ is $v \equiv b \rightarrow b$, obtained from ρ by the substitution $s_1 \equiv [(b \rightarrow b)/a]$. And

$$s_1(\sigma) \equiv s_1(b \rightarrow a) \equiv b \rightarrow (b \rightarrow b).$$

It is easy to see that $b \rightarrow (b \rightarrow b)$ is assignable to PQ . But it is not the principal type of PQ . Because if we change s_1 to a new substitution s'_1 by replacing b by a new variable c that does not already occur in σ , we get

$$\vdash_\lambda P : (c \rightarrow c) \rightarrow (b \rightarrow (c \rightarrow c)), \quad \vdash_\lambda Q : c \rightarrow c,$$

and hence PQ has a type $b \rightarrow (c \rightarrow c)$ that is not an instance of $b \rightarrow (b \rightarrow b)$.

To avoid this snag the PT algorithm will be careful to choose an m.g.c.i. v of $\langle \rho, \tau \rangle$ such that

$$(1) \quad \text{Vars}(v) \cap (\text{Vars}(\sigma) - \text{Vars}(\rho)) = \emptyset.$$

Given this precaution, we have now reduced the problem of finding $PT(PQ)$ to that of finding an m.g.c.i. of $\langle \rho, \tau \rangle$. More precisely, we need an algorithm to decide whether $\langle \rho, \tau \rangle$ has a common instance and, if the answer is “yes”, to output a most general one. A suitable algorithm will be given in the next section. It will not be direct, but will apply the *unification algorithm*, which has the advantage that its main properties are so widely known that they will only need to be outlined below.

3D Unification

Most readers have probably met unification before. This section merely summarizes the relevant definitions and basic properties for the reader who has not. The account is based on the classical one in Robinson 1965. (An alternative account is in Aho et al. 1986 §6.7, a thorough survey of major results is in Baader and Siekmann 1994, and a survey of the various applications of unification is in Knight 1989.)

3D1 Definition (Unifiers) (i) If there is a substitution s such that $s(\rho) \equiv s(\tau)$ we say $\langle \rho, \tau \rangle$ is *unifiable*; we call any such s a *unifier* of $\langle \rho, \tau \rangle$ and call $s(\rho)$ a *unification* of $\langle \rho, \tau \rangle$.

(ii) A *unifier* of a pair of sequences $\langle \langle \rho_1, \dots, \rho_n \rangle, \langle \tau_1, \dots, \tau_n \rangle \rangle$, both with the same length, is an s such that

$$s(\langle \rho_1, \dots, \rho_n \rangle) \equiv s(\langle \tau_1, \dots, \tau_n \rangle).$$

3D1.1 Note A unification of $\langle \rho, \tau \rangle$ is just a common instance obtained by making the same substitution in ρ as in τ . But not every common instance is a unification; indeed a pair $\langle \rho, \tau \rangle$ may have a common instance but not be unifiable at all. For example, consider

$$\rho \equiv a \rightarrow (b \rightarrow c), \quad \tau \equiv (a \rightarrow b) \rightarrow a;$$

this pair was shown in 3C2.1 to have a common instance, but no s can exist such that $s(\rho) \equiv s(\tau)$, because the latter would imply the impossible identity

$$s(a) \equiv s(a) \rightarrow s(b).$$

3D1.2 Note The problem of finding unifiers for pairs of type-sequences can be reduced to that for pairs of types as follows. Given two sequences $\langle \rho_1, \dots, \rho_n \rangle$ and $\langle \tau_1, \dots, \tau_n \rangle$, choose a variable b not occurring in any of these types and define

$$\rho^* \equiv \rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow b, \quad \tau^* \equiv \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow b;$$

then the given pair of sequences is unified by a substitution s iff $\langle \rho^*, \tau^* \rangle$ is unified by $s \upharpoonright \text{Vars}(\rho_1, \dots, \rho_n, \tau_1, \dots, \tau_n)$.

3D2 Definition (M.g.u.) (i) A **most general unifier (m.g.u.)** of $\langle \rho, \tau \rangle$ is a unifier \mathfrak{u} such that for every other unifier s of $\langle \rho, \tau \rangle$ we have

$$s(\rho) \equiv s'(\mathfrak{u}(\rho))$$

for some s' . If $v \equiv \mathfrak{u}(\rho)$ for some m.g.u. \mathfrak{u} of $\langle \rho, \tau \rangle$ we shall call v a **most general unification (m.g.u.)** of $\langle \rho, \tau \rangle$.

(ii) M.g.u.'s of pairs of type-sequences or deductions are defined similarly.

3D2.1 Exercise* Prove that the pair $\langle a \rightarrow (b \rightarrow b), (c \rightarrow c) \rightarrow a \rangle$ is unifiable with a most general unifier $\mathfrak{u} \equiv [(b \rightarrow b)/a, b/c]$, and with the corresponding most general unification being

$$(b \rightarrow b) \rightarrow (b \rightarrow b).$$

3D2.2 Lemma A type μ is a most general unification of $\langle \rho, \tau \rangle$ iff μ is a unification of $\langle \rho, \tau \rangle$ and all other unifications of $\langle \rho, \tau \rangle$ are instances of μ .

3D2.3 Lemma If \mathfrak{u} is an m.g.u. of $\langle \rho, \tau \rangle$ and \mathfrak{v} is a renaming of variables in $\mathfrak{u}(\rho)$, then $\mathfrak{v} \circ \mathfrak{u}$ is an m.g.u. of $\langle \rho, \tau \rangle$.

3D2.4 Notation From now on we shall often speak of “the” m.g.u. of $\langle \rho, \tau \rangle$ as if m.g.u.'s were unique. (By 3D2.2, 3B8.1 and 3D2.3 they are unique modulo renaming.)

3D2.5 Lemma (Avoiding variables) (i) Let \mathbb{V} be any finite set of type-variables. If $\langle \rho, \tau \rangle$ has an m.g.u. \mathfrak{u} , then it has an m.g.u. \mathfrak{u}' such that

$$\text{Dom}(\mathfrak{u}') = \text{Vars}(\rho) \cup \text{Vars}(\tau), \quad \text{Range}(\mathfrak{u}') \cap \mathbb{V} = \emptyset.$$

(ii) Similarly for m.g.u.'s of pairs of type-sequences or deductions.

Proof If \mathfrak{u} contains a component σ/a with $a \notin \text{Vars}(\rho) \cup \text{Vars}(\tau)$ we can remove this component from \mathfrak{u} without affecting $\mathfrak{u}(\rho)$ or $\mathfrak{u}(\tau)$. We can also add trivial components to enlarge $\text{Dom}(\mathfrak{u})$ if necessary, and the result can easily be seen to be still an m.g.u. We can then change the range of \mathfrak{u} by a renaming. \square

3D2.6 *Note* (M.g.u.'s and m.g.c.i.'s) The m.g.u. of a pair $\langle \rho, \tau \rangle$ may differ from its m.g.c.i. For instance the m.g.u. of $\langle a \rightarrow (b \rightarrow b), (c \rightarrow c) \rightarrow a \rangle$ was shown in 3D2.1 to be $(b \rightarrow b) \rightarrow (b \rightarrow b)$; but in contrast its m.g.c.i. is easily seen to be $(c \rightarrow c) \rightarrow (b \rightarrow b)$. And as noted in 3D1.1, it is also possible for $\langle \rho, \tau \rangle$ to have an m.g.c.i. but no m.g.u. at all.

However, in the special case that ρ and τ have no common variables every common instance is also a unification, because if

$$\nu \equiv s_1(\rho) \equiv s_2(\tau)$$

and $\text{Dom}(s_1) \subseteq \text{Vars}(\rho)$ and $\text{Dom}(s_2) \subseteq \text{Vars}(\tau)$ then $s_1 \cup s_2$ is defined and

$$\nu \equiv (s_1 \cup s_2)(\rho) \equiv (s_1 \cup s_2)(\tau)$$

This fact gives us the following lemma.

3D3 M.G.U.-M.G.C.I. Lemma (i) *If ρ and τ have no common variables, $\langle \rho, \tau \rangle$ has an m.g.u. iff it has an m.g.c.i., and the two are identical.*

(ii) *For all ρ and τ : if we change τ to an alphabetic variant τ^* with no variables in common with ρ , the unifications of $\langle \rho, \tau^* \rangle$ will be exactly the common instances of $\langle \rho, \tau \rangle$ and the m.g.u. of $\langle \rho, \tau^* \rangle$ will be the m.g.c.i. of $\langle \rho, \tau \rangle$.*

(iii) *Similarly for pairs of type-sequences or deductions.*

Thus the problem of finding m.g.c.i.'s has now been reduced to that of finding m.g.u.'s of pairs $\langle \rho, \tau \rangle$ with no variables in common. But searching for m.g.u.'s of pairs, with or without variables in common, can be done by an algorithm as follows.

3D4 Unification Theorem (J. A. Robinson) (i) *There is an algorithm which decides whether a pair of types $\langle \rho, \tau \rangle$ has a unifier, and, if the answer is "yes", constructs its m.g.u.*

(ii) *If a pair $\langle \rho, \tau \rangle$ has a unifier it has an m.g.u.*

(iii) *Parts (i)–(ii) hold also for pairs of deductions and for pairs of finite type-sequences.*

Proof (i) For Robinson's algorithm see 3D5 below; for a proof of its correctness see Robinson 1965 §5 pp. 32–33.

(ii)–(iii) Like (i). □

3D5 Unification Algorithm (Robinson 1965 §5.) *Input:* any pair $\langle \rho, \tau \rangle$ of types.

Intended output: either a correct statement that $\langle \rho, \tau \rangle$ is not unifiable or an m.g.u. \mathbb{u} of $\langle \rho, \tau \rangle$. [The algorithm will build \mathbb{u} in stages $\mathbb{u}_0, \mathbb{u}_1, \dots$, each \mathbb{u}_k being a composition of \mathbb{u}_{k-1} with a new substitution: at the k -th stage it will test whether $\mathbb{u}_k(\rho) \equiv \mathbb{u}_k(\tau)$, and if the answer is "yes" it will choose $\mathbb{u} \equiv \mathbb{u}_k$ and stop; but if not, it will "extend" \mathbb{u}_k and go to the next stage.]

Step 0. Choose $k = 0$ and $\mathbb{u}_0 \equiv e$ (the empty substitution).

Step $k+1$. Given k and \mathbb{u}_k , construct $\rho_k \equiv \mathbb{u}_k(\rho)$ and $\tau_k \equiv \mathbb{u}_k(\tau)$, and apply the *comparison procedure* below to $\langle \rho_k, \tau_k \rangle$. That procedure will output either a correct statement that $\rho_k \equiv \tau_k$ or a *disagreement pair* $\langle a, \alpha \rangle$ (see below) such that $\alpha \not\equiv a$.

If $\rho_k \equiv \tau_k$, choose $\mathbb{u} \equiv \mathbb{u}_k$.

If $\rho_k \neq \tau_k$ and the output of the comparison procedure is $\langle a, \alpha \rangle$, decide whether $a \in \text{Vars}(\alpha)$.¹

If $a \in \text{Vars}(\alpha)$, state that $\langle \rho, \tau \rangle$ is not unifiable and stop.

If $a \notin \text{Vars}(\alpha)$, then replace k by $k + 1$, choose $\mathfrak{u}_{k+1} \equiv [\alpha/a] \circ \mathfrak{u}_k$, and go to Step $k + 2$.

Comparison Procedure. Given a pair $\langle \mu, \nu \rangle$ of types, write μ and ν as symbol-strings, say

$$\mu \equiv s_1 \dots s_m, \quad \nu \equiv t_1 \dots t_n \quad (m, n \geq 1)$$

where each of $s_1, \dots, s_m, t_1, \dots, t_n$ is an occurrence of a parenthesis, arrow or variable.

If $\mu \equiv \nu$, state that $\mu \equiv \nu$ and stop.

If $\mu \not\equiv \nu$, choose the least $p \leq \text{Min}\{m, n\}$ such that $s_p \neq t_p$; it is not hard to show that one of s_p, t_p must be a variable and the other must be a left parenthesis or a different variable. Further, s_p can be shown to be the leftmost symbol of a unique subtype μ^* of μ . (If s_p is a variable, $\mu^* \equiv s_p$.) Similarly t_p is the leftmost symbol of a unique subtype ν^* of ν . Choose one of μ^*, ν^* that is a variable and call it “ a ”. (If both are variables, choose the one that is first in the sequence given in Definition 2A1.) Then call the remaining member of $\langle \mu^*, \nu^* \rangle$ “ α ”; the pair $\langle a, \alpha \rangle$ is called the **disagreement pair** for $\langle \mu, \nu \rangle$.

3D5.1 Note To prove that p exists in the case that $\mu \neq \nu$ in the comparison procedure we must show that it is not possible to have

$$t_1 \dots t_n \equiv s_1 \dots s_m t_{m+1} \dots t_n$$

with $n > m$. This is left as a (rather dull) exercise for the reader.

3D5.2 Note (History) As mentioned earlier, the above unification algorithm is due to J. A. Robinson, see Robinson 1965 §5; it was the first to be published complete with a correctness-proof. Robinson dates its initial implementation to 1962, but he credits a less smooth algorithm implicit in Prawitz 1960 with influence on its origin and mentions that the history of unification algorithms goes back at least as far as Herbrand 1930.² Another early unification algorithm is implicit in Maslov 1964. For more on the early history see the start of §3 in Baader and Siekmann 1994.

Although Robinson’s algorithm is easy to describe it is not particularly efficient to run, and many better algorithms have been published and implemented since the 1960’s. Useful surveys are in Knight 1989 and Baader and Siekmann 1994. A good discussion of efficiency is in §3.3 of the latter; here it is enough just to mention that there are algorithms in Paterson and Wegman 1978 §4 and Martelli and Montanari 1982 that run in linear time if their inputs $\langle \rho, \tau \rangle$ and outputs \mathfrak{u} are coded in a suitably compact way, but that without some coding even the mere printing-out of $\mathfrak{u}(\rho)$ may take exponential time in the worst cases. The problem of deciding whether a given pair $\langle \rho, \tau \rangle$ can be unified is known to be PTIME-complete (Dwork et al. 1984).

¹ This decision is called the “occurs check”.

² Robinson 1966, Robinson 1979 p. 292.

3D5.3 Corollary (i) *There exists an algorithm which decides whether a pair $\langle \rho, \tau \rangle$ has a common instance, and if the answer is “yes”, outputs an m.g.c.i. and a pair $\langle s_1, s_2 \rangle$ of m.g.c.i.-generators for $\langle \rho, \tau \rangle$.*

(ii) *Every pair of types with a common instance has an m.g.c.i.*

Proof By 3D3(ii) and the unification algorithm. □

3E The PT algorithm

The PT theorem (3A6) stated that there is an algorithm that decides whether a given untyped term M is typable in TA_λ , and outputs a principal deduction and principal type for M if the answer is “yes”. A suitable algorithm will now be described. The proof that it satisfies 3A6 will be given as a series of notes between the algorithm’s steps; these notes will also help to motivate the steps.

3E1 Principal Type (PT) Algorithm (Hindley 1969 §3.) **Input:** any λ -term M , closed or not.

Intended output: either a principal deduction Δ_M for M or a correct statement that M is not typable.

Case I. *If M is a variable, say $M \equiv x$, choose Δ_M to be the one-formula deduction $x:a \mapsto x:a$, where a is any type-variable. [Clearly Δ_M is principal for M .]*

Case II. *If $M \equiv \lambda x.P$ and $x \in FV(P)$, say $FV(P) = \{x, x_1, \dots, x_t\}$, apply the algorithm to P . If P is not typable, neither is M . If P has a principal deduction Δ_P its conclusion must have form*

$$x:\alpha, x_1:\alpha_1, \dots, x_t:\alpha_t \mapsto P:\beta$$

for some types $\alpha, \alpha_1, \dots, \alpha_t, \beta$. Apply rule $(\rightarrow I)_{\text{main}}$ to make a deduction of

$$x_1:\alpha_1, \dots, x_t:\alpha_t \mapsto (\lambda x.P):\alpha \rightarrow \beta.$$

Call this deduction $\Delta_{\lambda x.P}$.

Justification of Case II. We must show that the above $\Delta_{\lambda x.P}$ is principal for $\lambda x.P$. Let Δ be any other deduction of a type for $\lambda x.P$. By the subject-construction theorem (2B2) the last step in Δ must be by rule $(\rightarrow I)_{\text{main}}$, with form

$$\frac{x:\alpha', x_1:\pi_1, \dots, x_t:\pi_t \mapsto P:\beta'}{x_1:\pi_1, \dots, x_t:\pi_t \mapsto (\lambda x.P):\alpha' \rightarrow \beta'}$$

for some types $\alpha', \pi_1, \dots, \pi_t, \beta'$. Now the steps of Δ above the last one form a deduction Δ' that assigns a type to P ; but P has a principal deduction Δ_P , so

$\Delta' \equiv s(\Delta_P)$ for some s ; hence in particular

$$\alpha' \equiv s(\alpha), \quad \pi_1 \equiv s(\alpha_1), \dots, \pi_t \equiv s(\alpha_t), \quad \beta' \equiv s(\beta).$$

Consequently $\Delta \equiv s(\Delta_{\lambda x P})$.

Case III. If $M \equiv \lambda x.P$ and $x \notin FV(P)$, say $FV(P) = \{x_1, \dots, x_t\}$, apply the algorithm to P . If P is not typable then M is not. If P has a principal deduction Δ_P its conclusion must have form

$$x_1:\alpha_1, \dots, x_t:\alpha_t \mapsto P:\beta$$

for some types $\alpha_1, \dots, \alpha_t, \beta$. Choose a new type-variable d not in Δ_P and apply $(\rightarrow I)_{vac}$, vacuously discharging $x:d$, to get a deduction of

$$x_1:\alpha_1, \dots, x_t:\alpha_t \mapsto (\lambda x.P):d \rightarrow \beta.$$

Call this deduction $\Delta_{\lambda x P}$.

Justification of Case III. We must show that the above $\Delta_{\lambda x P}$ is principal. Let Δ be any other deduction of a type for $\lambda x.P$. By 2B2 its last step must be an application of rule $(\rightarrow I)_{vac}$, with form

$$\frac{x_1:\pi_1, \dots, x_t:\pi_t \mapsto P:\beta'}{x_1:\pi_1, \dots, x_t:\pi_t \mapsto (\lambda x.P):\alpha' \rightarrow \beta'}$$

for some $\alpha', \pi_1, \dots, \pi_t, \beta'$. The steps above the last one form a deduction Δ' of a type for P , and Δ_P is principal, so $\Delta' \equiv s(\Delta_P)$ for some s . Choose $s' \equiv s \cup [\alpha'/d]$; then $\Delta \equiv s'(\Delta_{\lambda x P})$.

Case IV. If $M \equiv PQ$, apply the algorithm to P and Q . If P or Q is untypable then so is M . If P and Q are both typable, suppose they have principal deductions Δ_P and Δ_Q . First rename type-variables, if necessary, to ensure that Δ_P and Δ_Q have no common type-variables. Next list the free term-variables in P and those in Q (noting that these lists may overlap); say

$$\begin{aligned} FV(P) &= \{u_1, \dots, u_p, w_1, \dots, w_r\} & (p, r \geq 0), \\ FV(Q) &= \{v_1, \dots, v_q, w_1, \dots, w_r\} & (q \geq 0), \end{aligned}$$

where $u_1, \dots, u_p, v_1, \dots, v_q, w_1, \dots, w_r$ are distinct.

Subcase IVa: $M \equiv PQ$ and $PT(P)$ is composite, say $PT(P) \equiv \rho \rightarrow \sigma$. Then the conclusions of Δ_P and Δ_Q have form, respectively,

- (1) $u_1:\theta_1, \dots, u_p:\theta_p, w_1:\psi_1, \dots, w_r:\psi_r \mapsto P:\rho \rightarrow \sigma,$
- (2) $v_1:\phi_1, \dots, v_q:\phi_q, w_1:\chi_1, \dots, w_r:\chi_r \mapsto Q:\tau.$

Apply the unification algorithm (3D5) to the pair of sequences

- (3) $\langle \psi_1, \dots, \psi_r, \rho \rangle, \quad \langle \chi_1, \dots, \chi_r, \tau \rangle.$

Subcase IVa1: the pair (3) has no unifier. Then PQ is not typable. [See the justification below.]

Subcase IVa2: the pair (3) has a unifier. Then the unification algorithm gives a most general unifier \mathbb{u} ; apply a renaming if necessary (3D2.5) to ensure that

$$(4) \quad \text{Dom}(\mathbb{u}) = \text{Vars}(\psi_1, \dots, \psi_r, \rho, \chi_1, \dots, \chi_r, \tau),$$

$$(5) \quad \text{Range}(\mathbb{u}) \cap \mathbb{V} = \emptyset,$$

where

$$(6) \quad \mathbb{V} = (\text{Vars}(\Delta_P) \cup \text{Vars}(\Delta_Q)) - \text{Dom}(\mathbb{u}).^1$$

Then apply \mathbb{u} to Δ_P and Δ_Q ; this changes their conclusions to, respectively,

$$u_1:\theta_1^*, \dots, u_p:\theta_p^*, w_1:\psi_1^*, \dots, w_r:\psi_r^* \mapsto P:\rho^* \rightarrow \sigma^*,$$

$$v_1:\phi_1^*, \dots, v_q:\phi_q^*, w_1:\chi_1^*, \dots, w_r:\chi_r^* \mapsto Q:\tau^*,$$

where $\theta_1^* \equiv \mathbb{u}(\theta_1)$, etc. And by the definition of \mathbb{u} we have

$$\psi_1^* \equiv \chi_1^*, \dots, \psi_r^* \equiv \chi_r^*, \quad \rho^* \equiv \tau^*.$$

Hence $(\rightarrow E)$ can be applied to the conclusions of $\mathbb{u}(\Delta_P)$ and $\mathbb{u}(\Delta_Q)$. Call the resulting combined deduction Δ_{PQ} ; its conclusion is

$$(7) \quad u_1:\theta_1^*, \dots, u_p:\theta_p^*, v_1:\phi_1^*, \dots, v_q:\phi_q^*, w_1:\chi_1^*, \dots, w_r:\chi_r^* \mapsto PQ:\sigma^*.$$

Justification of Subcase IVa. For IVa1 we must prove that if PQ is typable then $\langle \psi_1, \dots, \psi_r, \rho \rangle$ and $\langle \chi_1, \dots, \chi_r, \tau \rangle$ have a unifier, and for IVa2 we must prove that the above Δ_{PQ} is principal for PQ .

Justification of IVa1. If PQ is typable, there is a deduction Δ whose conclusion has form

$$(8) \quad u_1:\pi_1, \dots, u_p:\pi_p, v_1:\mu_1, \dots, v_q:\mu_q, w_1:\nu_1, \dots, w_r:\nu_r \mapsto PQ:\beta$$

for some types π_1, \dots, π_p , etc. By the subject-construction theorem (2B2), Δ must have been built by applying rule $(\rightarrow E)$ to two deductions Δ_1 and Δ_2 whose conclusions are

$$(9) \quad u_1:\pi_1, \dots, u_p:\pi_p, w_1:\nu_1, \dots, w_r:\nu_r \mapsto P:\alpha \rightarrow \beta,$$

$$(10) \quad v_1:\mu_1, \dots, v_q:\mu_q, w_1:\nu_1, \dots, w_r:\nu_r \mapsto Q:\alpha$$

for some type α . But Δ_P and Δ_Q are principal deductions for P and Q , so

$$\Delta_1 \equiv \mathfrak{r}_1(\Delta_P), \quad \Delta_2 \equiv \mathfrak{r}_2(\Delta_Q)$$

for some substitutions \mathfrak{r}_1 and \mathfrak{r}_2 such that

$$\text{Dom}(\mathfrak{r}_1) = \text{Vars}(\Delta_P), \quad \text{Dom}(\mathfrak{r}_2) = \text{Vars}(\Delta_Q).$$

¹ Roughly speaking, (5) and (6) say that when \mathbb{u} is applied to Δ_P and Δ_Q any new variables it introduces will differ from all those already in Δ_P and Δ_Q , and hence no unnecessary identifications of variables will be made. (Cf. the motivation in 3C4.) By the way, if the aim of the algorithm had been to construct merely a principal type and not a principal deduction, (5) could have been weakened by re-defining \mathbb{V} to consist of just the variables (if any) in σ that do not occur in the types in (3). (Cf. (1) in 3C4.)

Let $r \equiv r_1 \cup r_2$. (This is a well-defined substitution since Δ_P and Δ_Q were chosen such that $\text{Vars}(\Delta_P) \cap \text{Vars}(\Delta_Q) = \emptyset$.) Then

$$(11) \quad \Delta_1 \equiv \tau(\Delta_P), \quad \Delta_2 \equiv \tau(\Delta_Q),$$

$$(12) \quad \text{Dom}(\tau) = \text{Vars}(\Delta_P) \cup \text{Vars}(\Delta_Q).$$

Now look at v_1, \dots, v_r and α in (9). By comparing (9) and (1), since $\Delta_1 \equiv \tau(\Delta_P)$ we have

$$(13) \quad \langle v_1, \dots, v_r, \alpha \rangle \equiv \langle \tau(\psi_1), \dots, \tau(\psi_r), \tau(\rho) \rangle.$$

And by comparing (10) and (12), since $\Delta_2 \equiv \tau(\Delta_Q)$,

$$(14) \quad \langle v_1, \dots, v_r, \alpha \rangle \equiv \langle \tau(\chi_1), \dots, \tau(\chi_r), \tau(\tau) \rangle.$$

Thus $\langle \psi_1, \dots, \psi_r, \rho \rangle$ and $\langle \chi_1, \dots, \chi_r, \tau \rangle$ have a unifier, namely τ .

Justification of IVa2. We must prove that Δ_{PQ} is principal for PQ , i.e. that an arbitrary deduction Δ of (8) is an instance of Δ_{PQ} . For any such Δ , define Δ_1, Δ_2 and τ as above. By (6) and (12) we can express τ as $\tau \equiv \tau' \cup \tau''$, where

$$(15) \quad \tau' \equiv \tau \upharpoonright \mathbb{V}, \quad \tau'' \equiv \tau \upharpoonright \text{Dom}(\mathfrak{u}).$$

By (4), (13) and (14), τ'' is a unifier of

$$\langle \psi_1, \dots, \psi_r, \rho \rangle, \quad \langle \chi_1, \dots, \chi_r, \tau \rangle.$$

But \mathfrak{u} is an m.g.u. of this pair of sequences, so by the definition of m.g.u. (3D2) there exists \mathfrak{s} such that $\tau''(a) \equiv \mathfrak{s}(\mathfrak{u}(a))$ for all $a \in \text{Dom}(\mathfrak{u})$. We can clearly also assume

$$(16) \quad \text{Dom}(\mathfrak{s}) \subseteq \text{Range}(\mathfrak{u}).$$

It follows that

$$(17) \quad \tau'' =_{\text{ext}} \mathfrak{s} \circ \mathfrak{u}.$$

Since $\tau \equiv \tau' \cup \tau''$, we have by 3B4.1(ii),

$$(18) \quad \tau =_{\text{ext}} \tau' \cup (\mathfrak{s} \circ \mathfrak{u}).$$

Now $\tau', \mathfrak{s}, \mathfrak{u}$ satisfy conditions (i) and (ii) in the composition-extension lemma (3B6). In fact 3B6(i) holds because by (15) and (16) we have

$$\begin{aligned} \text{Dom}(\tau') \cap (\text{Dom}(\mathfrak{s}) \cup \text{Dom}(\mathfrak{u})) &\subseteq \mathbb{V} \cap (\text{Range}(\mathfrak{u}) \cup \text{Dom}(\mathfrak{u})) \\ &= \emptyset \end{aligned} \quad \text{by (5), (6),}$$

and 3B6(ii) holds by (5) and (15). Hence by (18) and 3B6,

$$\tau =_{\text{ext}} (\tau' \cup \mathfrak{s}) \circ \mathfrak{u}.$$

Therefore by (11),

$$\Delta_1 \equiv (\tau' \cup \mathfrak{s})(\mathfrak{u}(\Delta_P)), \quad \Delta_2 \equiv (\tau' \cup \mathfrak{s})(\mathfrak{u}(\Delta_Q)).$$

But Δ is a combination of Δ_1 and Δ_2 by rule ($\rightarrow E$) and Δ_{PQ} is a similar combination of $\mathfrak{u}(\Delta_P)$ and $\mathfrak{u}(\Delta_Q)$. Hence

$$\Delta \equiv (\tau' \cup \mathfrak{s})(\Delta_{PQ}).$$

Subcase IVb: $M \equiv PQ$ and $PT(P)$ is atomic, say $PT(P) \equiv b$. Then the conclusions of Δ_P and Δ_Q have form, respectively,

$$(19) \quad u_1:\theta_1, \dots, u_p:\theta_p, w_1:\psi_1, \dots, w_r:\psi_r \mapsto P:b,$$

$$(20) \quad v_1:\phi_1, \dots, v_q:\phi_q, w_1:\chi_1, \dots, w_r:\chi_r \mapsto Q:\tau.$$

Choose any variable $c \notin \text{Vars}(\Delta_P) \cup \text{Vars}(\Delta_Q)$ and apply the unification algorithm to the pair of sequences

$$(21) \quad \langle \psi_1, \dots, \psi_r, b \rangle, \quad \langle \chi_1, \dots, \chi_r, \tau \rightarrow c \rangle.$$

Subsubcase IVb1: the pair (21) has no unifier. Then PQ is not typable. [See the justification below.]

Subsubcase IVb2: the pair (21) has a unifier. Then the unification algorithm gives an m.g.u. \mathbb{u} ; apply 3D2.5 to ensure that

$$(22) \quad \text{Dom}(\mathbb{u}) = \text{Vars}(\psi_1, \dots, \psi_r, b, \chi_1, \dots, \chi_r, \tau \rightarrow c),$$

$$(23) \quad \text{Range}(\mathbb{u}) \cap \mathbb{V} = \emptyset,$$

where \mathbb{V} is the same as in (6). Then apply \mathbb{u} to Δ_P and Δ_Q . By the definition of \mathbb{u} ,

$$\mathbb{u}(b) \equiv \mathbb{u}(\tau \rightarrow c) \equiv \mathbb{u}(\tau) \rightarrow \mathbb{u}(c),$$

and thus the conclusions of $\mathbb{u}(\Delta_P)$ and $\mathbb{u}(\Delta_Q)$ are

$$u_1:\theta_1^*, \dots, u_p:\theta_p^*, w_1:\psi_1^*, \dots, w_r:\psi_r^* \mapsto P:\tau^* \rightarrow c^*,$$

$$v_1:\phi_1^*, \dots, v_q:\phi_q^*, w_1:\chi_1^*, \dots, w_r:\chi_r^* \mapsto Q:\tau^*,$$

where $*$ denotes application of the substitution \mathbb{u} , and

$$\psi_1^* \equiv \chi_1^*, \dots, \psi_r^* \equiv \chi_r^*.$$

Choose Δ_{PQ} to be the deduction obtained by applying rule $(\rightarrow E)$ to $\mathbb{u}(\Delta_P)$ and $\mathbb{u}(\Delta_Q)$; its conclusion is

$$u_1:\theta_1^*, \dots, u_p:\theta_p^*, v_1:\phi_1^*, \dots, v_q:\phi_q^*, w_1:\chi_1^*, \dots, w_r:\chi_r^* \mapsto PQ:c^*.$$

Justification of Subcase IVb. For IVb1 we must prove that if PQ is typable then $\langle \psi_1, \dots, \psi_r, b \rangle$ and $\langle \chi_1, \dots, \chi_r, \tau \rightarrow c \rangle$ have a unifier, and for IVb2 we must prove that Δ_{PQ} is principal for PQ .

Justification of IVb1. If PQ is typable there is a Δ whose conclusion has the form (8). By 2B2, Δ must have been built by applying $(\rightarrow E)$ to two deductions Δ_1 and Δ_2 with conclusions (9) and (10). But P and Q have principal deductions Δ_P and Δ_Q , so

$$(24) \quad \Delta_1 \equiv r_1(\Delta_P), \quad \Delta_2 \equiv r_2(\Delta_Q)$$

for some substitutions \mathfrak{r}_1 and \mathfrak{r}_2 such that $Dom(\mathfrak{r}_1) = Vars(\Delta_P)$ and $Dom(\mathfrak{r}_2) = Vars(\Delta_Q)$. Note in particular that

$$\alpha \rightarrow \beta \equiv \mathfrak{r}_1(b), \quad \alpha \equiv \mathfrak{r}_2(\tau).$$

Define

$$\mathfrak{r} \equiv \mathfrak{r}_1 \cup \mathfrak{r}_2 \cup [\beta/c].$$

This is well-defined because $c \notin Dom(\mathfrak{r}_1) \cup Dom(\mathfrak{r}_2)$. And $\mathfrak{r}(c) \equiv \beta$, and by (24),

$$\begin{aligned} \mathfrak{r}(\Delta_P) &\equiv \Delta_1, & \mathfrak{r}(\Delta_Q) &\equiv \Delta_2, \\ \mathfrak{r}(b) &\equiv \alpha \rightarrow \beta \equiv \mathfrak{r}(\tau \rightarrow c). \end{aligned}$$

Also

$$Dom(\mathfrak{r}) = Vars(\Delta_P) \cup Vars(\Delta_Q) \cup \{c\}.$$

Now look at v_1, \dots, v_r and $\alpha \rightarrow \beta$ in (9). By the above, we have

$$(25) \quad \langle v_1, \dots, v_r, \alpha \rightarrow \beta \rangle \equiv \langle \mathfrak{r}(\psi_1), \dots, \mathfrak{r}(\psi_r), \mathfrak{r}(b) \rangle,$$

$$(26) \quad \langle v_1, \dots, v_r, \alpha \rightarrow \beta \rangle \equiv \langle \mathfrak{r}(\chi_1), \dots, \mathfrak{r}(\chi_r), \mathfrak{r}(\tau \rightarrow c) \rangle.$$

Thus $\langle \psi_1, \dots, \psi_r, b \rangle$ and $\langle \chi_1, \dots, \chi_r, \tau \rightarrow c \rangle$ have a unifier, namely \mathfrak{r} .

Justification of IVb2. To prove Δ_{PQ} principal we must prove that an arbitrary deduction Δ of (8) is an instance of Δ_{PQ} . For any such Δ , define Δ_1 , Δ_2 and \mathfrak{r} as above. We can express \mathfrak{r} as $\mathfrak{r}' \cup \mathfrak{r}''$, where

$$\mathfrak{r}' \equiv \mathfrak{r} \upharpoonright \mathbb{V}, \quad \mathfrak{r}'' = \mathfrak{r} \upharpoonright Dom(\mathfrak{u}),$$

where \mathbb{V} and $Dom(\mathfrak{u})$ were defined in (6) and (22). By (25) and (26), \mathfrak{r}'' unifies

$$\langle \psi_1, \dots, \psi_r, b \rangle, \quad \langle \chi_1, \dots, \chi_r, \tau \rightarrow c \rangle.$$

Hence, just as in the justification of IVa2, $\mathfrak{r}'' =_{\text{ext}} \mathfrak{s} \circ \mathfrak{u}$ for some \mathfrak{s} such that

$$(27) \quad Dom(\mathfrak{s}) \subseteq Range(\mathfrak{u}).$$

Thus $\mathfrak{r} =_{\text{ext}} \mathfrak{r}' \cup (\mathfrak{s} \circ \mathfrak{u})$. Using (23) and (27) it is easy to see that \mathfrak{r}' , \mathfrak{s} , \mathfrak{u} satisfy the conditions in the composition-extension lemma (3B6), so by that lemma,

$$\mathfrak{r} =_{\text{ext}} (\mathfrak{r}' \cup \mathfrak{s}) \circ \mathfrak{u}.$$

Hence, similarly to the justification of IVa2, $\Delta \equiv (\mathfrak{r}' \cup \mathfrak{s})(\Delta_{PQ})$. □

3E1.1 Notes (i) The PT algorithm could have been shortened by combining Subcases IVa and IVb as was done in Hindley 1969: in IVa the unification algorithm could have been applied to

$$\langle \psi_1, \dots, \psi_r, \rho \rightarrow \sigma \rangle, \quad \langle \chi_1, \dots, \chi_r, \tau \rightarrow c \rangle$$

where c is a new variable, and this modification would have made IVa exactly parallel to IVb and the latter could have been omitted. But this chapter's aim is to explain clearly what is involved in computing a PT and not just to give the slickest presentation, and IVa seems easier to understand when not combined with IVb.

(ii) At the start of Case IV the algorithm could have been made more direct by omitting the renaming of the type-variables in Δ_P and Δ_Q and using common instances instead of unifiers in Subcases IVa and IVb. For example in IVa, instead of seeking an m.g.u. θ of the sequences

$$\langle \psi_1, \dots, \psi_r, \rho \rangle, \quad \langle \chi_1, \dots, \chi_r, \tau \rangle,$$

we could have sought an m.g.c.i. The justification of the algorithm would then have rested on 3D5.3, the analogue of the unification theorem for common instances.

3E2 Exercise Show that the closed terms in Table 3E2a have the principal types shown there. (The answers to (1)–(7) are displayed in 2A8.2–4 and in the answer to Exercise 2A8.7; these deductions can easily be shown to be principal. The others can be checked by applying the PT algorithm.)

By the way, the terms (10) and (11) in Table 3E2a are not in β -normal form; we shall see in 8B4 and 8B7 that there are no closed β -nfs with the same PT's as these terms.

An extended table of terms and their principal types is given at the end of the book for ease of reference.

Table 3E2a.

	Term	Principal type
(1)	B $\equiv \lambda xyz \cdot x(yz)$	$(a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b$
(2)	B' $\equiv \lambda xyz \cdot y(xz)$	$(a \rightarrow b) \rightarrow (b \rightarrow c) \rightarrow a \rightarrow c$
(3)	C $\equiv \lambda xyz \cdot xzy$	$(a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c$
(4)	I $\equiv \lambda x \cdot x$	$a \rightarrow a$
(5)	K $\equiv \lambda xy \cdot x$	$a \rightarrow b \rightarrow a$
(6)	S $\equiv \lambda xyz \cdot xz(yz)$	$(a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$
(7)	W $\equiv \lambda xy \cdot xyy$	$(a \rightarrow a \rightarrow b) \rightarrow a \rightarrow b$
(8)	$\bar{0}$ $\equiv \lambda xy \cdot y$	$a \rightarrow b \rightarrow b$
(9)	$\bar{1}$ $\equiv \lambda xy \cdot xy$	$(a \rightarrow b) \rightarrow a \rightarrow b$
(10)	$(\lambda xyz \cdot \mathbf{K}(xy)(xz))\mathbf{I}$	$a \rightarrow a \rightarrow a$
(11)	$\lambda xy \cdot (\lambda z \cdot x)(yx)$	$a \rightarrow (a \rightarrow b) \rightarrow a$
(12)	$\lambda xyz \cdot xy(xz)$	[untypable]
(13)	$\lambda x \cdot xx$	[untypable]

3E3 Theorem The relation $\vdash_\lambda M : \tau$ is decidable, i.e. there is an algorithm which accepts any M and τ and decides whether or not $\vdash_\lambda M : \tau$.

Proof Apply the PT algorithm to M . If M is not closed or not typable we cannot have $\vdash_\lambda M : \tau$. If M is closed and typable, decide whether τ is an instance of $PT(M)$. \square

3E4 Further Reading For an alternative introduction to PT's and type-checking see Aho et al. 1986 Ch. 6. The early literature on PT algorithms was mentioned in 3A7 above. The more recent literature on their uses and properties is fairly extensive and perhaps the best place to start would be the survey Tiurnyn 1990 and the papers in its bibliography, as well as Giannini et al. 1993.

Accounts of PT algorithms that include correctness-proofs are relatively rare, however; besides the proofs in Curry 1969 and Hindley 1969 §3 (and the unpublished ones in Morris 1968 and Damas 1984) there is one in Wand 1987.

3E4.1 Note (Efficiency) Results on efficiency and complexity of PT algorithms, mainly focusing on ML, are contained in Kanellakis and Mitchell 1989, Mairson 1990, Kfoury et al. 1990 and Kanellakis et al. 1991. In summary, there is a PT algorithm which can decide in linear time whether a λ -term containing no constants is typable, and if it is, will print out its PT in linear time coded in a suitably compressed form (as a directed acyclic graph or DAG). Also the typability problem for pure λ -terms is PTIME-complete just like the unification problem (3D5.2). (Proofs of this are in Tyszkiewicz 1988 and Mitchell 1996 Ch. 11.)

But when ML's *let* operator is added to the term-language the corresponding typability problem becomes PSPACE-hard (Kanellakis and Mitchell 1989) and DEXPTIME-complete (Mairson 1990, Kfoury et al. 1990).

Type assignment with equality

Let M be a closed term and $Types(M)$ be the set of all types assignable to M in TA_λ . As we saw in Chapter 2 this set might not stay invariant during a β -conversion of M . In fact Examples 2C2.2–6 and Note 2C3.2 showed that this set may increase during a reduction of M and sometimes M can even be converted to a term with no types at all in common with M .

As noted in 2C3.2 this lack of equality-invariance has not hindered TA_λ from a practical point of view, nor its descendant ML, as the main need in both systems has been simply for the subject-reduction theorem. However, on the theoretical side, equality-invariance of $Types(M)$ seems a desirable property from the viewpoint of any of the standard λ -calculus semantics (except possibly an operational semantics): if we believe terms represent functions of some sort and equal terms represent the same function, then equal terms should have the same types.

So this chapter will describe a system obtained by adding an equality-invariance rule to TA_λ . More precisely, two systems will be described in parallel, one for $=_\beta$ and the other for $=_{\beta\eta}$; both will have similar properties.

4A The equality rule

4A1 Definition (The systems $TA_{\lambda+\beta}$, $TA_{\lambda+\beta\eta}$) We define $TA_{\lambda+\beta}$ by adding to rules $(\rightarrow E)$ and $(\rightarrow I)$ in Definition 2A8 the new rule

$$(Eq_\beta) \frac{\Gamma \mapsto M:\tau \quad M =_\beta N}{\Gamma \mapsto N:\tau.}$$

If a formula $\Gamma' \mapsto M:\tau$ is provable in $TA_{\lambda+\beta}$ for some $\Gamma' \subseteq \Gamma$ we say

$$\Gamma \vdash_\beta M:\tau.$$

And iff there exist Γ and τ such that $\Gamma \vdash_\beta M:\tau$ we call M *$TA_{\lambda+\beta}$ -typable*.

We define $TA_{\lambda+\beta\eta}$, $\vdash_{\beta\eta}$ and *$TA_{\lambda+\beta\eta}$ -typability* similarly but with “ $=_{\beta\eta}$ ” instead of “ $=_\beta$ ”, calling the corresponding new rule “ $(Eq_{\beta\eta})$ ”.¹

4A1.1 Notation When confusion is unlikely (Eq_β) and $(Eq_{\beta\eta})$ may both be called just “ (Eq) ”.

¹ Strictly speaking, to make the (Eq) -rules meaningful we must also add the axioms and rules for $=_\beta$ and $=_{\beta\eta}$ to the definitions of $TA_{\lambda+\beta}$ and $TA_{\lambda+\beta\eta}$ in some way; the details of this are left to the reader.

The name “ $\text{TA}_{\lambda+\beta[\eta]}$ ” will be used to denote $\text{TA}_{\lambda+\beta}$ and $\text{TA}_{\lambda+\beta\eta}$ simultaneously when stating results that hold for both systems. Deducibility notation in such statements will be

$$\vdash_{\beta[\eta]} .$$

To emphasize the contrast with \vdash_{β} and $\vdash_{\beta\eta}$ we shall call \vdash_{λ} , in the present chapter only,

$$\vdash_{\text{no eq}} .$$

4A1.2 Example Some terms become typable in $\text{TA}_{\lambda+\beta[\eta]}$ that were not typable in TA_{λ} . For example, consider the term

$$P \equiv (\lambda uv.v)(\lambda x.xx)$$

which was shown in 2C2.2 to have no type although it reduces to \mathbf{I} which has type $a \rightarrow a$. In $\text{TA}_{\lambda+\beta[\eta]}$ we can assign a type to P by the following deduction.

$$\frac{\frac{v:a \mapsto v:a}{\mapsto (\lambda v.v):a \rightarrow a} (\rightarrow \mathbf{I}) \quad \lambda v.v =_{\beta} (\lambda uv.v)(\lambda x.xx)}{\mapsto (\lambda uv.v)(\lambda x.xx):a \rightarrow a.} (\text{Eq}_{\beta})$$

4A1.3 Remark By adding (Eq) to TA_{λ} we have trivially solved the problem of making the set of a term's types invariant with respect to conversion. But for this easy gain we pay at least two prices.

(i) Rule (Eq) is undecidable. That is, there is no algorithm that will apply to arbitrary Γ, τ, M, N and decide whether the formula $\Gamma \mapsto N:\tau$ is deducible from $\Gamma \mapsto M:\tau$ by (Eq). (Because the relations $=_{\beta}$ and $=_{\beta\eta}$ are undecidable.) This defect can be remedied by replacing (Eq) by a series of rules, each one corresponding to an axiom in a suitable definition of $=_{\beta}$ or $=_{\beta\eta}$, but this solution is far from neat.

(ii) The subject-construction theorem (2B2) breaks down: when there are (Eq)-steps present the structure of a deduction is no longer dictated by that of its subject. Also when the proof of a formula $\Gamma \mapsto M:\tau$ contains (Eq) we cannot infer that $\text{Subjects}(\Gamma) = FV(M)$ as we did in Lemma 2A10.

At first sight (i) and (ii) are serious drawbacks. However, the following theorems will show that (ii), at least, is not as bad as it might seem; we shall see that in fact rule (Eq) plays an unexpectedly small role in deductions and that $\text{TA}_{\lambda+\beta[\eta]}$ is tied to TA_{λ} very closely indeed.

4A2 Equality-postponement Theorem In $\text{TA}_{\lambda+\beta[\eta]}$ every deduction of $\Gamma \mapsto M:\tau$ can be changed into one in which the only use of (Eq) is at the end. That is, if $\Gamma \vdash_{\beta[\eta]} M:\tau$ then there exists a term $M' =_{\beta[\eta]} M$ such that

$$\Gamma \vdash_{\text{no eq}} M':\tau.$$

Proof Let Δ be a $\text{TA}_{\lambda+\beta[\eta]}$ -deduction of $\Gamma \mapsto M:\tau$. If (Eq) occurs in Δ above an occurrence of $(\rightarrow E)$, the relevant part of Δ will look like this:

$$\frac{\frac{\Gamma_1 \mapsto P:\rho \rightarrow \sigma \quad P =_{\beta[\eta]} P'}{(\text{Eq})} \quad \frac{\Gamma_2 \mapsto Q:\rho \quad Q =_{\beta[\eta]} Q'}{(\text{Eq})}}{\frac{\Gamma_1 \mapsto P':\rho \rightarrow \sigma \quad \Gamma_2 \mapsto Q':\rho}{(\rightarrow E)} \quad \Gamma_1 \cup \Gamma_2 \mapsto (P'Q'):\sigma.}$$

Replace this part by the following, noting that the assumptions and the conclusion stay the same.

$$\frac{\frac{\Gamma_1 \mapsto P:\rho \rightarrow \sigma \quad \Gamma_2 \mapsto Q:\rho}{(\rightarrow E)} \quad \frac{\Gamma_1 \cup \Gamma_2 \mapsto (PQ):\sigma \quad PQ =_{\beta[\eta]} P'Q'}{(\text{Eq})}}{\Gamma_1 \cup \Gamma_2 \mapsto (P'Q'):\sigma.}$$

If (Eq) occurs in Δ above an occurrence of $(\rightarrow I)_{\text{main}}$, the relevant part of Δ will look like this:

$$\frac{\frac{\Gamma, x:\rho \mapsto P:\sigma \quad P =_{\beta[\eta]} P'}{(\text{Eq})} \quad \frac{\Gamma, x:\rho \mapsto P':\sigma}{(\rightarrow I)_{\text{main}}}}{\Gamma \mapsto (\lambda x.P'):\rho \rightarrow \sigma.}$$

Replace this part by the following:

$$\frac{\frac{\Gamma, x:\rho \mapsto P:\sigma}{(\rightarrow I)_{\text{main}}} \quad \frac{\Gamma \mapsto (\lambda x.P):\rho \rightarrow \sigma \quad \lambda x.P =_{\beta[\eta]} \lambda x.P'}{(\text{Eq})}}{\Gamma - x \mapsto (\lambda x.P'):\rho \rightarrow \sigma.}$$

If (Eq) occurs above $(\rightarrow I)_{\text{vac}}$ make a similar replacement. (By the way, $(\rightarrow I)_{\text{vac}}$ is distinguished from $(\rightarrow I)_{\text{main}}$ in Definition 2A8 by whether x occurs in the context on its top line, not by whether $x \in FV(P)$, so converting P' to P will not change an application of $(\rightarrow I)_{\text{vac}}$ to one of $(\rightarrow I)_{\text{main}}$ or vice versa.¹)

Finally, if (Eq) occurs in Δ above another (Eq), combine them into one in the obvious way. The above replacement-procedure will terminate in a deduction with (Eq) at the end (if (Eq) occurs in it at all). \square

4A2.1 Note The above proof depends on little more than the very simple fact that if $P =_{\beta[\eta]} P'$ and $Q =_{\beta[\eta]} Q'$ then

$$PQ =_{\beta[\eta]} P'Q', \quad \lambda x.P =_{\beta[\eta]} \lambda x.P'.$$

But as a consequence it allows us to extend several important results from TA_λ to $\text{TA}_{\lambda+\beta[\eta]}$, as follows.

¹ In TA_λ , x occurred in the context on the top line of $(\rightarrow I)$ iff $x \in FV(P)$, but as remarked in 4A1.3(ii), this does not necessarily hold when (Eq) is present.

4A3 Weak Normalization Theorem for $TA_{\lambda+\beta[\eta]}$ Every $TA_{\lambda+\beta[\eta]}$ -typable term M has a $\beta[\eta]$ -nf M_* , and furthermore

$$\Gamma \vdash_{\beta[\eta]} M:\tau \iff \Gamma \vdash_{\text{no eq}} M_*:\tau.$$

Proof Let $\Gamma \vdash_{\beta[\eta]} M:\tau$. Then by 4A2 there exists $M' =_{\beta[\eta]} M$ such that

$$(1) \quad \Gamma \vdash_{\text{no eq}} M':\tau.$$

By the weak normalization theorem for TA_{λ} (2D5), M' has a nf M_* ; and by the Church-Rosser theorem M' reduces to M_* and M_* is also the nf of M . They by (1) and the subject-reduction theorem (2C1),

$$(2) \quad \Gamma \vdash_{\text{no eq}} M_*:\tau.$$

Conversely, if (2) holds then $\Gamma \vdash_{\beta[\eta]} M:\tau$ by (Eq). \square

4A3.1 Warning The weak normalization theorem cannot be strengthened to strong normalization. For example let

$$M \equiv (\lambda uv.v)((\lambda x.xx)(\lambda x.xx)).$$

Then $M =_{\beta} I$ so M is typable in both $TA_{\lambda+\beta}$ and $TA_{\lambda+\beta\eta}$. But M has an infinite reduction, so strong normalization fails for both systems.

4A3.2 Corollary The relation $\Gamma \vdash_{\beta[\eta]} M:\tau$ is equivalent to each of the following:

- (i) $\Gamma \vdash_{\text{no eq}} M_*:\tau$,
- (ii) $\Gamma \vdash M_* \vdash_{\text{no eq}} M_*:\tau$,
- (iii) $\Gamma \vdash_{\beta[\eta]} M_*:\tau$,
- (iv) $\Gamma \vdash M_* \vdash_{\beta[\eta]} M_*:\tau$,
- (v) $\Gamma \vdash M_* \vdash_{\beta[\eta]} M:\tau$,
- (vi) $\Gamma \vdash M \vdash_{\beta[\eta]} M:\tau$.

Proof Each of (i)–(vi) implies $\Gamma \vdash_{\beta[\eta]} M:\tau$ by (Eq) and the weakening property of “ \vdash ” (which holds for $\vdash_{\beta[\eta]}$ just as for \vdash_{λ} , see 2A9.1).

For the converse, the relation $\Gamma \vdash_{\beta[\eta]} M:\tau$ implies (i) by 4A3. And (i) implies (ii) by 2A11. Next, (ii) implies (iv) trivially, and (iv) implies (iii), (v) and (vi) by (Eq) and weakening. \square

4A3.3 Note When $\Gamma \vdash_{\beta[\eta]} M:\tau$ we cannot infer that $\text{Subjects}(\Gamma) \supseteq FV(M)$, but by the above corollary and 2A11 we can infer that

$$\text{Subjects}(\Gamma) \supseteq FV(M_*).$$

The next theorem will express the content of Corollary 4A3.2 very neatly for the case that M is closed.

4A4 Definition If M is closed, the set of all τ such that $\vdash_{\beta[\eta]} M:\tau$ holds will be called

$$\text{Types}_{\beta[\eta]}(M).$$

To emphasize the contrast, the set $Types(M)$ defined in 2C3 will be called here $Types_{no\ eq}(M)$.

4A5 Theorem *If M is closed and has β - and $\beta\eta$ -nf's $M_{*\beta}$ and $M_{*\beta\eta}$ respectively, then*

- (i) $Types_{\beta}(M) = Types_{\beta}(M_{*\beta}) = Types_{no\ eq}(M_{*\beta})$,
- (ii) $Types_{\beta\eta}(M) = Types_{\beta\eta}(M_{*\beta\eta}) = Types_{no\ eq}(M_{*\beta\eta})$.

Proof By 4A3.2. □

4A6 Definition (Principal types in $TA_{\lambda+\beta[\eta]}$) In $TA_{\lambda+\beta[\eta]}$ a type τ is a *principal type* of a term M iff

- (i) $\Gamma \mapsto M:\tau$ is deducible for some Γ ,
- (ii) if $\Gamma' \mapsto M:\tau'$ is deducible then τ' is an instance of τ .

4A7 PT Theorem for $TA_{\lambda+\beta[\eta]}$ *In $TA_{\lambda+\beta[\eta]}$ every typable term M has a principal type, and it is the same as the principal type in TA_{λ} of the $\beta[\eta]$ -nf of M .*

Proof By 4A3.2 and the PT theorem for TA_{λ} (3A6). □

4A8 Remark Results 4A2–7 show that the effect of adding rule (Eq) has been much less than we might have feared: (Eq) simply transfers types from normal forms to the terms that reduce to them, and consequently $TA_{\lambda+\beta[\eta]}$ is little more than the theory of assigning types to $\beta[\eta]$ -normal forms in TA_{λ} .

However, this is not quite the whole story. Rule (Eq) was motivated at the start of this section by semantic considerations and the semantics of $TA_{\lambda+\beta[\eta]}$ will be investigated in the next section.

But first two more syntactic results will be stated. The first is an analogue for $TA_{\lambda+\beta}$ of the subject-reduction theorem for TA_{λ} : if it merely asserted closure under \triangleright_{β} it would be trivial but it says slightly more.

4A9 Theorem (η -closure of $TA_{\lambda+\beta}$) *If $\Gamma \vdash_{\beta} P:\tau$ and $P \triangleright_{\beta\eta} Q$, then*

$$\Gamma \vdash_{\beta} Q:\tau.$$

Proof If $\Gamma \vdash_{\beta} P:\tau$ then by 4A3, P has a β -nf $P_{*\beta}$ such that

$$(1) \quad \Gamma \vdash_{no\ eq} P_{*\beta}:\tau.$$

Now $P \triangleright_{\beta\eta} Q$ and $P \triangleright_{\beta} P_{*\beta}$, so by 1C7.1 there exists T such that

$$Q \triangleright_{\beta} T, \quad P_{*\beta} \triangleright_{\beta\eta} T.$$

(In fact $P_{*\beta} \triangleright_{\eta} T$ by 1C9.2.) By (1) and subject-reduction (2C1),

$$\Gamma \vdash_{no\ eq} T:\tau.$$

Hence by (Eq $_{\beta}$),

$$\Gamma \vdash_{\beta} Q:\tau. \quad \square$$

4A10 Theorem (Undecidability of $\text{TA}_{\lambda+\beta[\eta]}$) *There is no algorithm that accepts arbitrary Γ, M, τ as inputs and decides whether $\Gamma \vdash_{\beta[\eta]} M:\tau$.*

Proof Let $\Gamma = \emptyset$ and $\tau \equiv a \rightarrow a$. For any M , if $\vdash_{\beta[\eta]} M:a \rightarrow a$ then M is closed and by 4A3, M has a nf M^* which is closed and has the property

$$\vdash_{\text{no eq}} M^*:a \rightarrow a$$

But by the subject-construction theorem (2B2) it is easy to see that a closed $\beta[\eta]$ -nf with type $a \rightarrow a$ in TA_λ must have form $\lambda x \cdot x$ for some x . Hence

$$\vdash_{\beta[\eta]} M:a \rightarrow a \iff M =_{\beta[\eta]} \mathbf{I}$$

Thus a test for $\vdash_{\beta[\eta]}$ would give a test for convertibility to \mathbf{I} , contrary to standard undecidability results (e.g. Barendregt 1984 Thm. 6.6.2 or HS 86 Cor. 5.6.1). \square

4B Semantics and completeness

In this section $\text{TA}_{\lambda+\beta}$ will be interpreted in an arbitrary λ -model and its rules proved sound and complete, and the same will be done for $\text{TA}_{\lambda+\beta\eta}$ in extensional λ -models. The concept of λ -model has been motivated, defined and thoroughly discussed in Barendregt 1984 Ch. 5 (especially Def. 5.2.7), and in HS 86 Ch. 11. There are several definitions of λ -model in these sources, all equivalent, and the one presented below has been chosen mainly because it emphasizes the concepts needed in the completeness proof. It is from HS 86 Def. 11.3.

4B1 Definition (Semantic environments) Given a non-empty set \mathbb{D} , a *semantic environment* in \mathbb{D} is any function E that assigns to each term-variable v a member $E(v)$ of \mathbb{D} . If E is a semantic environment, x a variable and $d \in \mathbb{D}$, the semantic environment

$$[d/x]E$$

is defined to be the same as E except that it assigns d to x . (By the way, if $E(x) = d$ then $[d/x]E = E$.)

4B2 Definition (λ -models) A λ -model is a triple $\mathcal{D} = \langle \mathbb{D}, \cdot, [\] \rangle$ where \mathbb{D} is a set with at least two members, \cdot is a 2-place function such that

$$d_1, d_2 \in \mathbb{D} \implies d_1 \cdot d_2 \in \mathbb{D},$$

and $[\]$ is a function that assigns to each E and each term M a member of \mathbb{D} called $[M]_E$, such that

- (i) $[x]_E = E(x)$ *for all term-variables x ,*
- (ii) $[PQ]_E = [P]_E \cdot [Q]_E$,
- (iii) $[\lambda x \cdot P]_E \cdot d = [P]_{[d/x]E}$ *for all $d \in \mathbb{D}$,*
- (iv) $[M]_{E_1} = [M]_{E_2}$ *if $E_1(v) = E_2(v)$ for all $v \in FV(M)$,*
- (v) $[M]_E = [N]_E$ *if $M \equiv_\alpha N$,*
- (vi) $[\lambda x \cdot P]_E = [\lambda x \cdot Q]_E$ *if $[P]_{[d/x]E} = [Q]_{[d/x]E}$ for all $d \in \mathbb{D}$.*

4B2.1 Notation The function \cdot is called *application* and $\llbracket \cdot \rrbracket$ is called the *interpretation* mapping.

Note that in the above definition as elsewhere in this book, “=” without a subscript “ β ” or “ $\beta\eta$ ” means “is identical to”.

4B3 Definition A λ -model $\mathcal{D} = \langle \mathbb{D}, \cdot, \llbracket \cdot \rrbracket \rangle$ is called *extensional* iff, for all d_1 and $d_2 \in \mathbb{D}$,

$$(\forall e \in \mathbb{D})(d_1 \cdot e = d_2 \cdot e) \implies d_1 = d_2.$$

4B4 Lemma (i) Every λ -model \mathcal{D} satisfies the theory of β -equality in the sense that

$$M =_{\beta} N \implies (\forall E) \llbracket M \rrbracket_E = \llbracket N \rrbracket_E.$$

(ii) A λ -model \mathcal{D} is extensional iff it satisfies $\beta\eta$ in the sense that

$$M =_{\beta\eta} N \implies (\forall E) \llbracket M \rrbracket_E = \llbracket N \rrbracket_E.$$

Proof HS 86 Thms. 11.12 and 11.15. □

4B4.1 Note A λ -model \mathcal{D} is a model of *untyped* λ -calculus, i.e. every term receives an interpretation in \mathbb{D} whether it is typable or not. This agrees with the Curry approach to type-theory described in 2A3. In contrast, in a model of Church’s type-theory only typed terms are interpreted, and instead of one domain \mathbb{D} the model has a distinct domain for each type. The definition of model for Church’s system can be found in Henkin 1950.

Types are interpreted in an arbitrary λ -model as follows.

4B5 Definition (Interpreting types) Let $\mathcal{D} = \langle \mathbb{D}, \cdot, \llbracket \cdot \rrbracket \rangle$ be any λ -model. An *interpretation of the type-variables* is any function V that assigns a subset of \mathbb{D} to each type-variable. Each such V generates an *interpretation of all the types*, $\llbracket \cdot \rrbracket_V$, defined as follows:

- (i) $\llbracket a \rrbracket_V = V(a)$ for all type-variables a ,
- (ii) $\llbracket \sigma \rightarrow \tau \rrbracket_V = \{d \in \mathbb{D} : (\forall d' \in \llbracket \sigma \rrbracket_V)(d \cdot d' \in \llbracket \tau \rrbracket_V)\}$.

4B6 Definition (Satisfaction) Let $\mathcal{D} = \langle \mathbb{D}, \cdot, \llbracket \cdot \rrbracket \rangle$ be a λ -model. A type-assignment $M : \tau$ is said to be *satisfied in \mathcal{D} by E and V* iff

$$\llbracket M \rrbracket_E \in \llbracket \tau \rrbracket_V.$$

A set Γ of type-assignments is said to be *satisfied* iff all its members are satisfied. A formula $\Gamma \mapsto M : \tau$ is said to be *valid in \mathcal{D}* iff every pair E, V satisfying Γ in \mathcal{D} also satisfies $M : \tau$ in \mathcal{D} .

Iff $\Gamma \mapsto M : \tau$ is valid in all λ -models (respectively, all extensional λ -models), we say

$$\Gamma \vDash_{\beta} M : \tau, \quad \Gamma \vDash_{\beta\eta} M : \tau.$$

4B6.1 Note The above definition of satisfaction is called the *simple semantics* for type-assignment. There are several other semantics-definitions in the literature, for example see Hindley 1983 §§4–5 and Mitchell 1988 §3.

Many different particular λ -models are described in the literature (for some examples see Barendregt 1984 Ch. 18 and HS 86 Ch. 12), but we shall not need to know about them here. In fact the only model used in the completeness-proof below will be the simplest of all kinds of model, a *term-model*.

4B7 Definition (The term-models $\mathbb{T}\mathbb{M}_\beta$, $\mathbb{T}\mathbb{M}_{\beta\eta}$) The domain \mathbb{D} of $\mathbb{T}\mathbb{M}_\beta$ is the set of all β -equality-classes of λ -terms: in more detail, for each M we define

$$[M]^\beta = \{P : P =_\beta M\},$$

and then define

$$\mathbb{D} = \{[M]^\beta : M \text{ is a } \lambda\text{-term}\}.$$

Application in $\mathbb{T}\mathbb{M}_\beta$ is defined thus:

$$[M]^\beta \cdot [N]^\beta = [MN]^\beta.$$

Terms are interpreted in $\mathbb{T}\mathbb{M}_\beta$ thus: if $FV(M) = \{x_1, \dots, x_n\}$ and $E(x_i) = [Q_i]^\beta$ for $i = 1, \dots, n$ (for some terms Q_1, \dots, Q_n), define

$$[M]_E = [[Q_1/x_1, \dots, Q_n/x_n]M]^\beta.$$

The definition of $\mathbb{T}\mathbb{M}_{\beta\eta}$ is similar, starting with $[M]^{\beta\eta} = \{P : P =_{\beta\eta} M\}$.

4B7.1 Definition A semantic environment in $\mathbb{T}\mathbb{M}_{\beta[\eta]}$ called E_0 is defined by setting

$$E_0(x) = [x]^{\beta[\eta]}.$$

4B7.2 Lemma (i) $\mathbb{T}\mathbb{M}_\beta$ and $\mathbb{T}\mathbb{M}_{\beta\eta}$ are λ -models and $\mathbb{T}\mathbb{M}_{\beta\eta}$ is extensional.

(ii) In $\mathbb{T}\mathbb{M}_{\beta[\eta]}$ we have $[M]_{E_0} = [M]^{\beta[\eta]}$ for all M .

Proof Routine. (For (i) see, e.g., Barendregt 1984 Prop. 5.2.12.) □

4B8 Soundness Theorem Let Γ be any type-context, M any λ -term and τ any type. Then

$$\Gamma \vdash_{\beta[\eta]} M : \tau \implies \Gamma \models_{\beta[\eta]} M : \tau.$$

Proof Straightforward. (See, e.g., Ben-Yelles 1979 Thm. 4.17.) □

4B9 Completeness Theorem Let Γ be any type-context, M any λ -term and τ any type. Then

$$\Gamma \models_{\beta[\eta]} M : \tau \implies \Gamma \vdash_{\beta[\eta]} M : \tau.$$

Proof [Hindley 1983a §3.] We shall prove the β -case; the proof of the $\beta\eta$ -case is almost identical. By the definition of “type-context”, Γ is a finite set of type-assignments

$$\Gamma = \{x_1:\rho_1, \dots, x_m:\rho_m\},$$

where x_1, \dots, x_m are distinct. Extend Γ to an infinite set Γ^+ in which every type in the language of TA_λ is assigned to an infinite number of term-variables (and no variable receives more than one type).

This can be done as follows. First list all the types as an infinite sequence, say τ_1, τ_2, \dots . Then note that in 1A1 we assumed the λ -language to contain an infinite number of term-variables, so there are an infinite number of variables distinct from x_1, \dots, x_m and from the free variables of M ; choose from these an infinite number of disjoint infinite sequences, say

$$\{v_{1,1}, v_{1,2}, \dots\}, \quad \{v_{2,1}, v_{2,2}, \dots\}, \quad \{v_{3,1}, v_{3,2}, \dots\}, \dots$$

Then define

$$\Gamma^+ = \Gamma \cup \{v_{i,j}:\tau_i: i \geq 1, j \geq 1\}.$$

In what follows, the notation “ $\Gamma^+ \vdash_\beta P:\sigma$ ” (for any given P and σ) will mean that there is a finite subset Γ^* of Γ^+ such that $\Gamma^* \vdash_\beta P:\sigma$.

In \mathbb{TM}_β define an interpretation v_0 of the type-variables by setting, for each type-variable a ,

$$v_0(a) = \{[P]^\beta: \Gamma^+ \vdash_\beta P:a\}.$$

Extend v_0 to an interpretation $[\]_{v_0}$ of all types as in 4B5. We shall now prove that, for all σ and all P :

$$(1) \quad [P]^\beta \in [\sigma]_{v_0} \iff \Gamma^+ \vdash_\beta P:\sigma;$$

i.e. that the interpretation of σ is exactly the set of all terms which receive type σ when the assignments in Γ^+ are assumed. \square

Proof of (1) Use induction on σ . If σ is an atom we have (1) by the definition of v_0 .

For the induction step let $\sigma \equiv \xi \rightarrow \eta$. By the definition of \mathbb{TM}_β and of the interpretation of $\xi \rightarrow \eta$ we have

$$[P]^\beta \in [\xi \rightarrow \eta]_{v_0} \iff (\forall Q) ([Q]^\beta \in [\xi]_{v_0} \implies [PQ]^\beta \in [\eta]_{v_0})$$

and by the induction hypothesis the right-hand side is equivalent to

$$(\forall Q) (\Gamma^+ \vdash_\beta Q:\xi \implies \Gamma^+ \vdash_\beta PQ:\eta)$$

So to complete the induction step it is enough to prove

$$(2) \quad \left\{ \begin{array}{l} (\forall Q) (\Gamma^+ \vdash_\beta Q:\xi \implies \Gamma^+ \vdash_\beta PQ:\eta) \\ \iff \Gamma^+ \vdash_\beta P:(\xi \rightarrow \eta). \end{array} \right.$$

Proof of (2) For “ \Leftarrow ”, use rule $(\rightarrow E)$. To prove “ \Rightarrow ”, suppose the upper clause in (2) holds and, as a special case, take Q to be a variable z not occurring in P and such that Γ^+ contains the assignment $z:\xi$. (By the construction of Γ^+ , z exists.) Then by the upper clause in (2),

$$\Gamma^+ \vdash_\beta Pz:\eta.$$

Hence by rule $(\rightarrow I)$,

$$\Gamma^+ - z \vdash_\beta (\lambda z.Pz):\xi \rightarrow \eta.$$

This means that there is a finite subset Γ^* of $\Gamma^+ - z$ such that

$$\Gamma^* \vdash_\beta (\lambda z.Pz):\xi \rightarrow \eta;$$

hence by 4B9

$$\Gamma^* \vdash_\beta P:\xi \rightarrow \eta,$$

which gives the “ \Rightarrow ” part of (2). Hence (2) holds and the proof of (1) is complete.

Deduction of the theorem from (1) If $\Gamma \models_\beta M:\tau$ then in particular, for $\mathbb{T}\mathbb{M}_\beta$ and E_0 and V_0 we have

$$[M]^\beta \in [\tau]_{V_0}$$

Hence by (1),

$$\Gamma^+ \vdash_\beta M:\tau.$$

That is $\Gamma^* \vdash_\beta M:\tau$ for some finite $\Gamma^* \subseteq \Gamma^+$. Hence by 4A3.2(vi),

$$(3) \quad \Gamma^* \upharpoonright M \vdash_\beta M:\tau.$$

But $\Gamma^* \upharpoonright M = \Gamma \upharpoonright M$ because $\Gamma^* \subseteq \Gamma^+$ and the extra term-variables in Γ^+ were chosen to be distinct from those in M . Hence by the weakening-property of \vdash ,

$$\Gamma \vdash_\beta M:\tau. \quad \square$$

4B10 Remark (Too many completeness-proofs?) In the above completeness-proof the full strength of the assumption that $\Gamma \mapsto M:\tau$ was valid in *every* model was not used; all we needed was that it was valid in *one* model, the term-model. There is another completeness proof in Barendregt et al. 1983, which uses a more complex kind of term model called a *filter model*. And there is a third proof in Coppo 1984 using the *graph model* $P\omega$. It looks almost as if the choice of model is quite irrelevant to the success of the completeness-proof.

Furthermore, $TA_{\lambda+\beta}$ and $TA_{\lambda+\beta\eta}$ can be proved sound and complete for at least three other variant definitions of the semantics, see Hindley 1983a §5, Hindley 1983b, and Mitchell 1988 §§3–4.

This richness of completeness-proofs says in effect that the differences which undoubtedly exist between these various kinds of models and definitions of semantics cannot be expressed in the rather limited language of $TA_{\lambda+\beta}$ and $TA_{\lambda+\beta\eta}$.

4B11 Remark By the way, although earlier remarks in this chapter may have given the impression that without equality-invariance a type-theory has no reasonable semantics, this is not strictly true. Plotkin 1994 has described a semantics for TA_λ (and some extensions) based on modelling the concept of reduction rather than equality. Plotkin's approach seems closer to the "spirit" of TA_λ ; in fact TA_λ has at its heart the notion of reduction rather than conversion, and has always been seen primarily as a tool for eliminating type-conflicts in practical computations or reductions, rather than as a formal theory of equality.

A version using typed terms

In Chapter 2 some care was taken to distinguish the Curry and Church approaches to type-theory from each other. Curry's approach involved assigning types to pre-existing untyped terms with each term receiving either an infinite set of types or none at all, whereas in Church's the terms were defined with built-in types with each term having a single type (see 2A3). In Curry's approach the types contained variables, in Church's they contained only constants.

This book focuses on the Curry approach. However, even in this approach it turns out to be very useful to introduce a typed-term language as an alternative notation for TA_λ -deductions. Although the tree-notation introduced in Chapter 2 shows very clearly what assumptions are needed in deducing what conclusions, it takes up a lot of space and is hard to visualise when the deduction is in any way complicated. And when manipulations and reductions of deductions are under discussion it is almost unmanageable. A much more compact alternative notation is needed, and this is what the typed terms in the present chapter will give.

We shall also define reduction of typed terms; typed terms will be shown in the next chapter to encode deductions in propositional logic as well as in TA_λ , and their reduction will be essentially the same as the reduction of deductions that is a standard tool in proof theory.

By the way, a cynical reader might think we are simply abandoning Curry's approach here and replacing it by Church's, but this is not so; the main positive feature of Curry's approach is the expressive power gained from its use of type-variables and the presence of an underlying language of untyped terms, and we shall not abandon these. All we shall do is replace space-hungry deduction-tree diagrams by neat and compact typed terms. And these terms will differ in several important ways from those in a true Church-style system (see 5A1.5 for example); they will simply be codes for TA_λ -deductions, nothing more.

5A Typed terms

In this section we shall define a system of typed terms isomorphic to TA_λ -deductions: for each Γ a set $\mathbb{T}(\Gamma)$ of typed terms will be defined that will encode deductions of formulae of form

$$\Gamma^- \mapsto M:\tau \qquad (\Gamma^- \subseteq \Gamma).$$

It might be useful to read the detailed definition of a TA_λ -deduction in 9C in parallel with this section.

5A1 Definition (Typed terms, $\mathbb{T}\mathbb{T}(\Gamma)$) Given a type-context Γ the set $\mathbb{T}\mathbb{T}(\Gamma)$ of *typed terms relative to Γ* is a set of expressions defined thus:

- (i) if Γ contains $x:\sigma$ then the expression x^σ is in $\mathbb{T}\mathbb{T}(\Gamma)$ and is called a *typed variable*;
- (ii) if $\Gamma_1 \cup \Gamma_2$ is consistent and $M^{\sigma \rightarrow \tau} \in \mathbb{T}\mathbb{T}(\Gamma_1)$ and $N^\sigma \in \mathbb{T}\mathbb{T}(\Gamma_2)$, then

$$(M^{\sigma \rightarrow \tau} N^\sigma)^\tau \in \mathbb{T}\mathbb{T}(\Gamma_1 \cup \Gamma_2);$$

- (iii) if Γ is consistent with $\{x:\sigma\}$ and $M^\tau \in \mathbb{T}\mathbb{T}(\Gamma)$, then

$$(\lambda x^\sigma. M^\tau)^{\sigma \rightarrow \tau} \in \mathbb{T}\mathbb{T}(\Gamma - x).$$

If M^τ is a typed term (relative to some Γ), τ is called *the type of M^τ* .

5A1.1 Notation Typed terms will be abbreviated using the same conventions as for untyped terms. Also some type-superscripts may be omitted. For example, depending on which of its types are to be emphasised in a particular discussion, a typed term $(x^{\rho \rightarrow \sigma} y^\rho)^\sigma$ may be called any of

$$(xy)^\sigma, \quad x^{\rho \rightarrow \sigma} y^\rho, \quad x^{\rho \rightarrow \sigma} y.$$

5A1.2 Exercise (Compare 2A8.2–4.) Show that $\mathbb{T}\mathbb{T}(\emptyset)$ contains the typed terms

$$\begin{aligned} &(\lambda x^{a \rightarrow b} y^{c \rightarrow a} z^c. (x^{a \rightarrow b} (y^{c \rightarrow a} z^c)^a)^b)^{(a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b}, \\ &(\lambda x^a. x^a)^{a \rightarrow a}, \\ &(\lambda x^a y^b. x^a)^{a \rightarrow b \rightarrow a}. \end{aligned}$$

5A1.3 Note (i) In the following term x appears decorated with two different types:

$$Q^b \equiv (x^{(a \rightarrow a) \rightarrow b} (\lambda x^a. x^a)^{a \rightarrow a})^b.$$

Despite this, Q^b is a genuine typed term, and in fact it is easy to verify that $Q^b \in \mathbb{T}\mathbb{T}(\Gamma)$, where $\Gamma = \{x:(a \rightarrow a) \rightarrow b\}$. Further, Q^b translates into a genuine TA_λ -deduction in an obvious way.

(ii) In contrast, the following expression is not a typed term (and does not translate into a TA_λ -deduction):

$$(x^{a \rightarrow b} x^a)^b.$$

5A1.4 Warning The subterms of a term in $\mathbb{T}\mathbb{T}(\Gamma)$ need not be in $\mathbb{T}\mathbb{T}(\Gamma)$. For example we have

$$(\lambda x^a. x^a)^{a \rightarrow a} \in \mathbb{T}\mathbb{T}(\emptyset)$$

but the subterm x^a is not in $\mathbb{T}\mathbb{T}(\emptyset)$. (It is in $\mathbb{T}\mathbb{T}(\{x:a\})$.)

5A1.5 Warning If $M^{\sigma \rightarrow \tau}$ and N^σ are typed terms we cannot always say that $(M^{\sigma \rightarrow \tau} N^\sigma)^\tau$ is one. (Because if $M^{\sigma \rightarrow \tau} \in \mathbb{T}\mathbb{T}(\Gamma_1)$ and $N^\sigma \in \mathbb{T}\mathbb{T}(\Gamma_2)$ we cannot apply 5A1(ii) unless $\Gamma_1 \cup \Gamma_2$ is consistent.)

This is a crucial difference between the present typed terms and those of Church, for example in Church 1940; the latter are typed in an “absolute” sense and have the property that if $M^{\sigma \rightarrow \tau}$ and N^σ are typed then so is $(M^{\sigma \rightarrow \tau} N^\sigma)^\tau$, but the present terms are only typed *relatively* to a given Γ .

5A2 Definition (Type-erasing) The *type-erasure* M^\dagger of M^τ is the untyped term obtained by erasing all types from M^τ .

We shall see in 5A5.1 that M^\dagger is the subject of the conclusion of the TA_λ -deduction that M^τ encodes.

5A3 Lemma If $M^\tau \in \mathbb{T}\mathbb{T}(\Gamma)$, then $FV(M^\dagger) \subseteq \text{Subjects}(\Gamma)$ and

- (i) $M^\tau \in \mathbb{T}\mathbb{T}(\Gamma^+)$ for all contexts $\Gamma^+ \supseteq \Gamma$,
- (ii) $M^\tau \in \mathbb{T}\mathbb{T}(\Gamma \upharpoonright M^\dagger)$,
- (iii) If $M^\tau \in \mathbb{T}\mathbb{T}(\Gamma')$ then $\Gamma' \upharpoonright M^\dagger = \Gamma \upharpoonright M^\dagger$.

Proof Straightforward induction on $|M^\dagger|$. □

5A4 Definition (Minimum context, $\text{Con}(M^\tau)$) If $M^\tau \in \mathbb{T}\mathbb{T}(\Gamma)$, its *minimum context* is

$$\text{Con}(M^\tau) = \Gamma \upharpoonright M^\dagger.$$

5A4.1 Note $\text{Con}(M^\tau)$ is determined uniquely by M^τ . In particular it is independent of Γ , in the sense that if $M^\tau \in \mathbb{T}\mathbb{T}(\Gamma) \cap \mathbb{T}\mathbb{T}(\Gamma')$ then $\Gamma \upharpoonright M^\dagger = \Gamma' \upharpoonright M^\dagger$ (by 5A3(iii)).

We shall see in 5A7 that if M^τ encodes a TA_λ -deduction of $\Gamma \mapsto M^\dagger : \tau$ then $\Gamma = \text{Con}(M^\tau)$. Also it is easy to see that

- (i) for all typed terms $M^{\sigma \rightarrow \tau}$ and N^σ
 - (a) $(M^{\sigma \rightarrow \tau} N^\sigma)^\tau$ is a typed term iff $\text{Con}(M^{\sigma \rightarrow \tau})$ is consistent with $\text{Con}(N^\sigma)$,
 - (b) if $\text{Con}(M^{\sigma \rightarrow \tau})$ is consistent with $\text{Con}(N^\sigma)$ then

$$\text{Con}((M^{\sigma \rightarrow \tau} N^\sigma)^\tau) = \text{Con}(M^{\sigma \rightarrow \tau}) \cup \text{Con}(N^\sigma);$$

- (ii) for all typed terms M^τ

- (a) $(\lambda x^\sigma. M^\tau)^{\sigma \rightarrow \tau}$ is a typed term iff $\text{Con}(M^\tau)$ is consistent with $x:\sigma$ (i.e. iff $\text{Con}(M^\tau)$ either contains $x:\sigma$ or does not contain x at all),
- (b) if $\text{Con}(M^\tau)$ is consistent with $x:\sigma$ then

$$\text{Con}((\lambda x^\sigma. M^\tau)^{\sigma \rightarrow \tau}) = \text{Con}(M^\tau) - x.$$

The translation mappings between typed terms and TA_λ -deductions will now be defined; they are very straightforward but will be given here in full for the sake of precision.

5A5 Definition (Translating deductions to typed terms) Let Δ be a TA_λ -deduction with conclusion $\Gamma \mapsto M:\tau$; define a typed term $T_\Delta^\tau \in \mathbb{T}(\Gamma)$ by following the clauses of the definition of Δ (see 9C1) thus:

- (i) if Δ is an axiom $x:\tau \mapsto x:\tau$, define $T_\Delta^\tau \equiv x^\tau$;
- (ii) if Δ is the result of applying $(\rightarrow E)$ to deductions Δ_1, Δ_2 of

$$\Gamma_1 \mapsto P:\sigma \rightarrow \tau, \quad \Gamma_2 \mapsto Q:\sigma,$$

and $T_{\Delta_1}^{\sigma \rightarrow \tau} \in \mathbb{T}(\Gamma_1)$ and $T_{\Delta_2}^\sigma \in \mathbb{T}(\Gamma_2)$, define

$$T_\Delta^\tau \equiv (T_{\Delta_1}^{\sigma \rightarrow \tau} T_{\Delta_2}^\sigma)^\tau;$$

- (iii) if Δ is the result of applying rule $(\rightarrow I)$ to a deduction Δ_1 of

$$\Gamma \mapsto P:\sigma,$$

and $\tau \equiv \rho \rightarrow \sigma$ and $\Gamma \cup \{x:\rho\}$ is consistent and $T_{\Delta_1}^\sigma \in \mathbb{T}(\Gamma)$, define

$$T_\Delta^\tau \equiv (\lambda x^\rho. T_{\Delta_1}^\sigma)^{\rho \rightarrow \sigma}.$$

5A5.1 Lemma If Δ is a TA_λ -deduction of $\Gamma \mapsto M:\tau$ then

$$T_\Delta^\tau \in \mathbb{T}(\Gamma), \quad T_\Delta^\tau \equiv M.$$

5A6 Definition (Translating typed terms to deductions) To each typed term M^τ we associate a TA_λ -deduction $\underline{\Delta}(M^\tau)$ as follows:

- (i) if $M^\tau \equiv x^\tau$ define $\underline{\Delta}(M^\tau)$ to be the atomic deduction

$$x:\tau \mapsto x:\tau;$$

- (ii) if $M^\tau \equiv (P^{\sigma \rightarrow \tau} Q^\sigma)^\tau$, $P^{\sigma \rightarrow \tau} \in \mathbb{T}(\Gamma_1)$, $Q^\sigma \in \mathbb{T}(\Gamma_2)$, $\Gamma_1 \cup \Gamma_2$ is consistent, and $\underline{\Delta}(P^{\sigma \rightarrow \tau})$ and $\underline{\Delta}(Q^\sigma)$ are deductions of

$$\Gamma'_1 \mapsto P^{\sigma \rightarrow \tau}:\sigma \rightarrow \tau, \quad \Gamma'_2 \mapsto Q^\sigma:\sigma$$

for some $\Gamma'_1 \subseteq \Gamma_1$ and $\Gamma'_2 \subseteq \Gamma_2$, define $\underline{\Delta}(M^\tau)$ to be the deduction obtained by applying $(\rightarrow E)$ to $\underline{\Delta}(P^{\sigma \rightarrow \tau})$ and $\underline{\Delta}(Q^\sigma)$;

- (iii) if $M^\tau \equiv (\lambda x^\rho. P^\sigma)^{\rho \rightarrow \sigma}$, $\tau \equiv \rho \rightarrow \sigma$, $P^\sigma \in \mathbb{T}(\Gamma)$, Γ is consistent with $\{x:\rho\}$, and $\underline{\Delta}(P^\sigma)$ is a deduction of

$$\Gamma' \mapsto P^\sigma:\sigma$$

for some $\Gamma' \subseteq \Gamma$, define $\underline{\Delta}(M^\tau)$ to be the deduction obtained by applying $(\rightarrow I)$ to $\underline{\Delta}(P^\sigma)$, discharging x .

5A7 Theorem The translation from Δ to T_Δ^τ is a one-to-one correspondence between all TA_λ -deductions and all typed terms, and its inverse is the translation from M^τ to $\underline{\Delta}(M^\tau)$. In particular,

- (i) if Δ is a deduction of $\Gamma \mapsto M:\tau$ then

$$T_\Delta^\tau \in \mathbb{T}(\Gamma), \quad T_\Delta^\tau \equiv M, \quad \underline{\Delta}(T_\Delta^\tau) \equiv \Delta,$$

(ii) if $M^\tau \in \mathbb{T}(\Gamma)$ then $\text{Con}(M^\tau) \subseteq \Gamma$ and $\Delta(M^\tau)$ is a deduction of

$$\text{Con}(M^\tau) \mapsto M^\tau : \tau.$$

Proof (i) By 5A5.1 and 5A6. (ii) Easy induction on $|M^\tau|$. □

5B Reducing typed terms

Most of the term-structure concepts introduced in Section 1A, and those in 9A, can be extended from untyped to typed terms in an obvious way. For example *position*, *subterm*, *occurs* and *component* are all defined for typed terms just as for untyped terms in 9A.

The only difference is that in a typed term every subterm is decorated with a type-superscript; in particular an *atom* has form x^τ not just x . For example if $M^\tau \equiv (x^{a \rightarrow b} y^a)^b$ we shall say that $x^{a \rightarrow b}$ and y^a occur in M^τ but not that x and y occur in M^τ .

Some of the concepts in 1A and 9A do not translate to typed terms quite so unambiguously however, so their typed definitions will be given below and their main properties will be stated.

5B1 Definition The *length*, $|M^\tau|$, of a typed term M^τ is the same as $|M^\tau|$, see 1A2.

5B2 Definition (Replacement) Let $P^\sigma \equiv \langle P^\sigma, p, M^\tau \rangle$ be a component of a typed term M^τ (see 9A2–3), and let T^σ be any typed term with the same type as P^σ . Define

$$\{T^\sigma / P^\sigma\}_p M^\tau$$

to be the result of replacing P^σ by T^σ at position p in M^τ , just as for untyped terms (see 9A6), but only when (i) or (ii) below holds:¹

- (i) $\text{Con}(T^\sigma) \subseteq \text{Con}(P^\sigma)$;
- (ii) the set $\text{Con}(T^\sigma) \cup \text{Con}(M^\tau) \cup \{x_1 : \xi_1, \dots, x_n : \xi_n\}$ is consistent, where $\underline{\lambda}x_1^{\xi_1}, \dots, \underline{\lambda}x_n^{\xi_n}$ are the abstractors in M^τ whose scopes contain \underline{p} .

5B2.1 Lemma (i) If 5B2(i) holds then $\{T^\sigma / P^\sigma\}_p M^\tau$ is a well-defined typed term and

$$\text{Con}(\{T^\sigma / P^\sigma\}_p M^\tau) \subseteq \text{Con}(M^\tau).$$

(ii) If 5B2(ii) holds then $\{T^\sigma / P^\sigma\}_p M^\tau$ is well-defined and

$$\text{Con}(\{T^\sigma / P^\sigma\}_p M^\tau) \subseteq \text{Con}(M^\tau) \cup \text{Con}(T^\sigma).$$

Proof Like the proofs of 9C5 and 9C6. □

5B2.2 Note If neither (i) nor (ii) holds in 5B2 we do not define $\{T^\sigma / P^\sigma\}_p M^\tau$. In this case the replacement of \underline{p}^σ by T^σ in M^τ will still produce an expression of some kind but it might not be a typed term. For example let

$$M^\tau \equiv (x^{a \rightarrow b} y^a)^b, \quad P^\sigma \equiv y^a, \quad T^\sigma \equiv x^a.$$

¹ Compare the conditions in the replacement lemmas for deductions, 9C5–6.

Then replacing y^a by x^a changes M^τ to $(x^{a \rightarrow b} x^a)^b$, which fails to be a typed term because $x:a \rightarrow b$ is inconsistent with $x:a$. (See 5A1.5.)

Also if $\rho \neq \sigma$ we do not define $\{T^\rho/P^\sigma\}_\rho M^\tau$.

5B3 Definition *Free* and *bound* variable-occurrences, and the set $FV(M^\tau)$, are defined just as in 1A6. (But all occurrences are now typed.)

5B3.1 Notes (i) $x^\sigma \in FV(M^\tau) \iff x:\sigma \in Con(M^\tau)$.

(ii) $x \in FV(M^\tau) \iff (\exists \sigma)(x^\sigma \in FV(M^\tau))$.

(iii) By (i) and the consistency of contexts,

$$x^\sigma, x^\rho \in FV(M^\tau) \implies \sigma \equiv \rho.$$

(iv) But a typed term can contain inconsistently typed variables if one variable is free and the other is bound (or if both are bound). An example is the term Q^b in 5A1.3:

$$Q^b \equiv (x^{(a \rightarrow a) \rightarrow b} (\lambda x^a. x^a)^{a \rightarrow a})^b.$$

5B4 Definition M^τ has *no bound-variable clashes* iff M^τ has no such clashes (as defined in 1A9).

5B4.1 Example Consider the term Q^b above and its type-erasure:

$$Q^b \equiv (x^{(a \rightarrow a) \rightarrow b} (\lambda x^a. x^a)^{a \rightarrow a})^b, \quad Q^b \equiv x(\lambda x. x).$$

This erasure has a clash so the above definition says Q^b has a clash. (We might feel like saying that Q^b is without clashes because $x^{a \rightarrow a \rightarrow b}$ and x^a are distinct typed variables, but this would make the theory of typed terms diverge too far from that of TA_λ -deductions. Definition 5B4 says in effect that a typed term has no clashes iff it represents a deduction whose conclusion has no clashes.)

5B5 Definition (Typed substitution) We define $[N^\sigma/x^\sigma]M^\tau$ by the following clauses. (Compare untyped substitution, 1A7.) These clauses will not cover all cases but all those in which $(\lambda x^\sigma. M^\tau)N^\sigma$ has no bound-variable clashes, which are all the cases needed in this book.

- (i) $[N^\sigma/x^\sigma]x^\sigma \equiv N^\sigma$;
- (ii) $[N^\sigma/x^\sigma]y^\tau \equiv y^\tau$ if $x \neq y$;
- (iii) $[N^\sigma/x^\sigma](P^{\rho \rightarrow \tau} Q^\rho)^\tau \equiv ([N^\sigma/x^\sigma]P^{\rho \rightarrow \tau} [N^\sigma/x^\sigma]Q^\rho)^\tau$;
- (iv) $[N^\sigma/x^\sigma](\lambda x^\xi. P^\eta)^{\xi \rightarrow \eta} \equiv (\lambda x^\xi. P^\eta)^{\xi \rightarrow \eta}$;
- (v) $[N^\sigma/x^\sigma](\lambda y^\xi. P^\eta)^{\xi \rightarrow \eta} \equiv (\lambda y^\xi. P^\eta)^{\xi \rightarrow \eta}$ if $x \neq y$, $x \notin FV(P^\eta)$;
- (vi) $[N^\sigma/x^\sigma](\lambda y^\xi. P^\eta)^{\xi \rightarrow \eta} \equiv (\lambda y^\xi. [N^\sigma/x^\sigma]P^\eta)^{\xi \rightarrow \eta}$
if $x \neq y$, $x \in FV(P^\eta)$, $y \notin FV(N^\sigma)$;
- (vii) $[N^\sigma/x^\sigma](\lambda y^\xi. P^\eta)^{\xi \rightarrow \eta} \equiv (\lambda z^\xi. [N^\sigma/x^\sigma][z^\xi/y^\xi]P^\eta)^{\xi \rightarrow \eta}$
if $x \neq y$, $x \in FV(P^\eta)$, $y \in FV(N^\sigma)$.

(In (vii) z is the first untyped variable not occurring in $N^\sigma P^\eta$.)

5B5.1 Notes The restrictions in the above clauses may seem over-strong in that they involve *untyped* variables and terms; but if they were weakened the last part of the useful lemma 5B5.2 below would fail.

Clause (iv) includes the case $\xi \neq \sigma$. But in this case x^σ cannot occur free in P^η , because if it did then $\lambda x^\xi.P^\eta$ would not be a typed term (see Definition 5A1(iii)). Thus substituting for x^σ should change nothing. And this is exactly what (iv) says.

No attempt has been made to define $[N^\rho/x^\sigma]$ when $\rho \neq \sigma$ or $[N^\sigma/x^\sigma]x^\tau$ when $\tau \neq \sigma$.

5B5.2 Lemma *If $((\lambda x^\sigma.M^\tau)N^\sigma)^\tau$ is a typed term with no bound-variable clashes, then*

- (i) $[N^\sigma/x^\sigma]M^\tau$ is defined and is a typed term with type τ ,
- (ii) $[N^\sigma/x^\sigma]M^\tau \equiv M^\tau$ if $x^\sigma \notin FV(M^\tau)$,
- (iii) $Con([N^\sigma/x^\sigma]M^\tau) \subseteq (Con(M^\tau) - x) \cup Con(N^\sigma)$,
- (iv) $([N^\sigma/x^\sigma]M^\tau)^\tau \equiv_\alpha [N^\sigma/x]M^\tau$.

Proof Parts (i)–(iv) are proved together by a straightforward but boring induction on $|M^\tau|$. (The assumption about $((\lambda x^\sigma.M^\tau)N^\sigma)^\tau$ implies in particular that $Con(M^\tau) - x$ is consistent with $Con(N^\sigma)$ and that x does not occur in N^σ .) \square

5B6 Definition (Typed α -conversion) The relation \equiv_α is defined just as for untyped terms (see 1A8), using replacements with form

$$(\alpha) \quad \lambda x^\sigma.M^\tau \equiv_\alpha \lambda y^\sigma.[y^\sigma/x^\sigma]M^\tau \quad (y \notin FV(M^\tau)).$$

5B6.1 Lemma *If $\lambda x^\sigma.M^\tau$ is a typed term and $y \notin FV(M^\tau)$, then $[y^\sigma/x^\sigma]M^\tau$ is defined and both sides of (α) are typed terms with the same type and minimum context. Hence the class of all typed terms is closed under α -conversion, and*

$$P^\tau \equiv_\alpha Q^{\tau'} \implies \tau \equiv \tau' \text{ and } Con(P^\tau) = Con(Q^{\tau'}).$$

5B6.2 Warning The condition “ $y \notin FV(M^\tau)$ ” in (α) cannot be weakened to “ $y^\sigma \notin FV(M^\tau)$ ”. Because if it were, we could α -convert a typed term to an expression that was not one, thus:

$$\lambda x^{a \rightarrow b}.x^{a \rightarrow b}y^a \equiv_\alpha \lambda y^{a \rightarrow b}.y^{a \rightarrow b}y^a.$$

5B7 Definition (Typed redexes and reduction) Typed \triangleright_β and $\triangleright_{\beta\eta}$ are defined just like the untyped relations in 1B and 1C. In particular *typed β - and η -redexes* have form

$$((\lambda x^\sigma.M^\tau)^{\sigma \rightarrow \tau}N^\sigma)^\tau, \quad (\lambda x^\sigma.(P^{\sigma \rightarrow \tau}x^\sigma)^\tau)^{\sigma \rightarrow \tau}$$

with $x \notin FV(P^{\sigma \rightarrow \tau})$, and their *contracta* are, respectively,

$$[N^\sigma/x^\sigma]M^\tau, \quad P^{\sigma \rightarrow \tau}.$$

And, just as in Chapter 1, a *contraction* is a replacement of a redex by its contractum.

A *typed β -nf* (*η -nf*, *$\beta\eta$ -nf*) is a typed term containing no β -(η -, $\beta\eta$ -) redexes.

5B7.1 Lemma (i) If a β - or η -contraction in a typed term P^τ changes P^τ to an expression X , then X is also a well-defined typed term with type τ .

(ii) If $P^\tau \triangleright_{\beta\eta} Q^\tau$, then

(a) $\text{Con}(P^\tau) \supseteq \text{Con}(Q^\tau)$,

(b) $P^\tau \triangleright_{\beta\eta} Q^\tau$.

Proof Prove (i) and (ii) together: for β use 5B5.2 and 5B2.1(i), and for η the proof is straightforward. \square

5B7.2 Note (i) When typed terms are interpreted as TA_λ -deductions a β -reduction of a typed term corresponds to a reduction of a deduction. (The idea of reducing deductions originated in Prawitz 1965 and there is a modern account in Troelstra and van Dalen 1988 Vol. 2 Ch. 10 §2; cf. also HS86 Def. 15.30.) In this interpretation the above lemma says that if we reduce a deduction of $\Gamma \mapsto P:\tau$, the result will still be a genuine deduction and will have conclusion $\Gamma^* \mapsto Q:\tau$ for some $\Gamma^* \subseteq \Gamma$ and some Q obtained by β -reducing P .

(ii) Part (ii) of the above lemma is rather like the subject-reduction theorem for TA_λ -deductions (2C1), but is not an exact analogue. A closer analogue is the following lemma which will also be used later.

5B7.3 Lemma (i) Let P^τ be a typed term and $P^\tau \triangleright_{1\beta} Q$ by contracting a β -redex R at a position r in P^τ . Then P^τ contains at position r a typed β -redex R^σ such that $R^\sigma \equiv R$, and contracting this R^σ changes P^τ to a typed term Q^τ such that

$$Q^\tau \equiv Q.$$

(ii) The same holds true for η -redexes.

Proof Same as the proof of 2C1 but with typed terms instead of TA_λ -deductions. \square

5B7.4 Lemma (i) A typed term P^τ has a β -reduction with length n iff P^τ has a β -reduction with length n .

(ii) A typed term P^τ is a β -nf iff P^τ is a β -nf.

(iii) Both (i) and (ii) hold for η .

Proof Note first that if a typed term P^τ contains a β -redex then so does P^τ . Then (i)–(iii) follow from this and 5B7.3. \square

5B8 Typed Church-Rosser Theorem (i) If $M^\tau \triangleright_\beta P^\tau$ and $M^\tau \triangleright_\beta Q^\tau$ then there exists a typed term T^τ such that $P^\tau \triangleright_\beta T^\tau$ and $Q^\tau \triangleright_\beta T^\tau$.

(ii) The same holds for η - and $\beta\eta$ -reductions.

Proof Like the untyped Church-Rosser theorems in 1B and 1C. \square

5B8.1 Warning (No typed convertibility relation) No attempt is made here to define a convertibility relation $=_\beta$ or $=_{\beta\eta}$ on typed terms by expansions and contractions

of redexes. First, we shall not need one. Second, since the typed terms in this chapter correspond exactly to TA_τ -deductions, any attempt to define such a typed equality would meet the same type-variation problems as were discussed for TA_τ in 2C. In fact, β -expanding a typed term may lead to an expression which is no longer a typed term (cf. 2C2.2–6).

5C Normalization theorems

Along with most other type-theories, TA_λ has what is called the *weak normalization* property:

(WN) *every typable term has a normal form.*

And many of them, including TA_λ , also have the *strong normalization* property:

(SN) *all reductions of a typable term are finite.*

These properties were stated for TA_λ in Theorems 2D5 and 2D6 without proof. In the present section a proof will be given of the analogue of WN for typed terms, and WN for typable terms will be deduced from it. No proof of SN will be given here as there is already a detailed proof in our main reference (see HS86 Appendix 2).

5C1 Weak Normalization (WN) Theorem (Turing 1942, Curry and Feys 1958 Cor. 9F9.2, etc.) *Every typed term P^π has both a β -nf and a $\beta\eta$ -nf.*

Proof The existence of a $\beta\eta$ -nf follows from that of a β -nf by the typed analogue of 1C9.4. It remains to prove that P^π has a β -nf, i.e. to give a strategy for reducing P^π and to prove this reduction terminates. This will be done by the method of Turing 1942, the earliest proof known and also the simplest. Some knowledge of 9B will be needed.

Let $P^\pi \in \mathbb{T}(\Gamma)$. Then each β -redex-occurrence \underline{R}^τ in P^π has form

$$\underline{R}^\tau \equiv ((\lambda x^\sigma \cdot M^\tau)^{\sigma \rightarrow \tau} N^\sigma)^\tau.$$

We shall call $(\lambda x \cdot M)^{\sigma \rightarrow \tau}$ the *function-part* of \underline{R}^τ and its type $\sigma \rightarrow \tau$ the *dominant type* of \underline{R}^τ , and the number of atom-occurrences in $\sigma \rightarrow \tau$ the *degree* of \underline{R}^τ :

$$\text{Deg}(\underline{R}^\tau) = |\sigma \rightarrow \tau|.$$

Now let the contraction of \underline{R}^τ change P^π to Q^π , and consider the residuals in Q^π of β -redex-occurrences in P^π . (Residuals are defined in 9B2.) By 9B2.2(iii) the degree of each residual is the same as that of the residual's parent in P^π .

Further, by the conclusion of 9B5.1, for each newly created β -redex-occurrence in Q^π (i.e. one that is not a residual of one in P^π) its function-part is an occurrence of either N^σ or $[N/x]M^\tau$; thus its type is either σ or τ , and so the degree of each newly created redex-occurrence is strictly less than the degree of \underline{R}^τ .

Now let $d(P^\pi)$ be the maximum of the degrees of all β -redex-occurrences in P^π and let \mathbb{S} be the set of all β -redex-occurrences in P^π with this degree. Reduce P^π by a minimal complete development of \mathbb{S} as defined in 9B4. By 9B4 and 9B4.1 this reduction is finite and its result is a term P_1^π containing no residuals of members of

S. All the redexes in P_1^π are either newly created or residuals of redexes not in \mathbb{S} ; hence

$$d(P_1^\pi) < d(P^\pi).$$

Continue this procedure by making further minimal complete developments of sets of redexes with maximum degree; since each development strictly decreases $d(P^\pi)$ we must eventually obtain a β -normal form of P^π . \square

5C1.1 Corollary *The WN theorem for typable terms (2D5) holds.*

Proof By 5B7.4 and the WN theorem for typed terms. \square

5C1.2 Historical notes The first known proof of WN for a type-theory equivalent to TA_λ was written around 1941 or '42 in unpublished notes by Alan Turing. (See Gandy 1980a for a transcript of Turing's notes, and Gandy 1977 pp. 178–180 for some comments on Turing's work in type-theory.)

But the first proof to be actually published was carried out by Curry in the late 1950's by a completely different technique (Curry and Feys 1958 §9F, especially Cor. 9F9.2). Its key step was a cut-elimination theorem for a particular formulation of TA_{CL} .¹

Also there is a WN theorem in Prawitz 1965 for reducing proofs in logic which is essentially equivalent to WN for TA_λ .

From the 1960's onwards the Turing method of proof was re-discovered, probably independently, several times (Morris 1968 §4F Thm. 2, Andrews 1971 Prop. 2.7.3, for example), and WN theorems were proved by various methods for many stronger type-theories than TA_λ .

In particular, the mid-60's saw a spate of proofs of WN for typed λ -calculi enhanced by primitive recursion operators (for example those in Tait 1965, Hanatani 1966, Hinata 1967, Sanchis 1967, Tait 1967, Diller 1968, Dragalin 1968 and Howard 1970); see Troelstra 1973 §§2.2.1–2.3.13 or HS86 Ch. 18 for descriptions of the background setting. At least two of these also included proofs of SN (see below).

5C2 Strong Normalization (SN) Theorem (Sanchis 1967 Thm. 8, Diller 1968 §6, etc.) *Let $P^\pi \in \mathbb{T}(\Gamma)$ for some Γ ; then, for β - and for $\beta\eta$ -reductions,*

- (i) *all reductions of P^π are finite,*
- (ii) *there is an algorithm which accepts P^π as input and outputs a number $k(P^\pi)$ such that all reductions of P^π have length $\leq k(P^\pi)$.*

Proof For (i) there is an accessible proof in HS86 Appendix 2: see Thm. A2.3 for β , and Thm. A2.4 for $\beta\eta$. Alternatively, see Barendregt 1992 Thm. 4.3.6.

For (ii), simply construct a finitely branching tree of reductions starting at P^π by doing all possible contractions at each step. By (i) this tree's branches are all finite

¹ Cut-elimination originated in the study of predicate logic in Gentzen 1935, and the relation between cut-elimination and normalization is explored in Zucker 1974. It depends on the correspondence between formulae and types to be described in the next chapter. Cut-elimination theorems for versions of TA_λ can be found in Seldin 1977 and 1978.

in length, and the famous König's Lemma on trees states that in this case the whole tree must be finite. Hence its branches can all be measured and the maximum of their lengths determined. Call this number $k(P^\pi)$.

But this algorithm is of course inefficient and the proof that $k(P^\pi)$ is well-defined depends on König's Lemma which is usually regarded as non-constructive. To remedy these defects several workers have devised constructive proofs of (ii); see Mints 1979, Gandy 1980b, de Vrijer 1987 and Schwichtenberg 1991. These proofs give more efficient ways of computing suitable bounds $k(P^\pi)$ (though the bounds they give are not always the least possible, see comments in de Vrijer 1987). \square

5C2.1 *Corollary The SN theorem for typable terms (2D6) holds.*

Proof By 5B7.4(i) and 5C2. \square

5C2.2 *Note* The history of SN began over twenty years later than that of WN. The first known explicit SN proofs were in Sanchis 1967 Thm. 8 and Diller 1968 §6, the former for weak reduction in combinatory logic enhanced by primitive recursion operators and the latter for $\lambda\beta$ -reduction similarly enhanced.¹

Most published proofs have depended on defining what is usually known as a *computability predicate* by a suitable induction, and proving first the SN property for all computable terms and then the computability of all terms P^π by induction on $|P^\pi|$. (The proof in HS86 Appendix 2 is typical of this approach, cf. also the WN proof in Tait 1967.) This method has been successfully adapted to so many different type-theories over the years that it is now in effect the standard one. In the early 1970's it was applied by Jean-Yves Girard in a generalised and strengthened form to prove SN for his strong second-order type-theory known as System F (Girard et al. 1989 Ch. 14), and it has since been applied to other second-order type-theories. An analysis with references is in Gallier 1990. Another account is in Barendregt 1992; Barendregt's System $\lambda 2$ is equivalent to Girard's System F.

¹ In each case manuscript versions were made available a couple of years before publication.

The correspondence with implication

One of the most interesting facts about TA_λ is that there is a very close correspondence between this system and propositional logic, in which the types assignable to closed terms in TA_λ turn out to be exactly the formulae provable in a certain formal logic of implication. This correspondence is often called the *Curry-Howard isomorphism* or the *formulae-as-types isomorphism*, and will be studied in this chapter.

The logic involved in this correspondence is not the classical logic of truth-tables however, but that of the intuitionist philosophers; it will be defined in the first section below.

The Curry-Howard isomorphism was first hinted at in print in Curry 1934 p.588, and was made explicit in Curry 1942 p.60 footnote 28 and Curry and Feys 1958 §9E. But it was viewed there as no more than a curiosity. The first people to see that it could be extended to other connectives and quantifiers and used as a technical tool to derive results were N. G. de Bruijn, William Howard and H. Läuchli in the 1960's. See Howard 1969, de Bruijn 1980 (an introduction to de Bruijn's AUTOMATH project which began in the 1960's), and Läuchli 1965.

This chapter will also define three rather interesting subsystems of intuitionist logic and show that they correspond to the three restricted classes of λ -terms defined in Section 1D. This correspondence was first noted by Carew Meredith in unpublished work around 1951 and was explored in detail in the thesis Rezus 1981.

6A Intuitionist implicational logic

In this section we shall define the implicational fragment of intuitionist logic. The original formulations of this logic in the 1920's were axiom-based systems and corresponded more closely to combinatory logic than λ -calculus, but it will be defined here in the more modern "Natural Deduction" style due to Gerhard Gentzen.

There is an elementary introduction to Natural Deduction in van Dalen 1980 §1.5, and more detailed accounts are in Gentzen 1935, Prawitz 1965, and Troelstra and van Dalen 1988 Chs. 2 and 10. But the style is so like the definition of TA_λ in 2A8 that the short introduction below will probably be plenty enough for the reader.¹

¹ This similarity of style is no coincidence; when the earliest formulations of TA_λ were written Gentzen's methods were just beginning to be known more widely; indeed Curry was one of the first to understand their importance.

As mentioned earlier, the formulae provable in the logic defined below will turn out to be exactly the types of closed λ -terms. However, we can also go one step deeper than this and get a correspondence between the proofs of these formulae and the λ -terms themselves. But to make this deeper correspondence work we must be very careful about the definition of “proof” and “deduction” in a Natural Deduction system, and we must clarify some features of such deductions that are not often emphasised in the standard literature on the topic.

Therefore some of the definitions below will involve position-labels and other tedious details that have so far been confined to Chapter 9. These will be needed when determining exactly which deductions correspond to which λ -terms.

6A1 Definition *Implicational formulae* (denoted by lower-case Greek letters other than “ λ ”) are built from *propositional variables* (denoted by “ a ”, “ b ”, “ c ”, ...) using the implication connective “ \rightarrow ” thus: if σ and τ are formulae, then so is $(\sigma \rightarrow \tau)$. (An infinite sequence of propositional variables is assumed to be given.)

6A1.1 Note (i) Of course implicational formulae are exactly the same as types. Parentheses will be omitted using the same association-to-the-right convention as for types, see 2A1.1.

(ii) There are no propositional constants and no other connectives than “ \rightarrow ”. (Because this book is only concerned with the Curry-Howard correspondence in its simplest and smoothest form.)

6A1.2 Background The intuitionist philosophy of mathematics was developed in the early 1900’s by L. E. J. Brouwer and his group in Holland, and a very readable introduction is in Heyting 1955. One feature of this philosophy is that some propositional formulae that are usually accepted as universally valid, and are tautologies in the usual truth-table sense, are regarded by the intuitionists as having only limited validity. One such formula is

$$(PL) \quad ((a \rightarrow b) \rightarrow a) \rightarrow a,$$

which is known as *Peirce’s law*. The logic developed by the intuitionists has attracted continuing interest over the years from many logicians and computer scientists quite independently of their philosophical views. For example many of the polymorphic type-theories in the current literature have intuitionist logic as their basis, not classical logic.

The formulae provable by the rules in the next definition can be shown to coincide exactly with the implicational formulae that the intuitionists accept as universally valid. (See Troelstra and van Dalen 1988, Ch.2 (System IPC) and Ch.10 §5 (separation theorem).)

6A2 Definition *Intuitionist logic* (or more precisely, its implicational fragment) has the following two rules:

$$(\rightarrow E) : \frac{\sigma \rightarrow \tau \quad \sigma}{\tau}, \quad (\rightarrow I) : \frac{[\sigma]}{\sigma \rightarrow \tau.}$$

Each application of $(\rightarrow I)$ is said to *discharge* (or *cancel*) some, all, or none of the occurrences of σ above τ , and must be accompanied by a *discharge label* that lists all the occurrences of σ it discharges (see below for details). If none are discharged the application of $(\rightarrow I)$ is called *vacuous*. Discharged occurrences of σ at the tops of branches must be marked by enclosing them in brackets.¹

Before formally defining deductions and proofs in this system it will be helpful to look at some examples of how the rules are used. In these examples a position-notation like that in 9A1–4 will be included in the deductions to allow the discharge labels to say precisely which occurrences have been discharged. The position of each formula will be written in parentheses beside it, and discharge labels will be written between braces “{ }”.

6A2.1 Example A proof of $(a \rightarrow a \rightarrow c) \rightarrow a \rightarrow c$.

$$\begin{array}{c} \frac{[a \rightarrow a \rightarrow c] \quad (0011) \quad [a] \quad (0012)}{a \rightarrow c \quad (001)} (\rightarrow E) \\ \frac{a \rightarrow c \quad (001) \quad [a] \quad (002)}{a \rightarrow c \quad (00)} (\rightarrow E) \\ \frac{a \rightarrow c \quad (00)}{a \rightarrow c \quad (0)} (\rightarrow I) \quad \{ \text{discharging } a \text{ at } 0012, 002 \} \\ \frac{a \rightarrow c \quad (0)}{(a \rightarrow a \rightarrow c) \rightarrow a \rightarrow a \rightarrow c \quad (\emptyset)} (\rightarrow I) \quad \{ \text{discharging } a \rightarrow a \rightarrow c \text{ at } 0011 \} \end{array}$$

6A2.2 Example A proof of $(a \rightarrow a \rightarrow c) \rightarrow a \rightarrow a \rightarrow c$.

$$\begin{array}{c} \frac{[a \rightarrow a \rightarrow c] \quad (00011) \quad [a] \quad (00012)}{a \rightarrow c \quad (0001)} (\rightarrow E) \\ \frac{a \rightarrow c \quad (0001) \quad [a] \quad (0002)}{a \rightarrow c \quad (000)} (\rightarrow E) \\ \frac{a \rightarrow c \quad (000)}{a \rightarrow c \quad (00)} (\rightarrow I) \quad \{ \text{discharging } a \text{ at position } 00012 \} \\ \frac{a \rightarrow c \quad (00)}{(a \rightarrow a \rightarrow c) \quad (0)} (\rightarrow I) \quad \{ \text{discharging } a \text{ at } 0002 \} \\ \frac{(a \rightarrow a \rightarrow c) \quad (0)}{(a \rightarrow a \rightarrow c) \rightarrow a \rightarrow a \rightarrow c \quad (\emptyset)} (\rightarrow I) \quad \{ \text{discharging } a \rightarrow a \rightarrow c \text{ at } 00011 \} \end{array}$$

6A3 Discussion (Partial discharging) The proof in 6A2.2 contains an example of what will be called *partial discharging*. Suppose rule $(\rightarrow I)$ in 6A2 is applied to a deduction of τ in which σ has $n \geq 1$ occurrences $\underline{\sigma}_1, \dots, \underline{\sigma}_n$ as assumptions. Definition

¹ The system defined above is often called *minimal logic*. But we shall later be interested in subsystems of this system so “minimal” seems inappropriate here.

6A2 allows $(\rightarrow I)$ to leave some of $\sigma_1, \dots, \sigma_n$ undischarged. For example in 6A2.2 the uppermost application of $(\rightarrow I)$ has two occurrences of a as assumptions above it but only discharges one.

It is standard practice to allow this in a Natural Deduction system, although few introductory accounts of Natural Deduction emphasise that this is so. Here we shall say that a system which allows it obeys the *partial discharge convention*. This convention is discussed in Prawitz 1965 Ch. I §4, Leivant 1979, and Troelstra and van Dalen 1988 Vol. 2, pp.559–560 and p.568 Ex. 10.8.1. (In the extreme, it allows us even to make a vacuous discharge of σ when $n \geq 1$; this would deduce $\sigma \rightarrow \tau$ and leave all of $\sigma_1, \dots, \sigma_n$ undischarged! An example of this extreme is in 6A3.3.)

In any system obeying this convention each occurrence of $(\rightarrow I)$ in a deduction must carry a label listing all the occurrences of σ that it discharges. These discharge-labels play so little role in standard accounts of natural deduction that they are almost always omitted, but when we come to make a correspondence between proofs and λ -terms they will be essential because *proofs with different discharge-labels will correspond to different terms*. We shall see this in 6B.

By the way, partial discharging is forbidden in TA_λ . In fact $(\rightarrow I)$ in 2A8 has an effect equivalent to discharging every occurrence of $x:\sigma$ in Γ that has not been discharged earlier; we shall say TA_λ obeys the *complete discharge convention*.

6A3.1 *Example* The following is a proof of $(a \rightarrow a \rightarrow c) \rightarrow a \rightarrow a \rightarrow c$ that is identical to that in 6A2.2 except for having different discharge-labels.

$$\begin{array}{c}
 \frac{[a \rightarrow a \rightarrow c] \quad (00011) \quad [a] \quad (00012)}{a \rightarrow c \quad (0001)} (\rightarrow E) \quad \frac{[a] \quad (0002)}{[a] \quad (0002)} (\rightarrow E) \\
 \frac{a \rightarrow c \quad (0001) \quad [a] \quad (0002)}{c \quad (000)} (\rightarrow I) \\
 \frac{c \quad (000)}{a \rightarrow c \quad (00)} (\rightarrow I) \quad \{ \text{discharging } a \text{ at } 00012, 0002 \} \\
 \frac{a \rightarrow c \quad (00)}{a \rightarrow a \rightarrow c \quad (0)} (\rightarrow I) \quad \{ \text{discharging } a \text{ vacuously} \} \\
 \frac{a \rightarrow a \rightarrow c \quad (0)}{(a \rightarrow a \rightarrow c) \rightarrow a \rightarrow a \rightarrow c \quad (\emptyset)} (\rightarrow I) \quad \{ \text{discharging } a \rightarrow a \rightarrow c \text{ at } 00011 \}
 \end{array}$$

6A3.2 *Note* Comparing 6A2.2 with 6A3.1 might suggest that a succession of partial discharges of the same formula can always be replaced by one complete discharge followed by some vacuous ones. And this suggestion is correct. (Though it would fail in restricted logics that forbid vacuous discharging.) Thus if partial discharging were forbidden the class of all provable formulae would not be reduced. However, the class of proofs would then become more restricted than the class of λ -terms.

6A3.3 *Example* The following proof illustrates an earlier remark that the partial discharge convention allows vacuous discharging of σ even when the set of available

occurrences of σ is not empty.

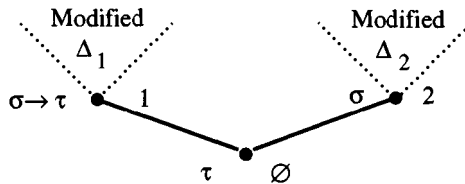
$$\frac{\frac{[a] \quad (00)}{a \rightarrow a \quad (0)} (\rightarrow I) \quad \{\text{discharging } a \text{ vacuously}\}}{a \rightarrow a \rightarrow a \quad (\emptyset)} (\rightarrow I) \quad \{\text{discharging } a \text{ at position } 00\}$$

Hopefully this and the earlier examples have given the reader some feeling for the concepts involved in a natural deduction. The formal definition of “natural deduction” will now be given for precision’s sake, though the reader may omit it without much loss. In it, a natural deduction will be a tree in which each node will carry either two or three labels: (1) a formula, (2) a position, and (3) either a discharge-label (if the node represents the conclusion of $(\rightarrow I)$) or a pair of brackets (if the node represents a discharged assumption), or no third label.

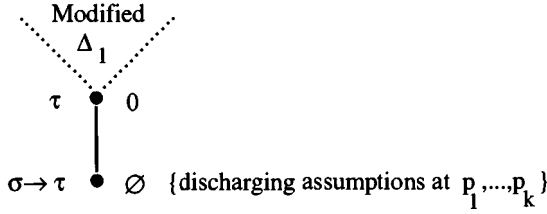
6A4 Definition *Natural deductions in the Intuitionist logic of implication* (or *logic deductions* for short) are trees, defined thus.

(i) An *atomic* deduction is a single node with two labels τ, \emptyset , where τ is a formula and \emptyset is the empty position.

(ii) If Δ_1 and Δ_2 are natural deductions whose bottom nodes are labelled respectively by $\sigma \rightarrow \tau, \emptyset$ and σ, \emptyset (and possibly a third label each), define a new deduction by first putting “1” on the left end of each position-label in Δ_1 and “2” on the left end of each position-label in Δ_2 , next doing the same to the positions in any discharge-labels in Δ_1 and Δ_2 , and finally placing an extra node beneath the two modified deductions, as shown below.



(iii) If Δ_1 is a natural deduction whose bottom node is labelled by τ, \emptyset (and possibly a third label), and σ is any formula, define a new deduction thus. First choose a (possibly empty) set of unbracketed occurrences of σ at the tops of branches in Δ_1 and label them with brackets, then put “0” on the left end of every position in Δ_1 , then place an extra node beneath the modified Δ_1 as shown in the diagram below, and finally give this node three labels: (1) $\sigma \rightarrow \tau$, (2) \emptyset , and (3) a *discharge-label* listing the positions of the chosen occurrences of σ . (If the set of chosen occurrences is empty we make the discharge-label say “*vacuous*”.)



6A4.1 Notation In practice natural deductions are never displayed in the space-wasting form shown in these diagrams, but in the horizontal-line form used in the examples earlier.

6A4.2 Definition Two occurrences of an assumption at branch-tops in a natural deduction Δ will be called *discharge-equivalent* iff they are discharged by the same occurrence of $(\rightarrow I)$, i.e. iff their names appear on the same non-vacuous discharge-label in Δ .

6A5 Definition (Deducibility and provability) Iff there is a deduction whose conclusion is τ and whose undischarged assumptions are occurrences of members of $\{\sigma_1, \dots, \sigma_n\}$, we say

$$\sigma_1, \dots, \sigma_n \vdash_{\text{int}} \tau.$$

(This notation may be used when $\sigma_1, \dots, \sigma_n$ are not distinct.) Iff $n = 0$ the deduction is called a *proof* and τ a *provable formula* or *theorem*, and we say

$$\vdash_{\text{int}} \tau.$$

6B The Curry-Howard isomorphism

The similarity between rules $(\rightarrow E)$ and $(\rightarrow I)$ in the previous section and the TA_λ -rules $(\rightarrow E)$ and $(\rightarrow I)$ in Chapter 2 suggests that every TA_λ -deduction might be changed into a logic deduction by stripping off all the contexts and terms in it. This is in fact true, modulo a few minor points like differences in the discharge conventions. The details are as follows.

6B1 Definition (Curry-Howard mapping from lambda to logic) If Δ is a TA_λ -deduction of $\Gamma \mapsto M : \tau$ (see 2A8), the *corresponding logic deduction* Δ_L is defined thus.

- (i) If $M \equiv x$ and Δ is $x : \tau \mapsto x : \tau$, choose Δ_L to be just τ .
- (ii) If $M \equiv PQ$ and $\Gamma = \Gamma_1 \cup \Gamma_2$ and the last step in Δ has form

$$\frac{\begin{array}{cc} (\Delta_1) & (\Delta_2) \\ \Gamma' \mapsto P : \sigma \rightarrow \tau & \Gamma_2 \mapsto Q : \sigma \end{array}}{\Gamma_1 \cup \Gamma_2 \mapsto (PQ) : \tau} (\rightarrow E)$$

construct Δ_L by applying rule $(\rightarrow E)$ of 6A2 to Δ_{1L} and Δ_{2L} .

(iii) If $M \equiv \lambda x.P$, $\tau \equiv \rho \rightarrow \sigma$, $\Gamma = \Gamma' - x$ and the last step in Δ is

$$\begin{array}{c}
 (\Delta') \\
 \hline
 \Gamma' \mapsto P:\sigma \\
 \hline
 \Gamma' - x \mapsto (\lambda x.P):(\rho \rightarrow \sigma),
 \end{array}
 \quad (\rightarrow I)$$

construct Δ_L by applying rule $(\rightarrow I)$ of 6A2 to Δ'_L and discharging all occurrences of ρ in Δ'_L whose positions are the same as the positions of the free occurrences of x in P .

6B1.1 Example If Δ_1 is the following TA_λ -deduction, $(\Delta_1)_L$ is the deduction in 6A2.2.

$$\begin{array}{c}
 x:a \rightarrow a \rightarrow c \mapsto x:a \rightarrow a \rightarrow c \quad z:a \mapsto z:a \\
 \hline
 x:a \rightarrow a \rightarrow c, z:a \mapsto (xz):a \rightarrow c \quad y:a \mapsto y:a \\
 \hline
 x:a \rightarrow a \rightarrow c, y:a, z:a \mapsto (xzy):c \quad (\rightarrow I) \quad \{\text{discharging } z\} \\
 x:a \rightarrow a \rightarrow c, y:a \mapsto (\lambda z \cdot xzy):a \rightarrow c \quad (\rightarrow I) \quad \{\text{discharging } y\} \\
 x:a \rightarrow a \rightarrow c \mapsto (\lambda yz \cdot xzy):a \rightarrow a \rightarrow c \quad (\rightarrow I) \quad \{\text{discharging } x\} \\
 \hline
 \mapsto (\lambda xyz \cdot xzy):(a \rightarrow a \rightarrow c) \rightarrow a \rightarrow a \rightarrow c
 \end{array}$$

6B1.2 Example If Δ_2 is the TA_λ -deduction below, $(\Delta_2)_L$ is the deduction in 6A3.1.

$$\begin{array}{c}
 x:a \rightarrow a \rightarrow c \mapsto x:a \rightarrow a \rightarrow c \quad z:a \mapsto z:a \\
 \hline
 x:a \rightarrow a \rightarrow c, z:a \mapsto (xz):a \rightarrow c \quad z:a \mapsto z:a \\
 \hline
 x:a \rightarrow a \rightarrow c, z:a \mapsto (xzz):c \quad (\rightarrow I) \quad \{\text{discharging } z\} \\
 x:a \rightarrow a \rightarrow c \mapsto (\lambda z \cdot xzz):a \rightarrow c \quad (\rightarrow I) \quad \{\text{vac. dis. } y:a\} \\
 x:a \rightarrow a \rightarrow c \mapsto (\lambda yz \cdot xzz):a \rightarrow a \rightarrow c \quad (\rightarrow I) \quad \{\text{discharging } x\} \\
 \hline
 \mapsto (\lambda xyz \cdot xzz):(a \rightarrow a \rightarrow c) \rightarrow a \rightarrow a \rightarrow c
 \end{array}$$

6B1.3 Note The λ -terms to which Δ_1 and Δ_2 assign types in the two above examples are distinct, although they have the same tree-structure. Correspondingly $(\Delta_1)_L$ and $(\Delta_2)_L$ are distinct, though they only differ in their discharge-labels.

6B2 Lambda-to-Logic Lemma Let x_1, \dots, x_n be distinct and Δ a TA_λ -deduction of

$$x_1:\rho_1, \dots, x_n:\rho_n \mapsto M:\tau.$$

Then Δ_L is a natural deduction in Intuitionist logic and gives

$$\rho_1, \dots, \rho_n \vdash \tau.$$

Proof Induction on $|M|$. □

6B2.1 Exercise By applying the Curry-Howard mapping to the TA_λ -proofs in 2A8.2–4 and the answers to 2A8.7, show that the following formulae are provable in Intuitionist logic. (Their names below date from the 1950's and are now standard in the logic literature; of course they are not accidental.)

- (B) $(a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b$,
- (B') $(a \rightarrow b) \rightarrow (b \rightarrow c) \rightarrow a \rightarrow c$,
- (I) $a \rightarrow a$,
- (K) $a \rightarrow b \rightarrow a$,
- (C) $(a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c$,
- (W) $(a \rightarrow a \rightarrow b) \rightarrow a \rightarrow b$.

6B3 Warning The Curry-Howard mapping is not one-to-one. For example consider the following TA_λ -deductions Δ_3 and Δ_4 which assign types to xyz and xyy .

$$\begin{array}{c} \Delta_3: \frac{x:a \rightarrow a \rightarrow c \mapsto x:a \rightarrow a \rightarrow c \quad y:a \mapsto y:a}{x:a \rightarrow a \rightarrow c \mapsto x:a \rightarrow a \rightarrow c \quad y:a \mapsto y:a} (\rightarrow E) \\ \frac{x:a \rightarrow a \rightarrow c, y:a \mapsto xy:a \rightarrow c \quad z:a \mapsto z:a}{x:a \rightarrow a \rightarrow c, y:a, z:a \mapsto xyz:c} (\rightarrow E) \end{array}$$

$$\begin{array}{c} \Delta_4: \frac{x:a \rightarrow a \rightarrow c \mapsto x:a \rightarrow a \rightarrow c \quad y:a \mapsto y:a}{x:a \rightarrow a \rightarrow c \mapsto x:a \rightarrow a \rightarrow c \quad y:a \mapsto y:a} (\rightarrow E) \\ \frac{x:a \rightarrow a \rightarrow c, y:a \mapsto xy:a \rightarrow c \quad y:a \mapsto y:a}{x:a \rightarrow a \rightarrow c, y:a \mapsto xyy:c} (\rightarrow E) \end{array}$$

It is easy to check that the Curry-Howard mapping assigns the same logic deduction to both, namely

$$\frac{\frac{a \rightarrow a \rightarrow c \quad a}{a \rightarrow c} (\rightarrow E)}{c} (\rightarrow E)$$

(More generally, if Δ is a TA_λ -deduction of $\Gamma \mapsto M:\tau$ and x and y receive the same type in Γ and we change Δ to a deduction of

$$[v/x, v/y]\Gamma \mapsto [v/x, v/y]M:\tau,$$

then Δ_L will not change.)

Since the Curry-Howard mapping is not one-to-one it cannot have an inverse, i.e. there is no logic-to-lambda mapping $(\)_\lambda$ such that

- (1) $(\Delta_\lambda)_L \equiv \Delta$ (for all logic deductions Δ),
- (2) $(\Delta_L)_\lambda \equiv \Delta$ (for all TA_λ -deductions Δ).

However there is a “one-sided inverse”, a mapping $(\)_\lambda$ satisfying (1) alone. In fact there are several such inverses, and the next definition will describe one of them; it will be obtained by choosing term-variables to be distinct whenever possible.

6B4 Definition (A logic-to-lambda mapping) To each natural deduction Δ in Intuitionist logic we assign a TA_λ -deduction Δ_λ thus.

(i) If Δ is a single formula τ , choose any term-variable x and let Δ_λ be

$$x:\tau \mapsto x:\tau.$$

(ii) If the last step in Δ is $(\rightarrow E)$ applied to the conclusions of deductions Δ' and Δ'' , and Δ'_λ and Δ''_λ have been defined and are deductions of

$$\Gamma' \mapsto M:\sigma \rightarrow \tau, \quad \Gamma'' \mapsto N:\sigma,$$

then replace all term-variables in Δ''_λ by distinct new ones (to make it have no term-variables in common with Δ'_λ , neither free nor bound), and apply the TA_λ -rule $(\rightarrow E)$; call the resulting deduction Δ_λ .

(iii) If the last step in Δ is an occurrence of $(\rightarrow I)$ with form

$$\frac{\left. \begin{array}{c} [\rho] \\ : \\ \sigma \end{array} \right\} \text{ a deduction } \Delta'}{\rho \rightarrow \sigma} \quad \{\text{discharging } k \geq 0 \text{ occurrences } \underline{\rho}_1, \dots, \underline{\rho}_k \text{ of } \rho\},$$

and Δ'_λ has been defined and its conclusion is

$$\Gamma, v_1:\rho, \dots, v_k:\rho \mapsto P:\sigma$$

where v_1, \dots, v_k are distinct from each other and from the variables in Γ , and v_1, \dots, v_k occur free in P at the same positions as $\underline{\rho}_1, \dots, \underline{\rho}_k$ have in Δ' (each v_i occurring only once in P), then proceed as follows.

If $k \geq 1$, replace all of v_1, \dots, v_k in Δ'_λ by one new variable x that does not occur elsewhere in Δ'_λ ; this changes Δ'_λ to a deduction of

$$\Gamma, x:\rho \mapsto P^*:\sigma \quad (P^* \equiv [x/v_1, \dots, x/v_k]P)$$

where x occurs in P^* at exactly the same positions as $\underline{\rho}_1, \dots, \underline{\rho}_k$ have in Δ' . Then apply $(\rightarrow I)_{\text{main}}$ to this modified Δ'_λ and call the resulting deduction Δ_λ . Its conclusion will be

$$\Gamma \mapsto (\lambda x.P^*):\rho \rightarrow \sigma$$

If $k = 0$, the conclusion of Δ'_λ must be simply $\Gamma \mapsto P:\sigma$; choose a new variable x not in Δ'_λ and apply $(\rightarrow I)_{\text{vac}}$ to deduce

$$\Gamma \mapsto (\lambda x.P):\rho \rightarrow \sigma$$

6B4.1 Note In the above definition clause (ii) makes as many variables as possible distinct, and then (iii) identifies some of these again. Lemma 6B5 will show that Δ_λ is well-defined for every Δ .

6B4.2 Examples (i) If Δ is the following logic deduction, then Δ_λ is the TA_λ -deduction Δ_3 in 6B3 (modulo replacements of distinct term-variables by distinct

term-variables).

$$\frac{\frac{a \rightarrow a \rightarrow c \quad a}{(\rightarrow E)} \quad a}{\frac{a \rightarrow c \quad a}{(\rightarrow E)} c}$$

(ii) If Δ is the logic deduction in 6A2.2, Δ_λ is the TA_λ -deduction in 6B1.1 (modulo \equiv_α).

(iii) If Δ is the logic deduction in 6A3.1, Δ_λ is the TA_λ -deduction in 6B1.2 (modulo \equiv_α).

6B4.3 Exercise* Construct and compare $(\Delta_5)_\lambda$ and $(\Delta_6)_\lambda$, where Δ_5 is the deduction in 6A3.3 and Δ_6 is the following deduction.

$$\frac{\frac{[a] \quad (00)}{a \rightarrow a \quad (0)} (\rightarrow I) \quad \{ \text{discharging } a \text{ at position } 00 \}}{a \rightarrow a \rightarrow a \quad (\emptyset)} (\rightarrow I) \quad \{ \text{discharging } a \text{ vacuously} \}$$

6B5 Logic-to-Lambda Lemma Let $\sigma_1, \dots, \sigma_n \vdash \tau$ by a logic deduction Δ , and for each $i \leq n$ let

$$\underline{\sigma}_{i,1}, \dots, \underline{\sigma}_{i,m_i}$$

be those occurrences of σ_i that are undischarged assumptions at branch-tops in Δ . Then Δ_λ is defined and is a TA_λ -deduction whose conclusion has form

$$x_{1,1}:\sigma_1, \dots, x_{1,m_1}:\sigma_1, \dots, x_{n,1}:\sigma_n, \dots, x_{n,m_n}:\sigma_n \mapsto M:\tau$$

where each $x_{i,j}$ occurs exactly once in M (at the same position as $\underline{\sigma}_{i,j}$ has in Δ) and M has no bound-variable clashes. Also

$$(\Delta_\lambda)_L \equiv \Delta.$$

Proof Straightforward induction on the number of steps in Δ . The change of variables in 6B4(ii) ensures that M has no clashes of bound variables and has only one occurrence of each of its free variables. \square

6B5.1 Note The Curry-Howard mapping from Δ to Δ_L is usually called the Curry-Howard *isomorphism*: how far can it justify this description? There are three levels to consider:

- (i) provable formulae \leftrightarrow types of closed terms,
- (ii) logic proofs $\leftrightarrow \text{TA}_\lambda$ -proofs,
- (iii) logic deductions $\leftrightarrow \text{TA}_\lambda$ -deductions.

By 6B5, the Δ_L -mapping has a one-sided inverse Δ_λ such that $(\Delta_\lambda)_L \equiv \Delta$, but for it to be an isomorphism in any real sense we should also have $(\Delta_L)_\lambda \equiv \Delta$. We have seen in 6B3 that this breaks down on level (iii). But on levels (i) and (ii) it holds true, as the next theorem will show.

But before that theorem a lemma will describe exactly how far the identity $(\Delta_L)_\lambda \equiv \Delta$ breaks down.

6B6 Lemma *Let Δ be a TA_λ -deduction of $\Gamma \mapsto P:\tau$. Then $(\Delta_L)_\lambda$ differs from Δ only in that some term-variables distinct in $(\Delta_L)_\lambda$ may be identical in Δ . In more detail:*

(i) $(\Delta_L)_\lambda$ is a TA_λ -deduction of a formula with form

$$\Gamma' \mapsto M:\tau$$

where, just as in 6B5, M has no bound-variable clashes, $FV(M) = \{x_1, \dots, x_n\}$, and each x_i occurs just once in M ; also, for some v_1, \dots, v_n not necessarily distinct,

$$\begin{aligned} P &\equiv_\alpha [v_1/x_1] \dots [v_n/x_n] M, \\ \Delta &\equiv_\alpha [v_1/x_1] \dots [v_n/x_n] (\Delta_L)_\lambda; \end{aligned}$$

(ii) if Δ is a proof of $\mapsto P:\tau$ then $(\Delta_L)_\lambda$ is a proof of $\mapsto P':\tau$ for some $P' \equiv_\alpha P$.

Proof Part (i) is proved by induction on $|M|$ with three cases following the clauses of 6B1. (The notation “ $[v_1/x_1] \dots [v_n/x_n] (\Delta_L)_\lambda$ ” means the deduction obtained by doing the substitutions $[v_1/x_1], \dots, [v_n/x_n]$ in all subjects in $(\Delta_L)_\lambda$.)

Part (ii) is a special case of (i). □

6B7 Curry-Howard Theorem (i) *The provable formulae of Intuitionist implicational logic are exactly the types of closed λ -terms.*

(ii) *The relation $\sigma_1, \dots, \sigma_n \vdash \tau$ holds in Intuitionist logic iff there exist M and x_1, \dots, x_n (distinct) such that*

$$x_1:\sigma_1, \dots, x_n:\sigma_n \vdash_\lambda M:\tau.$$

(iii) *The Δ -to- Δ_L mapping is a one-to-one correspondence between TA_λ -proofs and proofs in the Natural Deduction version of Intuitionist Implicational logic, and the Δ -to- Δ_λ mapping is its inverse (modulo \equiv_α). That is, for all TA_λ -proofs Δ ,*

$$(\Delta_L)_\lambda \equiv \Delta$$

(modulo \equiv_α in subjects in $(\Delta_L)_\lambda$), and for all logic proofs Δ ,

$$(\Delta_\lambda)_L \equiv \Delta.$$

Proof For (i) and (iii) use 6B2 and 6B6. For (ii) “if”, use 6B2.

For (ii) “only if”, let $\sigma_1, \dots, \sigma_n \vdash \tau$ be obtained by a logic deduction Δ , and apply rule $(\rightarrow I)$ n times to change Δ into a proof, Δ^* , of

$$\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow \tau.$$

Then construct $(\Delta^*)_\lambda$. By 6B5 its conclusion will be

$$\mapsto N:(\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow \tau)$$

for some closed λ -term N with no bound-variable clashes. Hence by the subject-construction theorem, N must have form $\lambda x_1 \dots x_n.M$ for some M and some distinct x_1, \dots, x_n , and $(\Delta^*)_\lambda$ must contain the formula

$$x_1:\sigma_1, \dots, x_n:\sigma_n \vdash_\lambda M:\tau.$$

(By the way, M has no bound-variable clashes because N has none.) □

6B7.1 Note The one-to-one correspondence in 5A7 between TA_λ -deductions and the typed terms of Chapter 5 can be combined with the above mappings to give a one-to-one correspondence between closed typed terms and logic proofs.

6B7.2 Corollary *To decide whether an implicational formula τ is provable in Intuitionist logic it is enough to decide whether there exists a closed term M such that $\vdash_\lambda M:\tau$, or equivalently, whether there exists a closed typed term with type τ .*

6B7.3 Note An algorithm to decide whether there exists a closed typed term M^τ will be described in 8D5.2. Via the above corollary this will give a decision-algorithm for Intuitionist implicational logic. Of course decision-procedures for this logic have been known for a long time,¹ but the one in 8D5.2 will also count the number of closed terms M^τ in β -normal form and will generate them one by one. This can be regarded as counting and generating all the irreducible Natural Deduction proofs of τ if we think of each M^τ as representing a proof via the Curry-Howard mapping.

6B7.4 Corollary *Peirce's Law, the formula $((a \rightarrow b) \rightarrow a) \rightarrow a$ introduced in 6A1.2, is not provable in Intuitionist logic.*

Proof By 8B6, Peirce's Law is not the type of a closed normal form; hence by the SN theorem it cannot be the type of any closed term. □

6B8 Note The Curry-Howard theorem has shown that the Intuitionist logic of pure implication corresponds in a very neat and natural way to the pure λ -calculus. Over the years this correspondence has been extended to more expressive logics and even to quite strong mathematical systems, and this has led to some useful techniques in proof theory, for example methods for extracting programs from proofs of existence. (See Crossley and Shepherdson 1993 for a modern account.)

But although each of these extensions is very important from a practical point of view, few of them have the directness and neatness of the original “core” system.

6C Some weaker logics

At the end of Chapter 1 three restricted classes of λ -terms were defined: the λI -terms (roughly speaking, terms without vacuous binding), the $\text{BCK}\lambda$ -terms (terms in which

¹ For example an old one is in Gentzen 1935 pp.405–407 (pp.83–105 of the English translation). A more efficient one is in Dyckhoff 1992 p.800 (the note after the proof of Thm. 1); it has been implemented in the University of St Andrews proof-discovery program MacLogic. Another one is in Hudelmaier 1993 along with a proof that it operates in $O(n \log n)$ -space. And one involving an explicit translation into λ -calculus is in Zaionc 1988.

each subterm has at most one non-binding occurrence of each variable), and the $\text{BCI}\lambda$ -terms (those that are both $\text{BCK}\lambda$ - and λI -terms).

Suppose we restrict TA_λ by looking only at terms in one of these three classes; does the corresponding logic obtained via the Curry-Howard mapping have any interesting properties? The answer turns out to be “yes” for each class, as we shall see in this section.¹

6C1 Definition (The relevance logic R_\rightarrow) The definition of R_\rightarrow is exactly like that of Intuitionist implicational logic in 6A2 except that vacuous discharging is not allowed. That is, when $\sigma \rightarrow \tau$ is the conclusion of rule $(\rightarrow\text{I})$ its discharge-label must show at least one occurrence of σ .

6C1.1 Note (Motivation for R_\rightarrow) In one important view of implication, a formula $\sigma \rightarrow \tau$ should not be provable unless σ is in some way relevant to τ . In this view the formula

$$a \rightarrow b \rightarrow a$$

is not universally valid, because it says in essence that if a statement a is true then every other statement b implies it, even when b has no connection with the meaning of a . There have been many different attempts to capture the notion of relevant implication by formal systems and R_\rightarrow is one of the earliest and simplest. In it, roughly speaking, we can only prove $\sigma \rightarrow \tau$ when σ has actually been used in the deduction of τ .

More about R_\rightarrow can be found in Anderson and Belnap 1975, especially §§3, 7, 8.3, 8.4, 8.18 and 9–14, and in Anderson et al. 1992, especially §§47.1, 63.2 and 71.2. The system has been invented independently at several different times; see Moh 1950, Church 1951 and the discussion in Dosen 1992b. It is known to be decidable (i.e. there is an algorithm to decide whether a given formula is provable in R_\rightarrow); see Anderson and Belnap 1975 §13 and Anderson et al. 1992 §63.²

6C1.2 Examples Of the proofs in the examples in 6A, those in 6A2.1 and 6A2.2 are correct R_\rightarrow -proofs. In particular, by 6A2.2, the formula

$$(a \rightarrow a \rightarrow c) \rightarrow a \rightarrow a \rightarrow c$$

is provable in R_\rightarrow . In contrast the proof of the same formula in 6A3.1 contains a vacuous discharge and hence is not an R_\rightarrow -proof. Nor is the proof of $a \rightarrow a \rightarrow a$ in 6A3.3.

Of the proofs obtained in Exercise 6B2.1, those of (B), (B'), (I), (C), and (W) are easily seen to be R_\rightarrow -proofs, but that of (K) is not.

6C2 Definition (BCK-logic) The system that will be called here *BCK-logic* (or more precisely, the implicational fragment of BCK-logic) is defined exactly like Intuitionist

¹ The logics defined in the present section and some others like them have come to be known as *substructural logics*.

² Though in the literature R_\rightarrow is often presented as the implicational fragment of a system R which has other connectives besides “ \rightarrow ” and has been proved undecidable (Urquhart 1984 or Anderson et al. 1992 §65).

logic in 6A2, except that multiple discharging is not allowed. That is, when $(\rightarrow I)$ is used its discharge-label must either be vacuous or contain only one occurrence of σ .

6C2.1 Note Roughly speaking, BCK-logic is a system in which an assumption cannot be used more than once; it is a logic of non-re-usable information. Interest in such logics dates back at least as far as Fitch 1936, and results on BCK logic can be found in a number of sources, for example Meredith and Prior 1963 §§7–8, Jaskowski 1963, Iséki and Tanaka 1978, Bunder 1986, 1991, Komori 1987, 1989, Blok and Pigozzi 1989, Hindley 1989, 1993, Dosen 1992a, Hirokawa 1991c, 1992b, 1993a–b, Komori and Hirokawa 1993, and those mentioned in their bibliographies. The reason for the name “BCK” will be made clear in the next section.

6C2.2 Examples Of the proofs constructed in Exercise 6B2.1, it is easy to see that those of (B), (B'), (I), (K) and (C) are correct BCK-proofs.

But the proof of (W) in 6A2.1 is not, because it contains a multiple discharge.

Of the two proofs of $(a \rightarrow a \rightarrow c) \rightarrow a \rightarrow a \rightarrow c$ in 6A2.2 and 6A3.1, the former is a BCK-proof but the latter is not.

The proof of $a \rightarrow a \rightarrow a$ in 6A3.3 is a BCK-proof.

6C3 Definition (BCI-logic) The definition of *BCI-logic* (to be precise, its implicational fragment) is like 6A2 except that both vacuous and multiple discharging are forbidden; i.e., when rule $(\rightarrow I)$ is used its discharge-label must contain exactly one occurrence of σ .

6C3.1 Note BCI-logic is a relevance logic of non-re-usable information. Most work on this system has been done in parallel with work on BCK-logic and the majority of the above references for BCK-logic treat both systems. Other references are, for example, Hirokawa 1991b, Avron 1992.

Incidentally, the BCI system above coincides with the implicational fragment of the *linear logic* of Girard 1987. Although the latter is a much deeper theory than any logic confined to a single connective and has been studied in many publications, a few of these publications give its implicational fragment some separate attention; an example is Avron 1988.

6C3.2 Examples Of the proofs constructed in Exercise 6B2.1, those of (B), (B'), (I), and (C) are correct BCI-proofs.

But the proofs of (K) and (W) are not.

Of the two proofs of $(a \rightarrow a \rightarrow c) \rightarrow a \rightarrow a \rightarrow c$ in 6A2.2 and 6A3.1, the former is a BCI-proof but the latter is not.

The proof of $a \rightarrow a \rightarrow a$ in 6A3.3 is not a BCI-proof.

6C4 Lemma *The sets of provable formulae of the four logics defined in 6B–C are related as follows:*

- (i) $BCI \subseteq BCK \subseteq \text{Intuitionist}$,
- (ii) $BCI \subseteq R_{\rightarrow} \subseteq \text{Intuitionist}$.

6C5 Refined Curry-Howard Theorem (i) *The provable formulae of the three logics defined in 6C1–3 are exactly the types of the following λ -terms:*

- R_{\rightarrow} : types of the closed λI -terms;
 BCK -logic : types of the closed $BCK\lambda$ -terms;
 BCI -logic : types of the closed $BCI\lambda$ -terms.

(ii) *The relation $\sigma_1, \dots, \sigma_n \vdash \tau$ holds in R_{\rightarrow} , BCK - or BCI -logic iff there exist M and x_1, \dots, x_n (distinct) such that*

$$x_1:\sigma_1, \dots, x_n:\sigma_n \vdash_{\lambda} M:\tau$$

and M is, respectively, a λI -term, $BCK\lambda$ -term or $BCI\lambda$ -term.

Proof By Lemmas 6C5.1 and 6C5.2 below. □

6C5.1 Lemma *Let Δ be a TA_{λ} -deduction of $\Gamma \mapsto M:\tau$ and let Δ_L be defined as in 6B1. Then*

- (i) *if M is a λI -term, Δ_L is an R_{\rightarrow} -deduction,*
- (ii) *if M is a $BCK\lambda$ -term, Δ_L is a BCK -deduction,*
- (iii) *if M is a $BCI\lambda$ -term, Δ_L is a BCI -deduction.*

Proof Straightforward induction on $|M|$. □

6C5.2 Lemma *Let Δ be an R_{\rightarrow} -, BCK - or BCI -deduction with Δ_{λ} defined as in 6B4, and let the conclusion of Δ_{λ} be $\Gamma \mapsto M:\tau$. Then*

- (i) *if Δ is an R_{\rightarrow} -deduction, M is a λI -term,*
- (ii) *if Δ is a BCK -deduction, M is a $BCK\lambda$ -term,*
- (iii) *if Δ is a BCI -deduction, M is a $BCI\lambda$ -term.*

Proof Straightforward induction on the number of steps in Δ . □

6D Axiom-based versions

As a matter of historical fact the four logics discussed in this chapter all made their first appearance not in the Natural Deduction form given above but in an axiom-based version. These axiom-based versions will play a role in the next chapter, so as a preparation they will be defined and discussed in this section.

But before defining the four logics separately some of the properties of axiom-based systems in general will be described, relative to an arbitrary set \mathbb{A} of axioms. Axiomatic systems are often called *Hilbert-style* systems to contrast with Natural Deduction.

6D1 Definition (\mathbb{A} -logics) Let \mathbb{A} be any set of implicational formulae that are tautologies in the classical truth-table sense. Then \mathbb{A} generates the following Hilbert-style system, which will be called the corresponding \mathbb{A} -logic.

Axioms: the members of \mathbb{A} .

Deduction-rules:

$$(\rightarrow E) \quad \frac{\sigma \rightarrow \tau \quad \sigma}{\tau}$$

$$(\text{Sub}) \quad \frac{\tau}{s(\tau)} \quad \text{[if } s \text{ is a substitution and no variable in } \text{Dom}(s) \text{ occurs in a non-axiom assumption in the deduction above the line]}$$

Deductions in an \mathbb{A} -logic are trees, with axioms and assumptions at the tops of branches and the conclusion at the bottom of the tree. The notation

$$\sigma_1, \dots, \sigma_n \vdash_{\mathbb{A}} \tau$$

means that there is a deduction whose non-axiom assumptions are some or all of $\sigma_1, \dots, \sigma_n$ and whose conclusion is τ . ($\sigma_1, \dots, \sigma_n$ need not all be distinct.)

When $n = 0$ the deduction is called a **proof** of τ and we call τ a **provable formula** or **theorem** of the \mathbb{A} -logic in question, and we say

$$\vdash_{\mathbb{A}} \tau.$$

The set of all theorems in an \mathbb{A} -logic may be called \mathbb{A}^+ .

6D1.1 *Note* (i) Rule $(\rightarrow E)$ is often called **modus ponens** or the **detachment rule**. In it, $\sigma \rightarrow \tau$ is called its **major premise** and τ its **minor premise**.

(ii) Rule (Sub) is the **substitution rule**. Its side-condition is standard in propositional logic and says, roughly speaking, that substitutions may be made for variables occurring only in axioms. It excludes such deductions as

$$\frac{a}{b} [b/a].$$

6D1.2 *Example* Let \mathbb{A} contain the formulae (C) and (K) shown in 6B2.1. Then the following deduction gives $\vdash_{\mathbb{A}} a \rightarrow a$. (Both its substitution-steps use the substitution $[(a \rightarrow b \rightarrow a)/b, a/c]$.)

$$\frac{\frac{(C) \quad (a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c}{(a \rightarrow (a \rightarrow b \rightarrow a) \rightarrow a) \rightarrow (a \rightarrow b \rightarrow a) \rightarrow a \rightarrow a} \quad (\text{Sub}) \quad \frac{(K) \quad a \rightarrow b \rightarrow a}{a \rightarrow (a \rightarrow b \rightarrow a) \rightarrow a} \quad (\text{Sub})}{(a \rightarrow b \rightarrow a) \rightarrow a \rightarrow a \quad (a \rightarrow b \rightarrow a)} \quad (\rightarrow E) \quad (K) \quad a \rightarrow b \rightarrow a}{a \rightarrow a} \quad (\rightarrow E)$$

6D1.3 *Warning* It is natural to expect the theory of \mathbb{A} -logics to be fairly simple, because every such logic is a subsystem of classical propositional logic and the latter is very well known. But this is not so. For example consider the **provability problem** for a set \mathbb{A} (the problem of deciding, for each type τ , whether τ is provable in \mathbb{A} -logic or not). For at least one \mathbb{A} -logic it is not yet known whether its provability problem is decidable although the logic has been studied for over 30 years (see 7D11(ii),

system T_{\rightarrow}). Further, for every recursively enumerable degree of unsolvability, however high, there is a finite set \mathbb{A} whose provability problem has that degree (Gladstone 1965 §6 Thm 1).

6D2 Definition In any \mathbb{A} -logic a *substitutions-first* deduction is a deduction in which rule (Sub) is only applied to axioms.

6D2.1 Lemma In any \mathbb{A} -logic, every deduction Δ can be replaced by a substitutions-first deduction Δ^* with the same assumptions, axioms and conclusion.

Proof Suppose rule (Sub) is applied below an application of rule (\rightarrow E), as follows:

$$\frac{\sigma \rightarrow \tau \quad \sigma}{\quad} (\rightarrow E)$$

$$\frac{\tau}{s(\tau)} (\text{Sub})$$

Then (Sub) can be moved up above (\rightarrow E), thus:

$$\frac{\frac{\sigma \rightarrow \tau}{s(\sigma) \rightarrow s(\tau)} (\text{Sub}) \quad \frac{\sigma}{s(\sigma)} (\text{Sub})}{\quad} (\rightarrow E)$$

$$s(\tau)$$

Also two successive (Sub)'s can be combined into one. The moving-up procedure ends when all (Sub)'s are at the tops of branches in the deduction-tree. By the restriction on (Sub) in 6D1, the top formula of each of these branches cannot be a non-axiom assumption, so it must be an axiom. \square

6D3 Definition *Hilbert-style Intuitionist logic* (of implication only) is the \mathbb{A} -logic whose \mathbb{A} has just the following four members:

- (B) $(a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b$,
- (C) $(a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c$,
- (K) $a \rightarrow b \rightarrow a$,
- (W) $(a \rightarrow a \rightarrow b) \rightarrow a \rightarrow b$.

6D4 Definition *Hilbert-style R_{\rightarrow}* is the \mathbb{A} -logic for $\mathbb{A} = \{(B), (C), (I), (W)\}$, where (I) is

- (I) $a \rightarrow a$.

6D5 Definition *Hilbert-style BCK-logic* (of implication only) is the \mathbb{A} -logic for the set $\mathbb{A} = \{(B), (C), (K)\}$.

6D6 Definition *Hilbert-style BCI-logic* (of implication only) is the \mathbb{A} -logic for the set $\mathbb{A} = \{(B), (C), (I)\}$.

6D6.1 *Note* (i) By Example 6D1.2, (I) is provable in Hilbert-style BCK-logic and Intuitionist logic.

(ii) Of course (B), (C), (I), (K) and (W) are the principal types of the λ -terms **B**, **C**, **I**, **K** and **W**, see Table 3E2a. But each one also expresses a property of implication that has its own interest quite independently of type-theory. Roughly speaking, (I) is the reflexivity property of implication, (C) states that hypotheses can be commuted, (K) that redundant hypotheses can be added, (W) that duplicates can be removed. Finally (B) can be viewed either as a transitivity property of implication or as a “right-handed” replacement property which says that if $a \rightarrow b$ holds, then a may be replaced by b in the formula $c \rightarrow a$.

(iii) The next theorem will describe the precise connection between \mathbb{A} -logics and λ -calculi; it will be a Hilbert-style analogue of the Curry-Howard theorem at the end of the previous section. As mentioned earlier, the type-system that corresponds most naturally to a Hilbert-style logic is combinatory logic not λ -calculus, but we can build a correspondence with λ -calculus if we are careful; the key will be the concept of *applicative combination* defined in 9F1.

6D7 Theorem (Curry-Howard for Hilbert systems) *Let $\{C_1, C_2, \dots\}$ be a finite or infinite set of typable closed λ -terms and let $\mathbb{A} = \{\gamma_1, \gamma_2, \dots\}$ where $\gamma_i \equiv PT(C_i)$. Then*

(i) *the theorems of \mathbb{A} -logic are exactly the types of the typable applicative combinations of C_1, C_2, \dots ,*

(ii) *the relation $\sigma_1, \dots, \sigma_n \vdash_{\mathbb{A}} \tau$ holds iff there exist an applicative combination M of C_1, C_2, \dots and some distinct term-variables x_1, \dots, x_n , such that*

$$x_1:\sigma_1, \dots, x_n:\sigma_n \vdash_{\lambda} M:\tau.$$

Proof Part (i) is the case $n = 0$ of (ii). To prove (ii) we first prove “if”. Let M be an applicative combination of $x_1, \dots, x_n, C_1, C_2, \dots$, and let Δ be a TA_{λ} -deduction of

$$(1) \quad x_1:\sigma_1, \dots, x_n:\sigma_n \mapsto M:\tau.$$

Corresponding to each occurrence of a C_i in M there will be an occurrence of $\mapsto C_i:s(\gamma_i)$ in Δ for some substitution s . Remove from Δ all steps above these occurrences of C_1, C_2, \dots , and replace each formula $\mapsto C_i:s(\gamma_i)$ by the type $s(\gamma_i)$. Then replace every other formula in Δ , say $\Gamma \mapsto N:\rho$, by the type ρ . (Cf. 6B1(i)–(ii).) The result is a Hilbert-style deduction giving

$$(2) \quad \sigma_1, \dots, \sigma_n \vdash_{\lambda} \tau.$$

To prove “only if” in (ii), let $\sigma_1, \dots, \sigma_n \vdash \tau$ in \mathbb{A} -logic. Then by 6D2.1 there is a deduction Δ of τ in which (Sub) is only applied to axioms. Change Δ to a TA_{λ} -deduction thus. First choose some distinct term-variables x_1, \dots, x_n and replace each undischarged branch-top occurrence of each σ_i in Δ by

$$x_i:\sigma_i \mapsto x_i:\sigma_i.$$

Next, note that each application of (Sub) in Δ will be applied to an axiom to give, say, $\gamma_j \vdash s(\gamma_j)$; replace it by a TA_{λ} -proof of $\mapsto C_j:s(\gamma_j)$. Then replace the logic rule (\rightarrow E) by the TA_{λ} -rule (\rightarrow E) throughout. The result is a TA_{λ} -deduction of (1) for some term M as required. \square

6D7.1 Corollary *The relation $\sigma_1, \dots, \sigma_n \vdash \tau$ holds in the Hilbert version of Intuitionist logic, R_{\rightarrow} , BCK- or BCI-logic iff*

$$x_1:\sigma_1, \dots, x_n:\sigma_n \vdash_{\lambda} M:\tau$$

holds for some distinct x_1, \dots, x_n and some M which is an applicative combination of x_1, \dots, x_n and, respectively, $\{\mathbf{B}, \mathbf{C}, \mathbf{K}, \mathbf{W}\}$, $\{\mathbf{B}, \mathbf{C}, \mathbf{I}, \mathbf{W}\}$, $\{\mathbf{B}, \mathbf{C}, \mathbf{K}\}$ or $\{\mathbf{B}, \mathbf{C}, \mathbf{I}\}$.

6D8 Theorem (Hilbert-Gentzen link) *For the Intuitionist, R_{\rightarrow} , BCK- and BCI-logics the relation*

$$\sigma_1, \dots, \sigma_n \vdash \tau$$

holds in the Natural Deduction version iff it holds in the Hilbert version.

Proof By the Curry-Howard theorems for Natural Deduction (6B7(ii), 6C5(ii)), the combinatory completeness theorems (9F3, 9F5), and Corollary 6D7.1. \square

6D8.1 Note The Hilbert-Gentzen link is usually proved directly without going through λ -calculus. The key step in a direct proof is a result usually called the *deduction theorem* which is a close analogue of the combinatory completeness theorems in 9F. (The deduction theorem is treated in many standard introductions to classical logic, see for example Hamilton 1988 Proposition 2.8; a study of the theorem in some different logics is in Bunder 1982.)

The converse principal-type algorithm

This chapter combines the theme of propositional logic from Chapter 6 with that of principal types from Chapter 3. We saw in the Curry-Howard theorem (6B7) that the types of the closed terms are exactly the theorems of the intuitionist logic of implication: hence the principal types of these terms must form a subset of these theorems, and the very natural question arises of just how large this subset is. Do its members form an aristocracy distinguished in some structural way from the general rabble of theorems, or can every theorem be a principal type?

The main result of the chapter will show that there is in fact no aristocracy: if a type τ is assignable to a closed term M but is not the principal type of M , then it is the principal type of another closed term M^* .

The proof will include an algorithm to construct M^* when τ and M are given. To build M^* an occurrence of M will be combined with some extra terms chosen from a certain carefully defined stock of “building blocks”, closed terms with known principal types; and the main aim of this chapter will be to build M^* from as restricted a set of building blocks as possible.

The algorithm in the earliest known proof needed full λ -calculus (Hindley 1969), but two later ones used only λI -terms as building blocks (Mints and Tammet 1991, Hirokawa 1992a §3) and another used an even more restricted class (Meyer and Bunder 1988 §9). The algorithm below will be based on the latter very economical one.

When it was first proved the existence of M^* seemed nothing more than a mere technical curiosity, but several years later it was discovered by David Meredith to be equivalent to the completeness of a neat form of resolution rule for Intuitionist implicational logic. This rule and its completeness problem will be described at the end of the chapter.

7A The converse PT theorems

In fact the situation is not quite as simple as suggested above, as the converse PT theorems for full λ -calculus and λI -calculus are very slightly different. They will be stated separately here.

First, recall that the class of unrestricted λ -terms is sometimes called the λK -calculus (because it contains $\mathbf{K} \equiv \lambda xy \cdot x$) and its terms λK -terms to distinguish them from restricted classes of terms.

7A1 Definition A λK -PT [respectively, λI -PT, $BCK\lambda$ -PT, $BCI\lambda$ -PT] is the principal type of a closed λK -term [λI -term, $BCK\lambda$ -term, $BCI\lambda$ -term].

A $BCKW$ -PT [$BCIW$ -PT, etc.] is the principal type of a $BCKW$ -combination [$BCIW$ -combination, etc.] as defined in 9F1.

7A1.1 Lemma A type is a λK -PT, λI -PT, $BCK\lambda$ -PT or $BCI\lambda$ -PT iff it is, respectively, a $BCKW$ -, $BCIW$ -, BCK - or BCI -PT.

Proof By 9F3 and 9F5. □

7A2 Converse PT Theorem for λK (i) Every type of a closed λK -term is also a λK -PT.

(ii) Further, there is an algorithm which accepts any typable closed λK -term M and any type τ of M , and constructs a closed λK -term M^* whose PT is τ .

(iii) Furthermore, $M^* \triangleright_{\beta} M$.

Proof There is proof in Hindley 1969 Thm. 3 of the corresponding result for combinatory logic. This translates straightforwardly into λ -calculus. It contains an algorithm that constructs, for every type σ , a closed λK -term I_{σ} such that

$$PT(I_{\sigma}) \equiv \sigma \rightarrow \sigma, \quad I_{\sigma} \triangleright_{\beta} \mathbf{I}.$$

The last step is to choose $M^* \equiv I_{\tau}M$. Then an application of the PT algorithm (3E1) shows that $PT(M^*) \equiv \tau$.

(An alternative proof of (i)–(ii) would be to apply Theorem 7A3 below to construct I_{σ} , but this would not give (iii) but only $M^* \triangleright_{\beta\eta} M$.) □

Before going on to the λI -theorem we pause to look at a particularly neat and interesting application of the λK -theorem. The following corollary shows even more clearly than Examples 2C2.2–6 how widely a term's set of types can vary when the term is β - or η -converted.

7A2.1 Corollary There exist two typable closed λK -terms P and P' such that $P =_{\beta} P'$ but P and P' have no types in common at all.

Proof The following two types have no substitution-instances in common (cf. 3C2.2):

$$\sigma \equiv a \rightarrow a, \quad \tau \equiv (b \rightarrow b) \rightarrow b.$$

Hence also the following have no instances in common:

$$\sigma \rightarrow \sigma, \quad \tau \rightarrow \tau.$$

But by 7A2 there exist I_{σ} and I_{τ} such that $I_{\sigma} =_{\beta} \mathbf{I} =_{\beta} I_{\tau}$ and

$$PT(I_{\sigma}) \equiv \sigma \rightarrow \sigma, \quad PT(I_{\tau}) \equiv \tau \rightarrow \tau.$$

Define $P \equiv I_{\sigma}$, $P' \equiv I_{\tau}$. □

- 7A3 Converse PT Theorem For λI** (i) Every type of a closed λI -term is also a λI -PT.
(ii) Further, there is an algorithm which accepts any typable closed λI -term M and any type τ of M , and builds a closed λI -term M^* whose PT is τ .
(iii) Furthermore, $M^* \triangleright_{\beta\eta} M$.

7A3.1 Proof-note An algorithm and correctness proof will be given in 7C. Note that (iii) is weaker than the corresponding result for λK in 7A2(iii): in λK we have \triangleright_β but in λI the proof will only give $\triangleright_{\beta\eta}$.

The first known algorithm for λI was devised in 1986 by Robert Meyer (Meyer and Bunder 1988 §9), and another was invented independently by Grigori Mints and Tanel Tammet (Mints and Tammet 1991 §2). Both of these originated in the context of Hilbert-style axiom-based logics and although they can easily be translated into λ -calculus they lose some of their directness when this is done. In 1990 a direct and natural λ -algorithm was described by Sachio Hirokawa (Hirokawa 1992a §3).

The most economical of these algorithms is the Meyer one, in the sense that it does not need all the λI -terms as building blocks but only a certain well-defined proper subset of them. So this algorithm is the one that will be described in 7C.

All three algorithms produce terms M^* satisfying (iii) as well as (ii). Though (iii) was first proved for the Hirokawa algorithm (Hirokawa 1992a §3), a proof will be given below for the Meyer algorithm and a careful reading of Mints and Tammet 1991 shows that (iii) holds for that algorithm too.

7A3.1 Question Can 7A3(iii) be strengthened to say “ $M^* \triangleright_\beta M$ ”?

7A3.2 Exercise* If the answer to the above question is “yes” then as a consequence there exist two β -equal typable λI -terms with no types in common (cf. Corollary 7A2.1 for λK). Show that this consequence holds even if the answer to the above question is “no”, by constructing two suitable λI -terms directly. (Hint (Martin Bunder): consider the term P in 2A8.8.)

7B Identifications

The construction of M^* from M and τ in the proof of 7A3 will depend on viewing τ as the result of applying certain substitutions to a particularly simple form of type. The present section introduces the notation needed for this.

First recall the substitution notation introduced in 3B. In particular (from 3B7) a **variables-for-variables** substitution is one with form

$$s \equiv [b_1/a_1, \dots, b_n/a_n]$$

where b_1, \dots, b_n are any variables; and a **renaming in τ** (where τ is a given type) is a variables-for-variables substitution such that $\{a_1, \dots, a_n\} = \text{Vars}(\tau)$ and b_1, \dots, b_n are distinct.

7B1 Definition For any type τ , an *identification in τ* (sometimes called a *contraction in τ*) is any substitution $[b/a]$ such that a, b both occur in τ . A type σ is *obtained from τ by identifications* iff

$$\sigma \equiv [b_n/a_n](\dots([b_1/a_1](\tau))\dots)$$

where $[b_1/a_1]$ is an identification in τ , $[b_2/a_2]$ is an identification in $[b_1/a_1]\tau$, etc.

7B1.1 Example If $\tau \equiv (a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$ then $[b/a]$ is an identification in τ , and

$$[b/a]\tau \equiv (b \rightarrow b \rightarrow c) \rightarrow (b \rightarrow b) \rightarrow b \rightarrow c.$$

7B1.2 Lemma Every variables-for-variables substitution can be performed by a renaming followed by a series of identifications. More precisely, if $\rho \equiv \mathfrak{s}(\tau)$ and \mathfrak{s} substitutes variables for variables, then

$$\rho \equiv [b_n/a_n](\dots([b_1/a_1](\mathfrak{v}(\tau)))\dots)$$

where $n \geq 0$, \mathfrak{v} is a renaming in τ , and each $[b_i/a_i]$ is an identification in

$$[b_{i-1}/a_{i-1}](\dots(\mathfrak{v}(\tau))\dots).$$

7B2 Definition A type τ is *skeletal* iff each variable in τ occurs exactly once. (The property of being skeletal is sometimes called the **1-property**.)

7B3 Lemma Every type τ can be obtained by identifications from a skeletal type τ° (which will be called a *skeleton* of τ).

Proof If τ is skeletal choose $\tau^\circ \equiv \tau$. If not, then for each variable b with two or more occurrences in τ , replace all but one of these occurrences by distinct new variables \square

7C The converse PT proof

This section contains a proof of Theorem 7A3 obtained by translating the proof in Meyer and Bunder 1988 §9 into λ -calculus. The strategy will be to show first that every type of a closed λ I-term can be obtained from a λ I-PT by a series of identifications, and then that identifications preserve the property of being a λ I-PT.

But a preliminary step will be to prove the special case of the theorem in which τ has form $\theta \rightarrow \theta$ where θ is skeletal.

7C1 Lemma Let θ be any skeletal type. Then $\theta \rightarrow \theta$ is the PT of a closed λ I-term I_θ such that

$$I_\theta \triangleright_{\beta\eta} \mathbf{I}.$$

Proof We shall construct I_θ by induction on $|\theta|$. If θ is an atom, say $\theta \equiv a$, choose

$$(1) \quad I_a \equiv \mathbf{I}.$$

Next, suppose $\theta \equiv \rho \rightarrow \sigma$ and I_ρ and I_σ have already been built such that

$$(2) \quad PT(I_\rho) \equiv \rho \rightarrow \rho, \quad PT(I_\sigma) \equiv \sigma \rightarrow \sigma,$$

$$(3) \quad I_\rho \triangleright_{\beta\eta} \mathbf{I}, \quad I_\sigma \triangleright_{\beta\eta} \mathbf{I}.$$

Choose

$$(4) \quad I_{\rho \rightarrow \sigma} \equiv \lambda xy. I_\sigma(x(I_\rho y)).$$

From (2) and the PT algorithm (3E1) it is easy to deduce that

$$(5) \quad PT(I_{\rho \rightarrow \sigma}) \equiv (\rho \rightarrow \sigma) \rightarrow (\rho \rightarrow \sigma).$$

(Note that the proof of (5) depends on the fact that $\text{Vars}(\rho) \cap \text{Vars}(\sigma) = \emptyset$, which holds because $\rho \rightarrow \sigma$ is skeletal.) Also, by (3),

$$\begin{aligned} I_{\rho \rightarrow \sigma} &\triangleright_{\beta\eta} \lambda xy. \mathbf{I}(x(\mathbf{I}y)) \\ &\triangleright_{\beta} \lambda xy. xy \\ &\triangleright_{\eta} \mathbf{I}. \end{aligned}$$

□

7C2 Lemma *Every type of a closed λI -term can be obtained from a λI -PT by identifications. In detail: if M is a closed λI -term and $\vdash_\lambda M : \tau$, there exists a closed typable λI -term M^+ such that*

- (i) $\tau \equiv \mathfrak{s}_k(\mathfrak{s}_{k-1}(\dots \mathfrak{s}_1(PT(M^+))\dots))$ for some identifications $\mathfrak{s}_1, \dots, \mathfrak{s}_k$,
- (ii) $M^+ \triangleright_{\beta\eta} M$,
- (iii) if τ is skeletal then $PT(M^+) \equiv \tau$.

Proof First construct a skeleton τ° of τ as in 7B3. (If τ itself is skeletal, $\tau^\circ \equiv \tau$.) By 7C1 there is a closed λI -term I_{τ° with

$$(1) \quad PT(I_{\tau^\circ}) \equiv \tau^\circ \rightarrow \tau^\circ, \quad I_{\tau^\circ} \triangleright_{\beta\eta} \mathbf{I}.$$

Let $\vdash_\lambda M : \tau$. Then M has a PT, call it σ . And by renaming variables in σ we can ensure that

$$(2) \quad \text{Vars}(\sigma) \cap \text{Vars}(\tau^\circ) = \emptyset.$$

Define

$$(3) \quad M^+ \equiv I_{\tau^\circ} M.$$

Then (ii) holds by (1). To prove (i), apply the PT algorithm to M^+ . The first step is to find an m.g.u. of $\langle \tau^\circ, \sigma \rangle$. To see that one exists, note that τ is an instance of τ° by the definition of τ° , and τ is an instance of σ because σ is principal, so $\langle \tau^\circ, \sigma \rangle$ has a common instance, namely τ ; and by (2) this instance is a unification of $\langle \tau^\circ, \sigma \rangle$; hence $\langle \tau^\circ, \sigma \rangle$ has an m.g.u. τ^+ and τ is an instance of τ^+ , say $\tau \equiv \mathfrak{s}(\tau^+)$. Then

$$(4) \quad PT(M^+) \equiv \tau^+.$$

To complete the proof of (i) the next step is to show that \mathfrak{s} is a variables-for-variables substitution. For this it is enough to show that

$$(5) \quad |\tau| \leq |\tau^+|,$$

because if s substituted a composite type for a variable in τ^+ it would increase $|\tau^+|$. But $|\tau| = |\tau^\circ|$ because τ is obtained from τ° by identifications; and τ^+ is an instance of τ° since τ^+ unifies $\langle \tau^\circ, \sigma \rangle$, so $|\tau^+| \geq |\tau^\circ|$; hence (5) holds.

Thus τ is obtained from a λ I-PT τ^+ by a variables-for-variables substitution s . By 7B1.2, s can be split into a renaming (which changes τ^+ to another PT of M^+) followed by some identifications. Hence (i) holds.

To prove (iii) note that if τ is skeletal then $\tau^\circ \equiv \tau$, so $M^+ \equiv I_\tau M$ and the PT of the latter is clearly τ . \square

7C2.1 Note By the above lemma, the proof of Theorem 7A3 will be complete if we can prove that identifications preserve the property of being a λ I-PT. This will be done in the next three lemmas.

In them the following notation will be useful: if a_1, \dots, a_n are distinct and each has exactly one occurrence in a given type τ , we may say

$$\tau\langle a_1, \dots, a_n \rangle, \quad \tau\langle \sigma_1, \dots, \sigma_n \rangle$$

for τ and $[\sigma_1/a_1, \dots, \sigma_n/a_n]\tau$ respectively.

7C3 Single-Replacement Lemma *If τ is skeletal and $a \in \text{Vars}(\tau)$ and $b \notin \text{Vars}(\tau)$, there exists a closed λ I-term Q whose PT is*

$$(i) \quad (a \rightarrow b)^* \rightarrow \tau\langle a \rangle \rightarrow \tau\langle b \rangle,$$

where $(a \rightarrow b)^* \equiv a \rightarrow b$ if the sole occurrence of a in τ is positive in the sense of Definition 9D3, but $(a \rightarrow b)^* \equiv b \rightarrow a$ if it is negative. Further,

$$(ii) \quad Q \vdash_{\beta\eta} \mathbf{I}.$$

Proof Q will be constructed by induction on $|\tau|$.

Basis: $\tau \equiv a$. Then $(a \rightarrow b)^* \equiv a \rightarrow b$. Choose $Q \equiv \lambda xy. xy$; then $Q \vdash_{\beta} \mathbf{I}$, and by the PT algorithm (3E1),

$$PT(Q) \equiv (a \rightarrow b) \rightarrow (a \rightarrow b).$$

Induction step: $\tau \equiv \rho \rightarrow \sigma$. Then ρ and σ are skeletal, so by 7C1 there exist λ I-terms I_ρ and I_σ with

$$(1) \quad PT(I_\rho) \equiv \rho \rightarrow \rho, \quad PT(I_\sigma) \equiv \sigma \rightarrow \sigma,$$

$$(2) \quad I_\rho \vdash_{\beta\eta} \mathbf{I}, \quad I_\sigma \vdash_{\beta\eta} \mathbf{I}.$$

Case 1: the sole occurrence of a in τ is positive and is in ρ . Then $(a \rightarrow b)^* \equiv a \rightarrow b$. But the positive occurrence of a in τ corresponds to a negative occurrence in ρ , so by the induction hypothesis there is a λ I-term $Q_{\rho-}$ such that

$$(3) \quad PT(Q_{\rho-}) \equiv (b \rightarrow a) \rightarrow \rho\langle a \rangle \rightarrow \rho\langle b \rangle,$$

and $Q_{\rho-} \vdash_{\beta\eta} \mathbf{I}$. By renaming the variables in (3) we get

$$PT(Q_{\rho-}) \equiv (a \rightarrow b) \rightarrow \rho\langle b \rangle \rightarrow \rho\langle a \rangle$$

Choose

$$(4) \quad Q \equiv \lambda xyz. I_\sigma(y(Q_{\rho-}xz)).$$

Since $(a \rightarrow b)^* \equiv a \rightarrow b$, we must prove that

$$(5) \quad PT(Q) \equiv (a \rightarrow b) \rightarrow (\rho \langle a \rangle \rightarrow \sigma) \rightarrow (\rho \langle b \rangle \rightarrow \sigma).$$

But this comes easily from applying the PT algorithm to Q . Finally it is simple to check that $Q \vdash_{\beta\eta} I$, using (2).

Case 2: the occurrence of a in τ is positive and is in σ . Then a occurs positively in σ , so by the induction hypothesis there exists $Q_{\sigma+}$ such that

$$(6) \quad PT(Q_{\sigma+}) \equiv (a \rightarrow b) \rightarrow \sigma \langle a \rangle \rightarrow \sigma \langle b \rangle,$$

and $Q_{\sigma+} \vdash_{\beta\eta} I$. Choose

$$(7) \quad Q \equiv \lambda xyz \cdot Q_{\sigma+} x (y(I_\rho z)).$$

Then $Q \vdash_{\beta\eta} I$ easily. Now $(a \rightarrow b)^* \equiv a \rightarrow b$ since a occurs positively in τ ; and by the PT algorithm and perhaps some renaming we get

$$(8) \quad PT(Q) \equiv (a \rightarrow b) \rightarrow (\rho \rightarrow \sigma \langle a \rangle) \rightarrow (\rho \rightarrow \sigma \langle b \rangle).$$

Case 3: the occurrence of a in τ is negative and is in ρ . Then a occurs positively in ρ , so by the induction hypothesis there exists $Q_{\rho+}$ such that

$$(9) \quad PT(Q_{\rho+}) \equiv (a \rightarrow b) \rightarrow \rho \langle a \rangle \rightarrow \rho \langle b \rangle,$$

and $Q_{\rho+} \vdash_{\beta\eta} I$. Choose

$$(10) \quad Q \equiv \lambda xyz \cdot I_\sigma(y(Q_{\rho+} xz)).$$

Then $Q \vdash_{\beta\eta} I$. Now $(a \rightarrow b)^* \equiv b \rightarrow a$ since a occurs negatively in τ ; and by the PT algorithm and perhaps some renaming, we get

$$(11) \quad PT(Q) \equiv (b \rightarrow a) \rightarrow (\rho \langle a \rangle \rightarrow \sigma) \rightarrow (\rho \langle b \rangle \rightarrow \sigma).$$

Case 4: the occurrence of a in τ is negative and is in σ . Then a occurs negatively in σ , so the induction hypothesis gives $Q_{\sigma-}$ such that

$$(12) \quad PT(Q_{\sigma-}) \equiv (b \rightarrow a) \rightarrow \sigma \langle a \rangle \rightarrow \sigma \langle b \rangle$$

and $Q_{\sigma-} \vdash_{\beta\eta} I$. Choose

$$(13) \quad Q \equiv \lambda xyz \cdot Q_{\sigma-} x (y(I_\rho z)).$$

Then $Q \vdash_{\beta\eta} I$. Now $(a \rightarrow b)^* \equiv b \rightarrow a$ since a occurs negatively in τ ; and by the PT algorithm and perhaps some renaming we get

$$(14) \quad PT(Q) \equiv (b \rightarrow a) \rightarrow (\rho \rightarrow \sigma \langle a \rangle) \rightarrow (\rho \rightarrow \sigma \langle b \rangle). \quad \square$$

7C4 Double-Replacement Lemma *If τ is skeletal and a_1, a_2 are distinct variables in τ , and b_1, b_2 are distinct variables not in τ , then there is a closed λI -term R whose PT is*

$$(i) \quad (a_1 \rightarrow b_1)^* \rightarrow (a_2 \rightarrow b_2)^* \rightarrow \tau \langle a_1, a_2 \rangle \rightarrow \tau \langle b_1, b_2 \rangle,$$

where, for each $i \leq 2$, $(a_i \rightarrow b_i)^* \equiv a_i \rightarrow b_i$ if the occurrence of a_i in τ is positive, but $(a_i \rightarrow b_i)^* \equiv b_i \rightarrow a_i$ if it is negative. Also

$$(ii) \quad R \vdash_{\beta\eta} I.$$

Proof By 7C3 there exist λ I-terms P and Q such that

$$\begin{aligned} PT(P) &\equiv (a_1 \rightarrow b_1)^* \rightarrow \tau \langle a_1, a_2 \rangle \rightarrow \tau \langle b_1, a_2 \rangle, \\ PT(Q) &\equiv (a_2 \rightarrow b_2)^* \rightarrow \tau \langle b_1, a_2 \rangle \rightarrow \tau \langle b_1, b_2 \rangle, \end{aligned}$$

and $PI \triangleright_{\beta\eta} I$ and $QI \triangleright_{\beta\eta} I$. Choose

$$(1) \quad R \equiv \lambda xyz \cdot Qy(Pxz). \quad \square$$

7C5 Identification Lemma *If τ is a λ I-PT then so are all types obtained from τ by identifications. Further, there is an algorithm which, when given a λ I-term M and $\tau \equiv PT(M)$ and $\sigma \equiv [b/a]\tau$ with $a, b \in \text{Vars}(\tau)$, will construct a λ I-term N such that*

$$PT(N) \equiv \sigma, \quad N \triangleright_{\beta\eta} M.$$

Proof Let $\tau \equiv PT(M)$ for some closed λ I-term M and let $\sigma \equiv [b/a]\tau$ where $a, b \in \text{Vars}(\tau)$. We must prove that σ is a λ I-PT. Let

$$\text{Vars}(\tau) = \{a, b, c_1, \dots, c_k\}$$

where a, b, c_1, \dots, c_k are distinct, and let p, q, n_1, \dots, n_k respectively be the number of occurrences of each of these variables in τ .

Define τ° to be the result of replacing each occurrence of each variable in τ by a distinct new variable; say

$$\tau^\circ \equiv \tau^\circ \langle a_1, a_2, \dots, a_p, b_1, b_2, \dots, b_q, c_{1,1}, \dots, c_{1,n_1}, \dots, c_{k,1}, \dots, c_{k,n_k} \rangle,$$

where a_1, \dots, a_p replace the occurrences of a , etc. Then replace a_1 and b_1 in τ° by two new variables f and g distinct from all those above, and call the result τ' :

$$\tau' \equiv \tau^\circ \langle f, a_2, \dots, a_p, g, b_2, \dots, b_q, c_{1,1}, \dots, c_{1,n_1}, \dots, c_{k,1}, \dots, c_{k,n_k} \rangle.$$

Apply the double-replacement lemma (7C4) to a_1, b_1, f, g and τ° : this gives a closed λ I-term R such that

$$(1) \quad PT(R) \equiv (a_1 \rightarrow f)^* \rightarrow (b_1 \rightarrow g)^* \rightarrow (\tau^\circ \rightarrow \tau')$$

where $(a_1 \rightarrow f)^*, (b_1 \rightarrow g)^*$ are defined as in 7C4 depending on whether a_1, b_1 occur positively or negatively in τ° . Define

$$(2) \quad N \equiv (\lambda x \cdot Rxx)I.$$

We shall see that $PT(N) \equiv [b/a]\tau$. First, the PT algorithm gives

$$PT(\lambda x \cdot Rxx) \equiv (a_1 \rightarrow f)^* \rightarrow [a_1/b_1, f/g](\tau^\circ \rightarrow \tau').$$

Hence

$$\begin{aligned} PT((\lambda x \cdot Rxx)I) &\equiv [a_1/f][a_1/b_1, f/g](\tau^\circ \rightarrow \tau') \\ (3) \quad &\equiv [a_1/b_1]\tau^\circ \rightarrow [a_1/f, a_1/g]\tau'. \end{aligned}$$

To finish computing $PT(N)$ we need an m.g.c.i. of $\langle [a_1/b_1]\tau^\circ, \tau' \rangle$. Now $[a_1/b_1]\tau^\circ$ is

$$\tau^\circ \langle a_1, a_2, \dots, a_p, a_1, b_2, \dots, b_q, c_{1,1}, \dots, c_{1,n_1}, \dots, c_{k,1}, \dots, c_{k,n_k} \rangle$$

and τ is

$$\tau^\circ \langle a, a, \dots, a, b, b, \dots, b, c_1, \dots, c_1, \dots, c_k, \dots, c_k \rangle.$$

Consider any common instance ρ of $\langle [a_1/b_1]\tau^\circ, \tau \rangle$: since $[a_1/b_1]\tau^\circ$ has a_1 in the position that was occupied by b_1 in τ° , the types in ρ at the a_1 - and b_1 -positions must be the same. Also, since ρ is an instance of τ , the types in ρ at the positions of a_1, \dots, a_p must be the same, and those at the positions of b_1, \dots, b_q must be the same. Hence an m.g.c.i. of $\langle [a_1/b_1]\tau^\circ, \tau \rangle$ is

$$\tau^\circ \langle b, b, \dots, b, b, b, \dots, b, c_1, \dots, c_1, \dots, c_k, \dots, c_k \rangle.$$

The substitution that generates this from $[a_1/b_1]\tau^\circ$ is

$$[b/a_1, b/a_2, \dots, b/a_p, b/b_2, \dots, b/b_q, c_1/c_{1,1}, \dots].$$

Hence by the PT algorithm and (3), $PT(N)$ is

$$[b/a_1, b/a_2, \dots, b/a_p, b/b_2, \dots, b/b_q, c_1/c_{1,1}, \dots]([a_1/f, a_1/g]\tau'),$$

which is

$$\tau^\circ \langle b, b, \dots, b, b, b, \dots, b, c_1, \dots, c_1, \dots, c_k, \dots, c_k \rangle,$$

and this is $[b/a]\tau$ as required.

Finally, $N \triangleright_{\beta\eta} M$ by (2) and 7C4(ii). □

The following lemma and algorithm complete the proof of Theorem 7A3.

7C6 Lemma *Algorithm 7C7 below accepts any τ and any M such that $\vdash_\lambda M:\tau$, and outputs a closed term M^* such that $PT(M^*) \equiv \tau$ and $M^* \triangleright_{\beta\eta} M$. Further, if M is a λI -term so is M^* .*

Proof See 7C7 and 7C1–5. □

7C7 Converse PT Algorithm (Meyer and Bunder 1988 §9.) **Input:** any pair $\langle \tau, M \rangle$ such that $\vdash_\lambda M:\tau$. (The relation $\vdash_\lambda M:\tau$ is decidable by 3E3.)

Step 1. Follow the proof of 7C2 to construct M^+ and $\tau^+ \equiv PT(M^+)$ and to find a series of identifications s_1, \dots, s_k ($k \geq 0$) such that

$$\tau \equiv s_k(\dots(s_1(\tau^+))\dots).$$

Step 2. If $k = 0$ in Step 1, choose $M^* \equiv M^+$. If $k \geq 1$, apply 7C5 first to τ^+ , then to $s_1(\tau^+)$, then to $s_2(s_1(\tau^+))$, etc., and choose M^* to be the last term N produced.

7C7.1 Note By the proofs of 7C1–5 we can see that the M^* produced by the above algorithm has form

$$M^* \equiv T_k(T_{k-1}(\dots(T_1(I_{\tau^+}M))\dots))$$

where T_1, \dots, T_k and I_{τ^+} are typable closed λI -terms that $\beta\eta$ -reduce to **I**.

7C7.2 Exercise* Apply the above algorithm to construct a term M^* whose PT is τ in the following cases:

- (i) $\tau \equiv (b \rightarrow b) \rightarrow b \rightarrow b, \quad M \equiv \mathbf{I};$
- (ii) $\tau \equiv b \rightarrow b \rightarrow b, \quad M \equiv \mathbf{K}.$

(By the way, in (ii) M is not a λI -term; but the algorithm works despite this, though in such a case it produces an M^* that is also not a λI -term.)

7C7.3 Note As mentioned earlier the above converse PT proof is not quite the one in Meyer and Bunder 1988 §9 but a λ -adaptation of it. The unadapted Meyer-Bunder proof is slightly stronger, and shows that the converse PT theorem holds not only for the class of all λI -terms but for a more restricted class too, namely the **BB'IW**-combinations as defined in 9F1.¹ For completeness' sake this result will now be stated as a formal theorem.

7C8 Theorem (Meyer and Bunder 1988 §9.) (i) *Every type of a **BB'IW**-combination is also a **BB'IW-PT**.*

(ii) *In detail: there is an algorithm which accepts any typable closed term M and any type τ of M , and outputs an applicative combination M^* of M and the closed λ -terms B, B', I, W , such that*

$$PT(M^*) \equiv \tau, \quad M^* \triangleright_{\beta\eta} M.$$

Proof Using the PT algorithm and some patience, it is easy to show that the proofs of Lemmas 7C1–5 stay valid if the definitions of $I_{\rho \rightarrow \sigma}$, Q , R and N in them are replaced by the following:

$$\begin{array}{lll} \text{in 7C1 (4):} & I_{\rho \rightarrow \sigma} & \equiv B(B'B)(BBB')I_\rho I_\sigma, \\ \text{in 7C3 (4):} & Q & \equiv BB(B(B'B')(BBB'))I_\sigma Q_{\rho-}, \\ \text{in 7C3 (7):} & Q & \equiv BB(B(B'B')(BBB'))I_\rho Q_{\sigma+}, \\ \text{in 7C3 (10):} & Q & \equiv BB(B(B'B')(BBB'))I_\sigma Q_{\rho+}, \\ \text{in 7C3 (13):} & Q & \equiv BB(B(B'B')(BBB'))I_\rho Q_{\sigma-}, \\ \text{in 7C4 (1):} & R & \equiv B(B'B')(BB')PQ, \\ \text{in 7C5 (2):} & N & \equiv WRIM. \end{array}$$

□

7D Condensed detachment

We now come to the connection with the resolution-style rule for propositional logic mentioned in the introduction to this Chapter. This rule was originally devised in the early 1950's under the name of the *condensed detachment rule*. It will be described here in the context of the \mathbb{A} -logic generated by an arbitrary set \mathbb{A} of axioms, see 6D1.

7D1 Definition *Rule (D)*, the *condensed detachment rule*, makes the deduction

$$(D) \quad \frac{\rho \rightarrow \sigma \quad \tau}{D(\rho \rightarrow \sigma)\tau},$$

where $D(\rho \rightarrow \sigma)\tau$ is an implicational formula defined thus: if ρ and τ have a common

¹ This class is a proper subclass of the closed λI -terms because it does not contain $C \equiv \lambda xyz.xzy$. (When $Cxyz$ is β -reduced the occurrence of z moves away from the rightmost position, and it is easy to see that no **BB'IW**-combination has this property.)

instance, construct an m.g.c.i. $v \equiv s_1(\rho) \equiv s_2(\tau)$ of $\langle \rho, \tau \rangle$ such that

- (i) $Vars(v) \cap (Vars(\sigma) - Vars(\rho)) = \emptyset$,
- (ii) $Dom(s_1) = Vars(\rho)$, $Dom(s_2) = Vars(\tau)$,

and define

- (iii) $D(\rho \rightarrow \sigma)\tau \equiv s_1(\sigma)$.

(If ρ and τ have no common instance $D(\rho \rightarrow \sigma)\tau$ is undefined and rule (D) is not applicable.)

7D1.1 Lemma *If $D(\rho \rightarrow \sigma)\tau$ is defined, it is the most general conclusion that can be obtained by $(\rightarrow E)$ from instances of $\rho \rightarrow \sigma$ and τ using an instance of $\rho \rightarrow \sigma$ as major premise. More precisely, if $v \equiv s_1(\rho) \equiv s_2(\tau)$ is an m.g.c.i. of $\langle \rho, \tau \rangle$ satisfying 7D1(i)–(ii) and r_1 and r_2 are any substitutions such that $r_1(\rho) \equiv r_2(\tau)$, then $r_1(\sigma)$ is an instance of $s_1(\sigma)$.*

Proof The definition of $D(\rho \rightarrow \sigma)\tau$ is almost identical to the procedure described in 3C4 for computing $PT(PQ)$ when $PT(P) \equiv \rho \rightarrow \sigma$ and $PT(Q) \equiv \tau$; hence the argument in 3C4 proves the present lemma. (By the way, condition 7D1(i) corresponds to (1) in 3C4.) \square

7D1.2 Notes (i) $D(\rho \rightarrow \sigma)\tau$ is unique modulo renaming of variables (by 3C3.2), so in future we may think of it as uniquely defined.

(ii) $D(\rho \rightarrow \sigma)\tau$ is also independent of renaming in $\rho \rightarrow \sigma$ and τ ; that is, if $D(\rho \rightarrow \sigma)\tau$ is defined and $\rho' \rightarrow \sigma'$ and τ' are alphabetic variants of $\rho \rightarrow \sigma$ and τ respectively then $D(\rho' \rightarrow \sigma')\tau'$ is defined and is an alphabetic variant of $D(\rho \rightarrow \sigma)\tau$. (By 3C3.2).

(iii) In any system whose rules include $(\rightarrow E)$ and (Sub) we can derive rule (D) as follows.

$$\frac{(\text{Sub}) \frac{\rho \rightarrow \sigma}{s_1(\sigma) \rightarrow s_1(\sigma)} \quad (\text{Sub}) \frac{\tau}{s_2(\tau)}}{(\rightarrow E) \frac{s_1(\sigma) \rightarrow s_1(\sigma)}{s_1(\sigma)}} \quad (s_2(\rho) \equiv s_2(\tau))$$

The following lemma expresses formally the fact that the definition of $D(\rho \rightarrow \sigma)\tau$ is the same as the procedure in 3C4 for computing $PT(PQ)$ when $PT(P) \equiv \rho \rightarrow \sigma$ and $PT(Q) \equiv \tau$.

7D2 Lemma (D. Meredith.) *If P and Q are typable closed terms with $PT(P) \equiv \rho \rightarrow \sigma$ and $PT(Q) \equiv \tau$, then*

- (i) PQ is typable iff $D(\rho \rightarrow \sigma)\tau$ is defined,
- (ii) $PT(PQ) \equiv D(\rho \rightarrow \sigma)\tau$ modulo permutation of variables.

7D3 Historical Note Although rule (D) is so close to the PT algorithm each of the two concepts was developed in total ignorance of work on the other until around 1978, when David Meredith was the first to realise the parallel. (Hindley and Meredith 1990.)

Rule (D) was invented by Carew Meredith, and David Meredith (his cousin) recalled learning it from him in 1954. Its first appearance in print was in Lemmon et al. 1957 §9. A definition of the rule with historical comments appears in D. Meredith 1977 and further historical comments are in Kalman 1983.

Precursors of the rule date back perhaps as far as the 1920's, however. In Poland, a particular D-computation was published in Lukasiewicz 1939 (p. 276 of the English translation); and there is indirect evidence¹ that Alfred Tarski, then a colleague of Lukasiewicz, might have formulated the D-concept (and therefore perhaps also an algorithm for computing $D(\rho \rightarrow \sigma)\tau$) even before 1930.

Carew Meredith's work on propositional logics involved computing $D(\rho \rightarrow \sigma)\tau$ in many special cases, for example those in Lemmon et al. 1957, but he does not seem to have written out his method as a formal algorithm, at least not in print. However, in 1957 an algorithm for constructing $D(\rho \rightarrow \sigma)\tau$ was formalized by David Meredith and implemented by him as a program for the computer UNIVAC I.²

Now by 7D2 the D-construction is the same as the core of the PT algorithm; hence David Meredith's 1957 D-program was probably in essence the first formal PT algorithm. It was also the first one to be run on a machine. (Cf. Comment 3A7 on the history of PT algorithms.)

7D4 Definition (Condensed logics) For any set \mathbb{A} of formulae, *condensed \mathbb{A} -logic* is obtained by replacing the two rules (\rightarrow E) and (Sub) in the definition of " \mathbb{A} -logic" in 6D1 by the single rule (D). We shall call its deductions *D-deductions*; they are trees constructed from axioms and assumptions by rule (D). For deducibility, etc. in a condensed logic we shall say *D-deducible*, *D-provable*, *D-proof*, *D-theorem*, and shall use the notation

$$\sigma_1, \dots, \sigma_n \vdash_{AD} \tau, \quad \vdash_{AD} \tau.$$

The set of all theorems of condensed \mathbb{A} -logic may be called $\mathbb{A}^{D\vdash}$.

7D4.1 Note By 7D1.2(iii), $\mathbb{A}^{D\vdash} \subseteq \mathbb{A}^{\vdash}$.

7D5 Definition Rule (D) is said to be *complete* for a given set \mathbb{A} of axioms (or \mathbb{A} is said to be *D-complete*) iff

$$\mathbb{A}^{D\vdash} = \mathbb{A}^{\vdash}.$$

7D5.1 Note The concepts of condensed logic and D-completeness have not been defined for Natural Deduction systems but only for Hilbert-style (axiom-based) logics.

7D5.2 Question It is natural to ask which sets \mathbb{A} are D-complete: in particular, which of the axiom-sets for Intuitionist, R_{\rightarrow} -, BCK- and BCI-logics in 6D3–6 are D-complete? This is the question that the converse PT theorem will help to answer, and the key to the answer is the following preliminary theorem.

¹ Mentioned in Kalman 1983 p. 447.

² Noted in Kalman 1983 p. 449 and in correspondence from David Meredith to the present author.

7D6 Meredith's Curry-Howard Theorem (D. Meredith.) *Let $\{C_1, C_2, \dots\}$ be a finite or infinite set of typable closed λ -terms, let $PT(C_i) \equiv \gamma_i$, and let $\mathbb{A} = \{\gamma_1, \gamma_2, \dots\}$. Then the theorems of condensed \mathbb{A} -logic are exactly the PTs of the typable applicative combinations of C_1, C_2, \dots*

Proof By 7D2. □

7D7 Theorem *The axiom-set $\{(B), (C), (I), (W)\}$ for R_{\rightarrow} , given in 6D4 is D-complete.*

Proof Let τ be provable in R_{\rightarrow} . Then by 6D7.1, τ is the type of an applicative combination M of **B**, **C**, **I** and **W**. Hence by 7C8, $\tau \equiv PT(M^*)$ for some applicative combination M^* of M , **B**, **B'**, **I** and **W**. Let M^{**} be the result of replacing **B'** in M^* by the following combination of **C**, **B** and **I**:

$$\mathbf{B}(\mathbf{B}(\mathbf{CIB})\mathbf{B})(\mathbf{CI}).$$

It is straightforward to check that this combination has the same PT as **B'** (in fact it also reduces to **B'**); so $PT(M^{**}) \equiv PT(M^*) \equiv \tau$. Hence τ is a **BCIW**-PT. But by 7D6 every **BCIW**-PT is a D-theorem of the logic whose axioms are the PT's of **B**, **C**, **I**, **W**, and this logic is R_{\rightarrow} . □

The next step in answering Question 7D5.2 is to prove the D-completeness of the axiom-set $\{(B), (C), (K), (W)\}$ for Intuitionist logic. This logic is stronger than R_{\rightarrow} , and if we view the D-completeness of a set of axioms as saying that deductions obtained by rule (Sub) can be imitated using rule (D) in combination with some of the axioms, it is natural to conjecture that if we strengthen a D-complete set it will remain D-complete. The following definition and lemma make this conjecture precise.

7D8 Definition For any sets \mathbb{A} and \mathbb{B} of formulae: \mathbb{B} -logic is called an *extension* of \mathbb{A} -logic iff $\mathbb{B}^+ \supseteq \mathbb{A}^+$; it is called a *D-extension* of \mathbb{A} -logic iff

$$\mathbb{B}^{D+} \supseteq \mathbb{A}^{D+}.$$

7D8.1 Note If $\mathbb{B} \supseteq \mathbb{A}$ then \mathbb{B} -logic is both an extension and a D-extension of \mathbb{A} -logic.

7D9 D-Extension Lemma *If \mathbb{A} is D-complete and $a \rightarrow a$ is an \mathbb{A} -theorem, then every set \mathbb{B} whose logic is a D-extension of \mathbb{A} -logic is D-complete.*

Proof Let $\mathbb{B}^{D+} \supseteq \mathbb{A}^{D+}$; to prove \mathbb{B} is D-complete we must show that if Δ is a deduction from \mathbb{B} by rules (\rightarrow E) and (Sub), each application of (\rightarrow E) or (Sub) in Δ can be replaced by one of (D).

Case (\rightarrow E). (\rightarrow E) is already a special case of (D).

Case (Sub). It is enough to show that if s is any substitution then

$$\tau \in \mathbb{B}^{D+} \implies s(\tau) \in \mathbb{B}^{D+}.$$

First note that $a \rightarrow a \in \mathbb{A}^+$ by assumption, so by (Sub),

$$s(\tau) \rightarrow s(\tau) \in \mathbb{A}^+.$$

But we have assumed $\mathbb{A}^+ = \mathbb{A}^{D+}$ and $\mathbb{A}^{D+} \subseteq \mathbb{B}^{D+}$, so

$$s(\tau) \rightarrow s(\tau) \in \mathbb{B}^{D+}.$$

Now it is easy to see from Definition 7D1 that

$$D(s(\tau) \rightarrow s(\tau), \tau) \equiv s(\tau).$$

Hence by rule (D) applied to the D-theorems $s(\tau) \rightarrow s(\tau)$ and τ we get

$$s(\tau) \in \mathbb{B}^{D+}. \quad \square$$

7D10 Theorem *The axiom-set $\{(B), (C), (K), (W)\}$ for Intuitionist logic given in 6D3 is D-complete.*

Proof By 7D7 and 7D9, since this logic is a D-extension of R_{\rightarrow} . \square

7D11 Note (Other D-complete logics) (i) *Classical logic*. The implicational fragment of classical logic is a Hilbert-style system defined by the axioms (B), (C), (K), (W) and

$$(PL) \quad ((a \rightarrow b) \rightarrow a) \rightarrow a.$$

((PL) is called *Pierce's law*, see 6A1.2 and 6B7.4.) It can be shown that a formula is provable in this system iff it is a tautology in the usual truth-table sense. (Prior 1955 Part I, Ch. III.) By 7D9–10 this system is D-complete.

(ii) *Ticket entailment*. A Hilbert-style logic defined by the axioms (B), (B'), (I) and (W) listed in 6B2.1 was introduced in Anderson and Belnap 1975 (see especially Ch. 1 §§6, 8.3.2), where it was called T_{\rightarrow} or *the logic of ticket entailment*. The details of T_{\rightarrow} and its motivation are not the concern of this book, but in a sense (B) and (B') are “right-handed” and “left-handed” replacement properties: if $a \rightarrow b$ holds, (B) says that a can be replaced by b in the formula $c \rightarrow a$ and (B') says that b can be replaced by a in $a \rightarrow c$. (Meanings for (I) and (W) were suggested in 6D6.1(ii).) This logic can be shown to be weaker than R_{\rightarrow} , but by 7D6 and 7C8 it is nevertheless D-complete.¹

(iii) A set of axioms whose logic is strictly weaker than T_{\rightarrow} was proved D-complete by N. Megill in unpublished notes in 1993, and an infinite series of ever weakening D-complete axiom-sets has since been constructed (Megill and Bunder 1996).

But not all axiom-sets are D-complete, as the next theorem will show.

7D12 Theorem *The axiom-sets $\{(B), (C), (K)\}$ and $\{(B), (C), (I)\}$ given in 6D5–6 are D-incomplete.*

¹ By the way, it is not yet known whether there is a decision-procedure for provability in T_{\rightarrow} .

Proof By 7D9 it is enough to prove the result for $\{(B), (C), (K)\}$. And by 7D6 it is enough to find a type τ that can be assigned to an applicative combination of **B**, **C**, **K** but is not the PT of such a combination. One such type (pointed out by A. Wronski, cf. Bunder 1986) is

$$\tau \equiv ((a \rightarrow a) \rightarrow a) \rightarrow a.$$

It is easy to deduce from the types of **C** and **K** in Table 3E2a that

$$\vdash_{\lambda} (\mathbf{C}(\mathbf{CKK})(\mathbf{CKK})): \tau.$$

On the other hand, τ cannot be the PT of an applicative combination M of **B**, **C**, **K**. Because if it were, M would be a BCK λ -term and hence $M\mathbf{I}$ would be one too by the definition of BCK λ -terms in 1D2; then by 2D8 $M\mathbf{I}$ would be typable. The last step in a deduction of a type for $M\mathbf{I}$ must have form

$$\frac{\vdash M: (\rho \rightarrow \rho) \rightarrow \sigma \quad \vdash \mathbf{I}: \rho \rightarrow \rho}{\vdash M\mathbf{I}: \sigma} (\rightarrow E)$$

Hence $(\rho \rightarrow \rho) \rightarrow \sigma$ would have to be an instance of $PT(M)$ which is τ . But then $\rho \rightarrow \rho$ would be an instance of $(a \rightarrow a) \rightarrow a$, which is impossible. \square

7D12.1 Warning D-completeness or -incompleteness is a property of sets of axioms, not of the logics they generate. In fact it is possible to have two sets \mathbb{A} and \mathbb{B} of axioms with $\mathbb{A}^+ = \mathbb{B}^+$ but such that \mathbb{A} is D-complete and \mathbb{B} is not. One example is $\mathbb{B} = \{(B), (C), (K)\}$ and $\mathbb{A} = \mathbb{B}^+$. There are also examples with \mathbb{A} and \mathbb{B} both finite (Martin Bunder, unpublished notes 1994).

This brings our excursion into propositional logic to an end. (Except that if one identifies λ -terms with deductions the whole of this book is a look at the proof-theory of propositional logic from a particular point of view.)

Counting a type's inhabitants

Given a type τ , how many closed terms can receive type τ in TA_λ ? As stated this question is trivial, since if the answer is not zero it is always infinite; for example the type $a \rightarrow a$ can be assigned to all members of the sequence **I**, **II**, **III**, etc. But if we change the question to ask only for terms in normal form, the answer is often finite and interesting patterns show up which are still not completely understood.

The aim of this chapter is to describe an algorithm from Ben-Yelles 1979 that answers the “how many” question for normal forms. For each τ it will decide in a finite number of steps whether the number of closed β -normal forms that receive type τ is finite or infinite, will compute this number in the finite case, and will list all the relevant terms in both cases.

Ben-Yelles' algorithm can be used in particular to test whether the number of terms with type τ is zero or not, and as mentioned in 6B7.3 this gives a test for provability in Intuitionist implicational logic.

The first section below will describe the sets to be counted. The next will show some examples of the algorithm's strategy in action. Then in 8C–D the algorithm will be stated formally, and the rest of the chapter will be occupied by a proof that the algorithm does what it claims to do.¹

8A Inhabitants

This section gives precise definitions and notations for the sets to be counted.

8A1 Definition (Inhabitants) An *untyped inhabitant* of τ is a closed term M such that $\vdash_\lambda M:\tau$. A *typed inhabitant* of τ is a closed typed term M^τ . The sets of all typed and untyped inhabitants of τ will be called $Habs_t(\tau)$ and $Habs_u(\tau)$ respectively, or, when no confusion is likely, just

$$Habs(\tau).$$

A *(β -)normal inhabitant* of a type is an inhabitant in β -nf. The sets of all typed and untyped normal inhabitants of τ will be called $Nhabs_t(\tau)$ and $Nhabs_u(\tau)$ respectively, or, when no confusion is likely, just

$$Nhabs(\tau).$$

¹ This fact was first proved mostly in Ben-Yelles 1979 and partly in Hirokawa 1993c.

A **$\beta\eta$ -normal inhabitant** of a type is an inhabitant in $\beta\eta$ -nf. The sets of all typed and untyped $\beta\eta$ -normal inhabitants of τ will both be called

$$Nhabs_{\eta}(\tau).$$

A type with at least one inhabitant is said to be ***inhabited***.

As remarked earlier the aim of this chapter is to count β -normal inhabitants. The following lemma will show that it does not matter whether we count typed or untyped inhabitants.

8A2 Lemma *If $M^{\tau} \in Nhabs_t(\tau)$ then $M^{\tau} \in Nhabs_u(\tau)$; further, the type-erasing mapping is a one-to-one correspondence between the typed and the untyped β -normal inhabitants of τ (modulo \equiv_{α}). The same holds for $\beta\eta$ -normal inhabitants.*

Proof Let $M \in \beta$ -nf; then M inhabits τ iff there exists a proof Δ of $\vdash M:\tau$; and by 2B3 this Δ is uniquely determined by M . And such proofs correspond one-to-one with typed closed terms by 5A7. \square

8A2.1 Notation All terms in this chapter will be typed unless explicitly stated otherwise. But for ease of reading they will often be written with some or all of their types omitted.

8A3 Definition The number $(0, 1, 2, \dots$ or $\infty)$ of members of a set \mathbb{S} , counted modulo \equiv_{α} if \mathbb{S} is a set of λ -terms, is called the ***cardinality*** of \mathbb{S} or

$$\#(\mathbb{S}).$$

For $\#(Nhabs(\tau))$ and $\#(Nhabs_{\eta}(\tau))$ we shall usually say just

$$\#(\tau), \quad \#_{\eta}(\tau).$$

8A4 Definition (Counting, enumerating) A distinction will be made in this book between ***counting*** and ***enumerating*** a set \mathbb{S} of β -normal inhabitants of a type τ :

(i) to ***count*** \mathbb{S} will mean to compute $\#(\mathbb{S})$ after a finite number of steps (even when $\#(\mathbb{S}) = \infty$);

(ii) to ***enumerate*** or ***list*** \mathbb{S} will mean to enumerate \mathbb{S} in the usual recursion-theoretic sense, i.e. to output a sequence consisting of all the members of \mathbb{S} (and no non-members!), continuing for ever if \mathbb{S} is infinite.

8A4.1 Comment The aim of Ben-Yelles' algorithm is to count inhabitants as well as enumerate them. Mere enumeration would be easy: we could simply list all closed typed β -nf's in some standard order and for each one decide whether it is an inhabitant of a given τ by looking at its type. But counting is not so easy: we must find a way of enumerating $Nhabs(\tau)$ which will tell us after only a finite number of steps whether the enumeration will continue for ever or not.

The strategy will be to do the counting in order of increasing *depth* in a sense to be defined below, and to first count certain inhabitants called *long β -nf's* from which

all others can be generated by η -reduction. Before defining “depth” and “long” we shall look briefly at the structure of arbitrary typed β -nf's (cf. untyped β -nf's, 1B10).

8A5 Lemma (Structure of a typed β -nf) *Let Γ be a type-context. Every β -nf $M^\tau \in \mathbb{T}\mathbb{T}(\Gamma)$ can be expressed uniquely in the form*

$$(i) \quad (\lambda x_1^{\tau_1} \dots x_m^{\tau_m} \cdot (v^{(\rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow \tau^*)} M_1^{\rho_1} \dots M_n^{\rho_n})^{\tau^*})^{(\tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow \tau^*)}$$

where $m \geq 0$, $n \geq 0$, and

$$(ii) \quad \tau \equiv \tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow \tau^*$$

for some τ^* , possibly composite, and each $M_j^{\rho_j}$ is a β -nf that is typed relative to

$$(iii) \quad \Gamma \cup \{x_1 : \tau_1, \dots, x_m : \tau_m\}$$

(and the set displayed in (iii) is consistent). Further, if M^τ is closed then $m \geq 1$ and there is an $i \leq m$ such that

$$(iv) \quad v \equiv x_i, \quad \tau_i \equiv \rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow \tau^*.$$

Proof Straightforward induction on $|M^\tau|$. Note that m , n etc. are determined uniquely, just as in 1B10 for untyped β -nf's. \square

8A5.1 Notation In the term in 8A5(i) the displayed occurrences of

$$\lambda x_1^{\tau_1}, \dots, \lambda x_m^{\tau_m}, \quad v^{(\rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow \tau^*)}, \quad M_1^{\rho_1}, \dots, M_n^{\rho_n}$$

are called respectively the term's *initial abstractors*, *head* and *arguments*.

8A6 Definition (Depth(M^τ)) Define the *depth* of a typed or untyped β -nf thus:

- (i) $Depth(y) = Depth(\lambda x_1 \dots x_m \cdot y) = 0$;
- (ii) $Depth(\lambda x_1 \dots x_m \cdot y M_1 \dots M_n) = 1 + \max_{1 \leq j \leq n} Depth(M_j)$ if $n > 0$.

8A6.1 Examples $Depth(\lambda uv \cdot uxvx) = Depth(z(\lambda w \cdot y)) = 1$,
 $Depth(\lambda x \cdot x(z(\lambda w \cdot y))(\lambda uv \cdot uxvx)) = 2$.

8A6.2 Lemma $Depth(M^\tau) = Depth(M^{\lambda}) < |M^{\lambda}|$.

8A7 Definition (Long β -nf's) A typed β -nf M^τ is called *long* or *maximal* iff every variable-occurrence \underline{z} in M^τ is followed by the longest sequence of arguments allowed by its type, i.e. iff each component with form $(zP_1 \dots P_n)(n \geq 0)$ that is not in a function position has atomic type.¹ An untyped β -nf M is called *long* relative to a type τ iff it is the type-erasure of a typed long β -nf M^τ . (By 8A2 M^τ is unique.)

The sets of all long normal inhabitants of τ (typed or untyped) will both be called

$$Long(\tau).$$

¹ Sometimes “long η -normal form” is used in the literature for long β -nf's but this is misleading, as these terms are β -nf's but not necessarily η -nf's.

8A7.1 *Example* Let $\tau \equiv ((a \rightarrow b) \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow c$. Then the following normal inhabitant of τ is not long:

$$M^\tau \equiv \lambda x^{(a \rightarrow b) \rightarrow c} y^{a \rightarrow b} . x^{(a \rightarrow b) \rightarrow c} y^{a \rightarrow b}$$

(because $y^{a \rightarrow b}$ has a type which “demands” an argument but none is provided). On the other hand the following one is long:

$$N^\tau \equiv \lambda x^{(a \rightarrow b) \rightarrow c} y^{a \rightarrow b} . x^{(a \rightarrow b) \rightarrow c} (\lambda z^a . y^{a \rightarrow b} z^a).$$

8A7.2 *Definition* The sets of all long normal inhabitants of τ (typed or untyped) with depth $\leq d$ will both be called

$$Long(\tau, d).$$

The next lemma shows that if we could enumerate long normal inhabitants the others would be obtainable from these by η -reduction.

8A8 Lemma (Completeness of $Long(\tau)$) (Ben-Yelles 1979 Lemma 3.9.) *Every normal inhabitant of τ can be η -expanded to a long normal inhabitant of τ . And this long inhabitant is unique (modulo \equiv_α); i.e.*

$$\{M^\tau, N^\tau \in Long(\tau) \text{ and } M^\tau =_\eta N^\tau\} \implies M^\tau \equiv_\alpha N^\tau.$$

Proof Let $P^\tau \in Nhabs(\tau)$. First we must η -expand P^τ to a term $P^{\tau+} \in Long(\tau)$. Then we must show that $P^{\tau+}$ is unique, i.e. that

$$(1) \quad M^\tau \in Long(\tau), \quad M^\tau \triangleright_\eta P^\tau \implies M^\tau \equiv_\alpha P^{\tau+}.$$

Suppose P^τ contains a **short component**, i.e. a component with form

$$(yQ_1 \dots Q_n)^\sigma \quad (n \geq 0)$$

that is not in function position and whose type σ is composite, say

$$\sigma \equiv \sigma_1 \rightarrow \dots \rightarrow \sigma_k \rightarrow a \quad (k \geq 1).$$

Choose distinct new variables z_1, \dots, z_k not occurring in P^τ and replace this component by

$$(\lambda z_1^{\sigma_1} \dots z_k^{\sigma_k} . ((yQ_1 \dots Q_n)^\sigma z_1^{\sigma_1} \dots z_k^{\sigma_k})^a)^\sigma.$$

This replacement is an η -expansion; it is easy to check that its result is still a β -normal form. (cf. Example 8A7.1.)

Make similar replacements until there are no short components left in P^τ . (Each replacement may introduce new short components with types $\sigma_1, \dots, \sigma_k$, but these types are shorter than σ so the replacement process will terminate.)

Call the result $P^{\tau+}$. The proof that this term satisfies (1) is a routine induction on $|P^\tau|$. \square

8A8.1 *Corollary* $Nhabs(\tau) = \emptyset \iff Long(\tau) = \emptyset$.

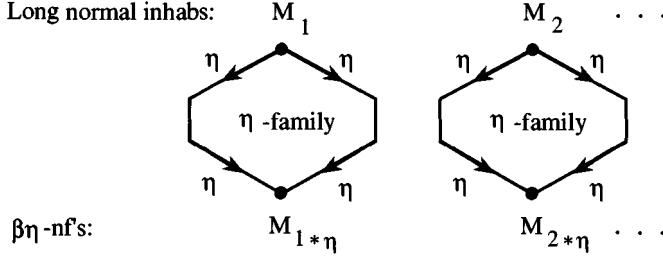


Fig. 8A10a.

8A9 Definition (η -family) The set of all terms obtained by η -reducing M^τ will be called the η -family of M^τ (just as for untyped terms in 1C3), or

$$\{M^\tau\}_\eta.$$

8A9.1 Note Let $M^\tau \in \mathbb{T}(\Gamma)$ for some Γ . Then $\{M^\tau\}_\eta$ is finite (by the typed analogue of 1C3.1) and all its members are in $\mathbb{T}(\Gamma)$ (by 5B7.1). If M^τ is a β -nf then so are all the members of $\{M^\tau\}_\eta$ (by the typed analogue of 1C9.3). Hence

$$(i) \quad M^\tau \in Nhabs(\tau) \implies \{M^\tau\}_\eta \subseteq Nhabs(\tau).$$

Also, if M^τ is a β -nf its η -family contains exactly one $\beta\eta$ -nf (by the typed analogue of 1C9.3). Finally, each normal inhabitant of τ is in the η -family of exactly one long normal inhabitant, by 8A8.

The following lemma summarises this situation.

8A10 Lemma (i) *The η -families of the long typed normal inhabitants of τ partition $Nhabs(\tau)$ into non-overlapping finite subsets, each η -family containing just one long member and just one $\beta\eta$ -nf. (See Fig. 8A10a.)*

(ii) *$\#(\tau)$ is finite or infinite or zero according as $\#_\eta(\tau)$ is finite or infinite or zero.*

(iii) *$\#_\eta(\tau) = \#(Long(\tau))$.*

8A11 Definition (Principal inhabitants) An untyped inhabitant M of τ is called *principal* iff τ is a principal type of M . A typed inhabitant M^τ of τ is called *principal* iff the deduction of $\mapsto M^\tau : \tau$ that it encodes (see 5A7) is principal.

The sets of all principal inhabitants of τ (typed or untyped) will both be called

$$Princ(\tau).$$

The sets of all principal β -normal inhabitants of τ (typed or untyped) will both be called

$$Nprinc(\tau).$$

8A11.1 Lemma *M^τ is a typed principal β -normal inhabitant of τ iff M^τ is an untyped principal β -normal inhabitant of τ .*

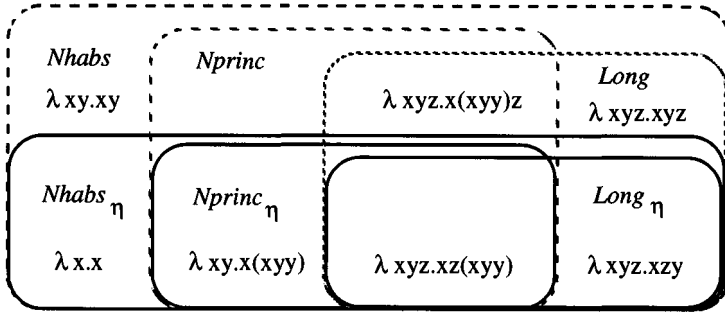


Fig. 8A12a.

8A11.2 Lemma Let $M^{\tau+}$ be the unique member of $Long(\tau)$ to which $M^{\tau}\eta$ -expands (see 8A8). Then

$$M^{\tau} \in Nprinc(\tau) \implies M^{\tau+} \in Nprinc(\tau).$$

Proof The η -expansion in the proof of 8A8 preserves principality because of the way the types given to z_1, \dots, z_k are determined by the type σ of the component that is replaced. \square

8A12 Comment (Sets of nf's) Three sets of β -nf's have been defined so far in this section, namely

$$Nhabs(\tau), \quad Long(\tau), \quad Nprinc(\tau),$$

and the sets of all $\beta\eta$ -nf's in these sets will be called respectively

$$Nhabs_{\eta}(\tau), \quad Long_{\eta}(\tau), \quad Nprinc_{\eta}(\tau).$$

To clarify the relations between these six sets consider the type

$$\tau \equiv (a \rightarrow a \rightarrow a) \rightarrow a \rightarrow a \rightarrow a.$$

(See Fig. 8A12a.) For this τ the six sets are all distinct and in fact there is a term in every space in Fig. 8A12a except one. In detail:

- (i) $\lambda x^{a \rightarrow a \rightarrow a}. x^{a \rightarrow a \rightarrow a} \in Nhabs_{\eta} - (Nprinc \cup Long)$;
- (ii) $\lambda x^{a \rightarrow a \rightarrow a} y^a. (xy)^{a \rightarrow a} \in Nhabs - Nhabs_{\eta} - (Nprinc \cup Long)$;
- (iii) $\lambda x^{a \rightarrow a \rightarrow a} y^a z^a. (xyz)^a \in Long - Nhabs_{\eta} - Nprinc$;
- (iv) $\lambda x^{a \rightarrow a \rightarrow a} y^a. (x(xyy))^{a \rightarrow a} \in Nprinc_{\eta} - Long_{\eta}$;
- (v) $\lambda x^{a \rightarrow a \rightarrow a} y^a z^a. (x(xyy)z)^a \in Nprinc \cap Long - Nhabs_{\eta}$;
- (vi) $\lambda x^{a \rightarrow a \rightarrow a} y^a z^a. (xz(xyy))^a \in Nprinc_{\eta} \cap Long_{\eta}$;
- (vii) $\lambda x^{a \rightarrow a \rightarrow a} y^a z^a. (xzy)^a \in Long_{\eta} - Nprinc_{\eta}$.

Proof of 8A12 (i) The term shown in (i) is clearly a $\beta\eta$ -nf and is not long. It fails to encode a principal deduction for $\lambda x \cdot x$ because the $PT(\lambda x \cdot x)$ is not τ but $a \rightarrow a$.

(ii) This term is obtained by η -expanding the term in (i).

(iii) This term is obtained by η -expanding (i) until it becomes long.

(iv) This term is easily shown to be principal by the PT algorithm (3E1). It fails to be long because its second x from the right has only one argument.

(v) This term is obtained from (iv) by η -expansion; both occurrences of x now have two arguments.

(vi) This term is like (v) but z and xyy have been reversed to make it an η -nf.

(vii) This term is clearly long. However, its PT is not τ but

$$(a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c.$$

8A12.1 Exercise* In Fig. 8A12a one of the eight spaces is empty: show that if τ is changed to the following type then every space in that figure will contain a term:

$$\tau \equiv (a \rightarrow a \rightarrow a \rightarrow a) \rightarrow a \rightarrow a \rightarrow a \rightarrow a.$$

8A13 Remark (Sets containing non-nf's) This chapter is concerned mainly with counting normal forms so the sets $Habs(\tau)$ and $Princ(\tau)$ will play almost no role. However, it is worth noting their relation to $Nhabs(\tau)$ and $Nprinc(\tau)$ before going on.

(i) If τ has an inhabitant M then M has a β -nf M^*_β by WN (5C1) and M^*_β is also an inhabitant of τ by subject-reduction (2C1); hence

$$Habs(\tau) \neq \emptyset \iff Nhabs(\tau) \neq \emptyset.$$

(ii) Next, by the converse PT theorem (7A2) every type with an inhabitant has a principal one, so

$$Habs(\tau) \neq \emptyset \iff Princ(\tau) \neq \emptyset.$$

(iii) In contrast it is not true that $Habs(\tau) \neq \emptyset \Rightarrow Nprinc(\tau) \neq \emptyset$. It is not even true that $Princ(\tau) \neq \emptyset \Rightarrow Nprinc(\tau) \neq \emptyset$. Because τ may have an inhabitant M , even a principal one, such that $PT(M)$ changes when M is reduced to M^*_β . An example is

$$\tau \equiv a \rightarrow a \rightarrow a;$$

by 8B4 this type's only normal inhabitants are $\lambda xy \cdot x$ and $\lambda xy \cdot y$ and it is easy to check that neither of these is principal; on the other hand Table 3E2a(10) showed that τ has a non-normal principal inhabitant

$$(\lambda xyz \cdot K(xy)(xz))I.$$

8B Examples of the search strategy

The core of the counting algorithm will be a *search algorithm* which will seek long normal inhabitants of τ with increasing depths $d = 0, 1, 2, \dots$. The strategy of this algorithm will depend on nothing more than some surprisingly simple comments about the structure of long typed terms. These comments will be collected in 8B2 below and then applied in some examples which will show the search strategy in action.

8B1 Lemma Every type τ can be expressed uniquely in the following form, where $m \geq 0$ and e is an atom:

$$\tau \equiv \tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow e.$$

Proof Easy induction on $|\tau|$. □

8B1.1 Notation The occurrences of τ_1, \dots, τ_m and e displayed above will be called the *premises* and *conclusion* (or *tail*) of τ respectively, and m will be called the *arity* of τ . (See Definition 9E5 for more details.)

Two type-occurrences will be called *isomorphic* iff they are occurrences of the same type. (cf. Definition (iv).) If the tail-components of σ and τ are isomorphic we may say

$$\text{Tail}(\sigma) \cong \text{Tail}(\tau).$$

8B2 Comments (Long typed β -nf's) Let τ be any type; say τ has form

$$\tau \equiv \tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow e \quad (m \geq 0, e \text{ an atom})$$

and let M^τ be any β -nf with type τ . By 8A5, M^τ has form

$$(\lambda x_1^{\tau_1} \dots x_k^{\tau_k} \cdot (v^{(\rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow \tau^*)} M_1^{\rho_1} \dots M_n^{\rho_n})^{\tau^*}) (\tau_1 \rightarrow \dots \tau_k \rightarrow \tau^*)$$

where $0 \leq k \leq m$ and $\tau^* \equiv \tau_{k+1} \rightarrow \dots \tau_m \rightarrow e$. If M^τ is long (see 8A7), then

- (i) $k = m$,
- (ii) $\tau^* \equiv e$,
- (iii) the types of x_1, \dots, x_m coincide with the premises of τ ,
- (iv) the tail of the type of v is isomorphic to that of τ ,
- (v) if M^τ is closed then $m \geq 1$ and v is an x_i ($1 \leq i \leq m$) and

$$\tau_i \equiv \rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow e.$$

The following examples show how the above comments are applied.

8B3 Example (A type τ with $\#(\tau) = 1$) (Ben-Yelles 1979 p. 42.) The following type has exactly one normal inhabitant:

$$\tau \equiv (a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c.$$

And its normal inhabitant (which is also both long and principal) is

$$\mathbf{S}^\tau \equiv \lambda x^{a \rightarrow b \rightarrow c} y^{a \rightarrow b} z^a \cdot xz(yz).$$

Proof We shall start by proving that $\text{Long}(\tau) = \{\mathbf{S}^\tau\}$.

Step 1. First look at the structure of τ : in the notation of 8B2 we have $m = 3$ and

$$e \equiv c, \quad \tau_1 \equiv a \rightarrow b \rightarrow c, \quad \tau_2 \equiv a \rightarrow b, \quad \tau_3 \equiv a.$$

Hence any $M^\tau \in \text{Long}(\tau)$ must have just three initial abstracted variables, say

$$M^\tau \equiv (\lambda x_1^{\tau_1} x_2^{\tau_2} x_3^{\tau_3} \cdot (v^{(\rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow c)} M_1^{\rho_1} \dots M_n^{\rho_n})^c) (\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \rightarrow c).$$

Now by 8B2(iv), v must be one of x_1, x_2, x_3 whose type's tail is isomorphic to that of τ . The tail of τ is \underline{c} , and the only τ_i whose tail is an occurrence of c is τ_1 , so v must be x_1 . Since τ_1 has two premises, x_1 must be followed by exactly two arguments. Hence M must have form

$$(1) \quad M \equiv \lambda x_1^{a \rightarrow b \rightarrow c} x_2^{a \rightarrow b} x_3^a \cdot x_1^{a \rightarrow b \rightarrow c} U^a V^b.$$

Step 2. Search for suitable U^a and V^b . First U^a : its type is an atom so U^a cannot be an abstract, by 8B2(i). Thus U^a has form

$$U^a \equiv (wP_1 \dots P_r)^a \quad (r \geq 0)$$

where w is an x_i whose type's tail is isomorphic to the tail of the type of U . This tail is an occurrence of a , so the only possibility is

$$w \equiv x_3.$$

Since the type of x_3 has no premises we have $r = 0$ and hence

$$(2) \quad U^a \equiv x_3^a.$$

Next search for V^b . Since b is an atom, V^b cannot be an abstract and its head must be an x_i whose type's tail is an occurrence of b . The only possibility is x_2 , and the type of x_2 allows only one argument, so

$$(3) \quad V^b \equiv x_2^{a \rightarrow b} W^a$$

for some W^a .

Step 3. Search for W^a . Just as for U^a , the only possibility is

$$(4) \quad W^a \equiv x_3^a.$$

Conclusion. Modulo \equiv_α there is at most one term in $Long(\tau)$, namely

$$\mathbf{S}^\tau \equiv \lambda x_1^{a \rightarrow b \rightarrow c} x_2^{a \rightarrow b} x_3^a \cdot x_1 x_3 (x_2 x_3).$$

It is easy to check that \mathbf{S}^τ is actually in $Long(\tau)$. Hence, by 8A10 and the fact that \mathbf{S}^τ is η -irreducible,

$$Nhabs(\tau) = \{\mathbf{S}^\tau\}.$$

By the PT algorithm it is easy to check that τ is a principal type of \mathbf{S}^τ ; hence

$$Nprinc(\tau) = \{\mathbf{S}^\tau\}. \quad \square$$

8B4 Example (A type τ with $\#(\tau) = m$) For each $m \geq 2$ the following τ has exactly m normal inhabitants and all are long and non-principal:

$$\tau \equiv a \rightarrow \dots \rightarrow a \rightarrow a \quad (m + 1 \text{ a's}).$$

In fact its normal inhabitants are the following m terms (called *projectors* or *selectors*):

$$\Pi_i^m \equiv \lambda x_1^a \dots x_m^a \cdot x_i^a \quad (1 \leq i \leq m).$$

Proof By 8B2, any $M^\tau \in \text{Long}(\tau)$ must have form

$$M^\tau \equiv \lambda x_1^a \dots x_m^a \cdot v V_1 \dots V_n$$

where $v \equiv x_i$ for some $i \leq m$. But the types of x_1, \dots, x_m have no premises, so $n = 0$. Hence

$$M^\tau \equiv \lambda x_1^a \dots x_m^a \cdot x_i^a.$$

It is easy to see that every such selector is in fact in $\text{Long}(\tau)$. Also all these selectors are η -irreducible, so by 8A10,

$$N\text{habs}(\tau) = \text{Long}(\tau).$$

Now $m \geq 2$, so by the PT algorithm no selector is in $N\text{princ}(\tau)$; hence

$$N\text{princ}(\tau) = \emptyset. \quad \square$$

8B5 Example (Types τ with $\#(\tau) = 0$) (i) No atomic type has inhabitants.

(ii) No type that is skeletal (i.e. in which each atom occurs at most once, cf. 7B2) has inhabitants.

Proof (i) Every type with an inhabitant has a normal one by 8A13(i). But every type with a normal inhabitant has arity $m \geq 1$ by 8B2(iii).

(ii) Let τ be skeletal and let $\tau \equiv \tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow e$ ($m \geq 1$). If τ had inhabitants it would have at least one long normal one by 8A8, and by 8B2 this inhabitant would have form

$$\lambda x_1 \dots x_m \cdot x_i M_1 \dots M_n$$

with x_i having type τ_i and the tail of τ_i being an occurrence of e . But τ is skeletal, so e cannot occur in any τ_i . Hence τ has no inhabitants. \square

8B5.1 Corollary *Intuitionist implicational logic is consistent in the sense that not all formulae are provable.*

Proof If an atomic formula e were provable it would be the type of a closed term M^e by 6B7.1, contradicting 8B5. \square

8B6 Example (Peirce's law, cf. 6A1.2) The type $\tau \equiv ((a \rightarrow b) \rightarrow a) \rightarrow a$ has no inhabitants.

Proof Suppose $M^\tau \in \text{Long}(\tau)$. Then by 8B2, M^τ must have form

$$M^\tau \equiv \lambda x^{(a \rightarrow b) \rightarrow a} \cdot v U_1 \dots U_n \quad (n \geq 0).$$

And $v \equiv x$, since M^τ is closed. Hence $n = 1$, since the type of x has only one premise. Thus

$$M^\tau \equiv \lambda x^{(a \rightarrow b) \rightarrow a} \cdot x^{(a \rightarrow b) \rightarrow a} U^{a \rightarrow b}$$

for some $U^{a \rightarrow b}$. Since $a \rightarrow b$ has just one premise, $U^{a \rightarrow b}$ must have form

$$U^{a \rightarrow b} \equiv \lambda y^a \cdot (w V_1 \dots V_r)^b \quad (r \geq 0).$$

Since M^τ is closed, w must be x or y . But w must have a type whose tail is an occurrence of b and neither x nor y has such a type, so no suitable $U^{a \rightarrow b}$ exists. Thus $Long(\tau) = \emptyset$ and hence $Nhabs(\tau) = \emptyset$ by 8A8.1. Hence $Habs(\tau) = \emptyset$ by 8A13(i). \square

8B7 Exercise* Verify the information in Table 8B7a by proving that the types shown there have no other inhabitants than those displayed. The notation used in the table's last row is:

$$\begin{aligned} P_0 &\equiv \lambda x \cdot x! \\ P_n &\equiv \lambda x \cdot x(\lambda y_1 \cdot x(\dots(\lambda y_n \cdot x!) \dots)) & (n \geq 1), \\ Q_{n,i} &\equiv \lambda x \cdot x(\lambda y_1 \cdot x(\dots(\lambda y_n \cdot x(\lambda z \cdot y_i)) \dots)) & (n \geq 1, 1 \leq i \leq n). \end{aligned}$$

Table 8B7a. *Some types and their normal inhabitants*

Type τ	$Nhabs(\tau)$	$Long(\tau)$	$Nprinc(\tau)$	Notes
1 a	None	None	None	See Ex. 8B5
2 $a \rightarrow a$	I	I	I	I $\equiv \lambda x \cdot x$
3 $a \rightarrow b \rightarrow a$	K	K	K	K $\equiv \lambda xy \cdot x$
4 $(a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b$	B	B	B	B $\equiv \lambda xyz \cdot x(yz)$
5 $(a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c$	C	C	C	C $\equiv \lambda xyz \cdot xzy$
6 $(a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$	S	S	S	S $\equiv \lambda xyz \cdot xz(yz)$ see Ex. 8B3
7 $(a \rightarrow a \rightarrow b) \rightarrow a \rightarrow b$	W	W	W	W $\equiv \lambda xy \cdot xyy$
8 $a \rightarrow \dots \rightarrow a \rightarrow a$ $\{m + 1a's, m \geq 2\}$	Π_1^m, \dots, Π_m^m	Π_1^m, \dots, Π_m^m	None	$\Pi_i^m \equiv \lambda x_1 \dots x_m \cdot x_i$ see Ex. 8B4
9 $(a \rightarrow a) \rightarrow a \rightarrow a$	I , $\bar{0}$, $\bar{1}$, $\bar{2}, \dots$	$\bar{0}$, $\bar{1}$, $\bar{2}, \dots$	$\bar{2}$, $\bar{3}, \dots$	\bar{n} $\equiv \lambda xy \cdot x^n y$
10 $(a \rightarrow b) \rightarrow a \rightarrow b$	I , $\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$ $\equiv \lambda xy \cdot xy$
11 $((a \rightarrow b) \rightarrow a) \rightarrow a$	None	None	None	τ is Peirce's law see Ex. 8B6
12 $((a \rightarrow a) \rightarrow a) \rightarrow a$	$P_0, P_n,$ $Q_{n,i}$	$P_0, P_n,$ $Q_{n,i}$	$P_n, Q_{n,i}$	$n \geq 1, 1 \leq i \leq n;$ see Ex. 8B7 for $P_0, P_n, Q_{n,i}$

8C The search algorithm

The search strategy introduced informally in the last section will be crystallised into a formal algorithm in this one. The treatment will be in full detail as a preparation for the correctness-proof at the end of the chapter, but the core strategy is very simple as the examples in the last section have shown, and the reader who prefers to stay on an informal level should omit the present section and move on to the counting algorithm in 8D.

The method of the search algorithm will be to first look for members of

$\text{Long}(\tau)$ with depth $d = 0$, then $d = 1$, etc. The search will output a sequence $\mathcal{A}(\tau, 0), \mathcal{A}(\tau, 1), \mathcal{A}(\tau, 2), \dots$ of finite sets of expressions that will serve as “approximations” to these members. Those in $\mathcal{A}(\tau, d)$ will look like typed β -nf’s with depth $\leq d$ but may have some “holes” to be filled in by the algorithm at a later stage. These holes will be represented by new symbols called *meta-variables* and the approximations will be called *nf-schemes*. (In Example 8B3 the expression on the right-hand side of (1) was in effect a nf-scheme and “ U ” and “ V ” were its meta-variables.)

To construct $\mathcal{A}(\tau, d+1)$ the approximations in $\mathcal{A}(\tau, d)$ will be extended by replacing their meta-variables by certain chosen nf-schemes with depth 1.

8C1 Definition (Nf-schemes) We assume given an infinite sequence of expressions called *meta-variables*, distinct from each other and from term-variables. Meta-variables will be denoted by

$$“V”, “V_1”, “V_2”, \dots$$

Nf-schemes are defined like terms in 1A1 except that they may contain meta-variables or term-variables or both, and must satisfy the following restrictions:

- (i) each nf-scheme is a β -nf without bound-variable clashes;
- (ii) meta-variables do not bind, i.e. λV is forbidden;
- (iii) in a composite nf-scheme meta-variables only occur in argument positions (as defined in 9A1);
- (iv) each meta-variable in a nf-scheme occurs only once.

Proper nf-schemes are those that contain at least one meta-variable.

8C1.1 Examples The expression $\lambda xyz \cdot (xV_1V_2)$ is a nf-scheme. So also is the atomic expression V . But the following are not nf-schemes:

$$\lambda V \cdot V, \quad Vxy, \quad \lambda x \cdot V.$$

8C1.2 Notes (i) The reader familiar with “context” notation will see that a nf-scheme with k meta-variables is essentially a context containing k different kinds of hole.

(ii) The restrictions in 8C1 are imposed simply because no wider class of expressions will be needed below.

(iii) Non-proper nf-schemes are just terms (in β -nf without bound-variable clashes).

(iv) In what follows, a *new* variable or meta-variable will be one that has not been used earlier in the chapter.

(v) Most of the term-notation introduced in Chapter 1 extends to nf-schemes in an obvious way, and the same holds for the definition of depth in 8A6. But a few concepts need defining afresh and the next three definitions will cover these.

8C2 Definition A *closed* nf-scheme is one containing no free variables. (But possibly containing meta-variables, for example $\lambda x \cdot xV$ and V are closed.)

8C3 Definition (Typed nf-schemes, $\text{TNS}(\Gamma)$) In this chapter a *type-context* Γ is any set of assignments whose subjects are meta-variables or variables, such that

no subject receives more than one type in Γ (cf. 2A5). The set $\text{TNS}(\Gamma)$ of all **typed nf-schemes relative to** Γ is defined just like $\text{TT}(\Gamma)$ in 5A1 but satisfying the restrictions in the definition of “nf-scheme” (8C1).

The typed-term notation in Chapter 5 will be used here for typed nf-schemes also.

8C4 Definition (Long typed nf-schemes) A typed nf-scheme X^τ is **long** iff each component of X^τ with form

$$(zY_1 \dots Y_n)^\sigma \quad (n \geq 0)$$

that is not in a function position has atomic type. (cf. Definition 8A7.)

8C4.1 Example The nf-scheme (i) below is long, though in contrast the term (ii) is not:

- (i) $x^{(a \rightarrow b) \rightarrow c} \vee a \rightarrow b,$
- (ii) $x^{(a \rightarrow b) \rightarrow c} z a \rightarrow b.$

The description of the search algorithm will now be given: it will begin with a precise statement of what the algorithm does (in the following theorem), followed by the algorithm itself.

8C5 Search Theorem for Long(τ) (Ben-Yelles 1979 §§3.14–3.17.) *The search algorithm in 8C6 below accepts as input any composite type τ and outputs a finite or infinite sequence of sets $\mathcal{A}(\tau, d)$ ($d = 0, 1, 2, \dots$) such that for all $d \geq 0$,*

- (i) *each member of $\mathcal{A}(\tau, d)$ is a closed long typed nf-scheme with type τ , and is either*
 - (a) *a proper nf-scheme with depth d , or*
 - (b) *a term with depth $d - 1$;*
- (ii) *$\mathcal{A}(\tau, d)$ is finite;*
- (iii) *$\text{Long}(\tau, d) \subseteq \mathcal{A}(\tau, 0) \cup \dots \cup \mathcal{A}(\tau, d + 1)$;*
- (iv) *if we call the set of all terms in $\mathcal{A}(\tau, d)$ “ $\mathcal{A}_{\text{terms}}(\tau, d)$ ”, then*

$$\text{Long}(\tau) = \bigcup_{d \geq 0} \mathcal{A}_{\text{terms}}(\tau, d)$$

8C5.1 Proof-note Parts (i)–(ii) will be proved by induction on d in the notes accompanying the steps of the search algorithm.

Part (iii) states that the algorithm is “complete”, i.e. finds everything it is looking for; its proof is postponed to 8F1.

Part (iv) follows easily from (i)–(iii).

8C6 Search Algorithm (Ben-Yelles 1979 §§3.14–17.) **Input:** any type τ . If τ is an atom it has no inhabitants (by 8B5). If τ is composite, proceed as follows.

Step 0. Choose any meta-variable V and define

$$\mathcal{A}(\tau, 0) = \{V^\tau\}.$$

Note. This $\mathcal{A}(\tau, 0)$ trivially satisfies 8C5(i)–(ii). (The algorithm may be seen as building approximations to an unknown term M^τ ; V^τ is the weakest approximation and represents the fact that at this stage we know nothing at all about M^τ other than its type.)

Step $d + 1$. Assume $\mathcal{A}(\tau, d)$ has been defined and satisfies 8C5(i)–(ii).

Substep I. If $\mathcal{A}(\tau, d) = \emptyset$ or no member of $\mathcal{A}(\tau, d)$ contains meta-variables then stop. (In this case $\mathcal{A}(\tau, d + 1)$ is undefined and the algorithm's output is just the finite sequence $\mathcal{A}(\tau, 0), \dots, \mathcal{A}(\tau, d)$.)

Substep II. Otherwise, begin the construction of $\mathcal{A}(\tau, d + 1)$ by listing the proper nf-schemes in $\mathcal{A}(\tau, d)$ and applying IIa–IIb below to each one.

Subsubstep IIa. Given any proper $X^\tau \in \mathcal{A}(\tau, d)$, list the meta-variables in X^τ ; say they are

$$V_1^{\rho_1}, \dots, V_q^{\rho_q} \quad (q \geq 1),$$

and apply IIa1–IIa2 to each one.

Part IIa1. Given any meta-variable V^ρ in an $X^\tau \in \mathcal{A}(\tau, d)$, say

$$(1) \quad \rho \equiv \sigma_1 \rightarrow \dots \rightarrow \sigma_m \rightarrow a \quad (m \geq 0);$$

first list all $i \leq m$ for which $\text{Tail}(\sigma_i) \cong a \cong \text{Tail}(\rho)$. (If there are none or $m = 0$, go direct to IIa2.) For each such i , σ_i has form

$$(2) \quad \sigma_i \equiv \sigma_{i,1} \rightarrow \dots \rightarrow \sigma_{i,n_i} \rightarrow a \quad (n_i \geq 0).$$

Define

$$(3) \quad Y_i^\rho \equiv \lambda x_1^{\sigma_1} \dots x_m^{\sigma_m} \cdot (x_i^{\sigma_i} V_{i,1}^{\sigma_{i,1}} \dots V_{i,n_i}^{\sigma_{i,n_i}})^a,$$

where the x 's and V 's are distinct new variables and meta-variables. (Y_i^ρ is called a **suitable replacement** for V^ρ .)

Part IIa2. List the abstractors that cover the (unique) occurrence of V^ρ in X^τ , in the order they occur in X^τ from left to right; say they are

$$(4) \quad \underline{\lambda} z_1^{\zeta_1}, \dots, \underline{\lambda} z_t^{\zeta_t} \quad (t \geq 0).$$

List all $j \leq t$ (if any) such that $\text{Tail}(\zeta_j) \cong a$. For each such j , ζ_j has form

$$(5) \quad \zeta_j \equiv \zeta_{j,1} \rightarrow \dots \rightarrow \zeta_{j,h_j} \rightarrow a \quad (h_j \geq 0).$$

Define

$$(6) \quad Z_j^\rho \equiv \lambda x_1^{\sigma_1} \dots x_m^{\sigma_m} \cdot (z_j^{\zeta_j} V_{j,1}^{\zeta_{j,1}} \dots V_{j,h_j}^{\zeta_{j,h_j}})^a,$$

where the x 's and V 's are distinct and new. (Z_j^ρ is called a **suitable replacement** for V^ρ .)

Notes. (i) It is easy to see that each Y_i^ρ defined in IIa1 and each Z_j^ρ defined in IIa2 is a long nf-scheme with depth ≤ 1 and the same type as V^ρ , so V^ρ can be replaced by Y_i^ρ or Z_j^ρ in X^τ without violating type-restrictions or the restrictions in the definition of long nf-scheme (8C1, 8C3–4). And the result of making such a replacement will clearly have depth $\leq d + 1$.

(ii) Y_i^ρ and Z_j^ρ need not contain meta-variables. In fact Y_i^ρ is without meta-variables iff σ_i is an atom, and in this case

$$Y_i^\rho \equiv \lambda x_1^{\sigma_1} \dots x_m^{\sigma_m} \cdot x_i^{\sigma_i}.$$

Similarly Z_j^ρ is without meta-variables iff ζ_j is an atom, and in this case

$$Z_j^\rho \equiv \lambda x_1^{\sigma_1} \dots x_m^{\sigma_m} \cdot z_j^{\zeta_j}.$$

Hence if Y_i^ρ or Z_j^ρ is without meta-variables its depth is 0 and if V^ρ is replaced by it then $\text{Depth}(X^\tau)$ will not increase.

(iii) The total number of suitable replacements (Y 's and Z 's) for V^ρ is at most $m + t$.

Subsubstep IIb. When IIa1–IIa2 have been applied to all the meta-variables in X^τ the result is a list of suitable replacements for each V_i in X^τ .

If one or more of V_1, \dots, V_q has no suitable replacements, abandon X^τ , calling it a **reject**, and start applying Substep II to the next member of $\mathcal{A}(\tau, d)$. (A reject will generate no members of $\mathcal{A}(\tau, d + 1)$.)

If all of V_1, \dots, V_q in X^τ have suitable replacements, X^τ is called **extendable**; in this case list all possible sequences

$$(7) \quad \langle W_1^{\rho_1}, \dots, W_q^{\rho_q} \rangle$$

where W_i is a suitable replacement for V_i for $i = 1, \dots, q$. For each sequence (7) construct a new nf-scheme $X^{*\tau}$ from X^τ by simultaneously replacing V_i by W_i in X^τ for $i = 1, \dots, q$. (Call each sequence (7) a **suitable multi-replacement** and call $X^{*\tau}$ an **extension** of X^τ ; if this extension is a term call it a **success**.)

Notes. (i) The number of extensions of X^τ is finite since each is generated by a suitable multi-replacement and the number of these is clearly finite (for a given X^τ).

(ii) To construct an extension each meta-variable in X^τ is replaced by either a nf-scheme with depth 1 or a term with depth 0. If the latter holds for all the meta-variables in X^τ the extension has depth $\leq d$ and is a success. If the former holds for at least one meta-variable, the extension has depth $\leq d + 1$ and contains meta-variables.

Substep III. Finally, if the set $\mathcal{A}(\tau, d)$ contains at least one proper nf-scheme, define $\mathcal{A}(\tau, d + 1)$ to be the set containing all the extensions of all the extendable proper nf-schemes in $\mathcal{A}(\tau, d)$.

Notes. (i) By the notes after IIa and IIb above it is easy to check that $\mathcal{A}(\tau, d + 1)$ satisfies 8C5(i) and (ii).

(ii) By the way, $\mathcal{A}(\tau, d + 1)$ does not contain $\mathcal{A}(\tau, d)$ as a subset. □

8C6.1 *Example* (cf. Example 8B3.) Applying the search algorithm to the type

$$\tau \equiv (a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$$

produces the following sets:

$$\begin{aligned}\mathcal{A}(\tau, 0) &= \{V^\tau\}, \\ \mathcal{A}(\tau, 1) &= \{\lambda x_1^{a \rightarrow b \rightarrow c} x_2^{a \rightarrow b} x_3^a \cdot x_1^{a \rightarrow b \rightarrow c} V_1^a V_2^b\}, \\ \mathcal{A}(\tau, 2) &= \{\lambda x_1^{a \rightarrow b \rightarrow c} x_2^{a \rightarrow b} x_3^a \cdot x_1^{a \rightarrow b \rightarrow c} x_3^a (x_2^{a \rightarrow b} V_3^a)\}, \\ \mathcal{A}(\tau, 3) &= \{\lambda x_1^{a \rightarrow b \rightarrow c} x_2^{a \rightarrow b} x_3^a \cdot x_1^{a \rightarrow b \rightarrow c} x_3^a (x_2^{a \rightarrow b} x_3^a)\}.\end{aligned}$$

8C6.2 *Example* (cf. Example 8B4.) Applying the search algorithm to the type

$$\tau \equiv a \rightarrow \dots \rightarrow a \rightarrow a$$

(with $m+1$ a 's) produces the following sets:

$$\begin{aligned}\mathcal{A}(\tau, 0) &= \{V^\tau\}, \\ \mathcal{A}(\tau, 1) &= \{(\lambda x_1^a \dots x_m^a \cdot x_1^a), \dots, (\lambda x_1^a \dots x_m^a \cdot x_m^a)\}.\end{aligned}$$

8C6.3 *Example* (cf. Example 8B6.) Applying the search algorithm to the type

$$\tau \equiv ((a \rightarrow b) \rightarrow a) \rightarrow a$$

produces the following sets:

$$\begin{aligned}\mathcal{A}(\tau, 0) &= \{V^\tau\}, \\ \mathcal{A}(\tau, 1) &= \{\lambda x_1^{(a \rightarrow b) \rightarrow a} \cdot x_1^{(a \rightarrow b) \rightarrow a} V_1^{a \rightarrow b}\}, \\ \mathcal{A}(\tau, 2) &= \emptyset.\end{aligned}$$

8C6.4 *Exercise** List members of the sets $\mathcal{A}(\tau, 0), \mathcal{A}(\tau, 1), \dots$ for all the types in Table 8B7a.

8C6.5 *Comments* (i) Besides Ben-Yelles' 1979 search algorithm there are others in Zaionc 1985 and 1988 and in Takahashi, Akama and Hirokawa 1994. Also procedures essentially equivalent to the main parts of 8C6 are used in some decision-algorithms for provability in intuitionist logic, cf. 6B7.3, and in algorithms that seek to unify pairs of typed λ -terms, for example the original one in Huet 1975 §3.

(ii) The algorithm in 8C6 can be viewed as a context-free grammar for generating a language whose expressions are exactly the inhabitants of τ . In the standard notation for context-free grammars (see for example Hopcroft and Ullman 1979 Ch. 4) the meta-variables in 8C6 would be called non-terminal variables and each suitable replacement defined in Subsubstep IIa of 8C6 would determine a production in the grammar. The set of all such productions need not be finite, however, so the original definition of "grammar" must be relaxed if this view is to be made precise. This view was taken in Huet 1975 §3, Zaionc 1985 §3 and 1988 §5, and was used and analysed in detail in Takahashi, Akama and Hirokawa 1994.

(iii) A very smooth description of a search algorithm in terms of solving sets of polynomial equations over the domain $\{0, 1, 2, \dots, \infty\}$ is in Zaionc 199-.

8D The Counting algorithm

In this section the search algorithm will be extended to count and enumerate normal inhabitants. The extension will be very simple indeed to state; but the proof that it works will need some care and will be postponed to Sections 8E–F.

As noted in 8A2, it does not matter whether typed or untyped inhabitants are counted since there is a one-to-one correspondence between the two: in this section “ $Long(\tau)$ ” and “ $Nhabs(\tau)$ ” will denote sets of typed terms.

8D1 Definition (i) For the sets $\mathcal{A}(\tau, d)$ and $\mathcal{A}_{\text{terms}}(\tau, d)$ introduced in 8C5, define

$$\begin{aligned}\mathcal{A}(\tau, \leq d) &= \mathcal{A}(\tau, 0) \cup \dots \cup \mathcal{A}(\tau, d), \\ \mathcal{A}_{\text{terms}}(\tau, \leq d) &= \mathcal{A}_{\text{terms}}(\tau, 0) \cup \dots \cup \mathcal{A}_{\text{terms}}(\tau, d).\end{aligned}$$

(ii) For any τ , recall from 2A2 that $|\tau|$ is the number of atom-occurrences in τ and $\|\tau\|$ is the number of distinct atoms in τ ; define

$$\mathbb{D}(\tau) = |\tau| \times \|\tau\|.$$

8D2 Stretching Lemma *If $Long(\tau)$ has a member M^τ with depth $d \geq \|\tau\|$ then it has members with depths greater than any given integer, and hence is infinite.*

Proof-note The proof will be given in 8F2 (cf. Ben-Yelles 1979 §§3.20–3.21). It will show that if M^τ is in $Long(\tau)$ and has depth $d \geq \|\tau\|$ then M^τ must contain two distinct components \underline{B} and \underline{B}' with the same type and with one inside the other. Then replacing the smaller one by the larger will change M^τ to a new term $M^{*\tau}$ deeper than M^τ , and it will be easy to check that $M^{*\tau}$ is a genuine typed term and is in $Long(\tau)$. Then a similar replacement in $M^{*\tau}$ will change it to a still deeper term, and so on.¹

8D3 Shrinking Lemma *If $Long(\tau)$ has a member M^τ with depth $\geq \mathbb{D}(\tau)$ then it has a member N^τ with*

$$\mathbb{D}(\tau) - \|\tau\| \leq \text{Depth}(N^\tau) < \mathbb{D}(\tau).$$

Proof-note The proof will be given in 8F3 (cf. Ben-Yelles 1979 Thm. 3.28). Just as for the previous lemma, M^τ will be seen to contain two components \underline{B} and \underline{B}' with the same type and one inside the other; but now $M^{*\tau}$ will be constructed by replacing the larger by the smaller. A problem is that if \underline{B} and \underline{B}' are not chosen carefully this replacement risks deleting some abstractors and giving a non-closed $M^{*\tau}$: to prevent this, and to ensure that the depth of $M^{*\tau}$ lies in the range required, \underline{B} and \underline{B}' will be chosen only after a close analysis of the structure of M^τ .²

¹ This lemma is very like the pumping lemma in the theory of context-free grammars and regular languages, see for example Hopcroft and Ullman 1979 §3.1 and Comment 8C6.5(ii) above. And the next lemma will also be very like one in that theory, see Hopcroft and Ullman 1979 §3.3 Theorem 3.7, though its proof here will be more complicated due to the presence of bound variables. For a precise statement of the analogy between typed λ -calculus and the theory of context-free grammars see Takahashi et al. 1994, where a generalised concept of context-free grammar is introduced and generalizations of the classical results are proved which imply the corresponding λ -results.

² Ben-Yelles 1979 Thm. 3.28, whose lines the proof in 8F3 will follow, was actually slightly weaker than the above lemma. The first proof of a full-strength shrinking lemma was outlined in Hirokawa 1993c; it was different from the one to be given in 8F3.

8D3.1 Corollary *If $\text{Long}(\tau)$ has a member M^τ with depth $\geq \mathbb{D}(\tau)$ then it has a member N^τ with*

$$\|\tau\| \leq \text{Depth}(N^\tau) < \mathbb{D}(\tau).$$

Proof If $\text{Long}(\tau)$ has a member then τ is composite by 8B5. Hence $|\tau| \geq 2$, so

$$\mathbb{D}(\tau) - \|\tau\| \geq 2\|\tau\| - \|\tau\| = \|\tau\|.$$

The result then follows by the shrinking lemma. \square

8D4 Theorem (Counting long normal inhabitants) (Ben-Yelles 1979 Cor. 3.33.) *When given a type τ the algorithm in 8D5 below outputs $\#(\text{Long}(\tau))$ and an enumeration of $\text{Long}(\tau)$.*

Proof See the notes in brackets in the following algorithm. \square

8D5 Counting Algorithm for $\text{Long}(\tau)$ *If τ is an atom, $\text{Long}(\tau)$ is empty by 8B5. If τ is composite, apply the search algorithm (8C6) to τ ; this outputs a finite or infinite sequence of sets*

$$\mathcal{A}(\tau, d) \quad (d = 0, 1, 2, \dots).$$

Stop the search algorithm at $d = \mathbb{D}(\tau)$ and enumerate $\mathcal{A}_{\text{terms}}(\tau, \leq \mathbb{D}(\tau))$.

[By 8C5 this set is finite and contains all members of $\text{Long}(\tau)$ with depth $< \mathbb{D}(\tau)$.]

Case I: $\mathcal{A}_{\text{terms}}(\tau, \leq \mathbb{D}(\tau)) = \emptyset$. Then $\text{Long}(\tau) = \emptyset$.

[By 8D3, if $\text{Long}(\tau)$ had a member with depth $\geq \mathbb{D}(\tau)$ it would have one with depth $< \mathbb{D}(\tau)$.]

Case II: $\mathcal{A}_{\text{terms}}(\tau, \leq \mathbb{D}(\tau))$ has a member with depth $\geq \|\tau\|$. Then by 8D2 $\text{Long}(\tau)$ is infinite. To enumerate $\text{Long}(\tau)$, apply the search algorithm to enumerate $\mathcal{A}_{\text{terms}}(\tau, d)$ for $d = 0, 1, 2, \dots$

[By 8C5(iv) the union of these sets is $\text{Long}(\tau)$.]

Case III: $\mathcal{A}_{\text{terms}}(\tau, \leq \mathbb{D}(\tau))$ has members but they all have depth $< \|\tau\|$. Then $\text{Long}(\tau) = \mathcal{A}_{\text{terms}}(\tau, \leq \mathbb{D}(\tau))$, which is finite.

[The only way for $\text{Long}(\tau)$ to differ from this set would be for $\text{Long}(\tau)$ to have members with depth $d \geq \mathbb{D}(\tau)$, but by 8D3.1 it would then have a member with $\|\tau\| \leq d < \mathbb{D}(\tau)$ contrary to the assumption of the present case.] \square

8D5.1 Corollary (Counting $\beta\eta$ -normal inhabitants) *The algorithm in 8D5 can be used to count and enumerate $N\text{habs}_\eta(\tau)$ for every τ .*

Proof By 8A10 the members of $N\text{habs}_\eta(\tau)$ are the η -nf's of those of $\text{Long}(\tau)$. And by 8A10(iii), $\#_\eta(\tau) = \#(N\text{habs}_\eta(\tau)) = \#(\text{Long}(\tau))$. \square

8D5.2 Corollary (Emptiness test) *The algorithm in 8D5 can be used to decide whether a type τ has no inhabitants.*

Proof By 8A13(i), $Habs(\tau) = \emptyset \Leftrightarrow \#(Nhabs(\tau)) = \#(\tau) = 0$. And by 8A10(ii) and (iii), $\#(\tau) = 0 \Leftrightarrow \#(Long(\tau)) = 0$. \square

8D6 Theorem (Counting all normal inhabitants) (Ben-Yelles 1979 Ch. 3.) *When given a type τ the algorithm in 8D7 below outputs $\#(\tau)$ and an enumeration of the set $Nhabs(\tau)$ of all the β -normal inhabitants of τ .*

8D7 Counting Algorithm for $Nhabs(\tau)$ *If τ is an atom then $Nhabs(\tau)$ is empty by 8B5. If τ is composite, apply Algorithm 8D5 to count $Long(\tau)$.*

Case I: *If $Long(\tau) = \emptyset$ then $Nhabs(\tau) = \emptyset$. [By 8A8.1.]*

Case II: *If $Long(\tau) \neq \emptyset$ then $Nhabs(\tau)$ is counted and enumerated by counting and enumerating $Long(\tau)$ and enumerating the η -family of each member of $Long(\tau)$.*

[By 8A10 these η -families are finite and their union is $Nhabs(\tau)$.]

Before ending this section we shall digress from the main theme to look at an interesting application of the counting algorithm to types that contain only one atom. Such types have been studied and used as tools by several workers, for example see Statman 1979a, 1979b and 1980, Dekkers 1988, and Zaionc 1987b and 1990.

8D8 Definition A type τ will be called *monatomic* iff $\|\tau\| = 1$, i.e. iff only one atom occurs in τ .

8D9 Theorem (Ben-Yelles 1979 §3.25.) *Let τ be a monatomic type with the form $\tau \equiv \tau_1 \rightarrow \dots \tau_m \rightarrow a$ ($m \geq 0$). Then*

- (i) *if at least one τ_i is composite, $\#(\tau)$ is either ∞ or 0;*
- (ii) *if $\tau_1 \equiv \dots \equiv \tau_m \equiv a$ then $\#(\tau) = m$.*

Proof Part (ii) is just 8B4. To prove (i), assume $Nhabs(\tau)$ is finite and non-empty and assume τ has form

$$\tau \equiv \tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow a$$

with $m \geq 1$ and at least one premise composite. By 8A8 $Long(\tau)$ is also finite and has a member M^τ . By 8D2, $Depth(M^\tau) < \|\tau\| = 1$, so M^τ must have form

$$M^\tau \equiv \lambda x_1^{\tau_1} \dots x_m^{\tau_m} \cdot x_p^{\tau_p} \quad (1 \leq p \leq m).$$

And $\tau_p \equiv a$ since M^τ is long. Choose a composite premise of τ ; say it is τ_i and has form

$$\tau_i \equiv \tau_{i,1} \rightarrow \dots \rightarrow \tau_{i,m_i} \rightarrow a \quad (m_i \geq 1).$$

For $j = 1, \dots, m_i$ each $\tau_{i,j}$ has form

$$\tau_{i,j} \equiv \tau_{i,j,1} \rightarrow \dots \rightarrow \tau_{i,j,m_{i,j}} \rightarrow a \quad (m_{i,j} \geq 0).$$

For each $j \leq m_i$ choose distinct new variables $y_1, \dots, y_{m_{i,j}}$ and define

$$P_j^{\tau_{i,j}} \equiv \lambda y_1^{\tau_{i,j,1}} \dots y_{m_{i,j}}^{\tau_{i,j,m_{i,j}}} \cdot x_p^a$$

where x_p^a is the rightmost variable in M^r (and is therefore distinct from the y 's). Then define

$$N^\tau \equiv \lambda x_1^{\tau_1} \dots x_m^{\tau_m} \cdot (x_i^{\tau_i} P_1^{\tau_{i,1}} \dots P_{m_i}^{\tau_{i,m_i}})^a.$$

Clearly $N^\tau \in \text{Long}(\tau)$. But $\text{Depth}(N^\tau) \geq 1$ since N^τ contains $m_i \geq 1$ arguments. Hence $\text{Long}(\tau)$ is infinite by 8D2. \square

8D10 Problems The above algorithms have solved the problem of counting normal inhabitants. But the following related problems are still open.

(i) *Counting principal normal inhabitants* ($N\text{princ}(\tau)$, $N\text{princ}_\eta(\tau)$). I know of no way to count these sets that covers the case when $N\text{habs}(\tau)$ is infinite. Hirokawa 1991a contains a structural analysis of principal deductions that may help. It is applied to give interesting results in Hirokawa 1991a–c, 1992b and 1993a–b, and these results include stating some sufficient conditions for $N\text{princ}(\tau)$ to have only one member.

(ii) *Counting principal inhabitants in general* ($\text{Princ}(\tau)$). Each member of $\text{Princ}(\tau)$ must have a β -nf and that nf must have type τ , but since β -reduction does not preserve principality there is no direct connection between $\text{Princ}(\tau)$ and $N\text{princ}(\tau)$. By the way, if $N\text{princ}(\tau) = \emptyset$ but $\text{Princ}(\tau) \neq \emptyset$ then τ has the curious property of being the principal type of a non-normal form but not of a normal form. (An example of such a type is $a \rightarrow a \rightarrow a$, see 8A13(iii).) Is there any neat characterization of the types that have this property?

(iii) *Counting inhabitants in restricted classes of terms*, for example the λ 1-terms, BCK λ -terms or BCI λ -terms introduced in 1D. The search algorithm would give no trouble in such a class but the crucial shrinking lemma (8D3) would need a new proof for each restricted class, and indeed might fail for some classes. An example of a result for a restricted class is the proof in Komori and Hirokawa 1993 that for BCK λ -terms $\#(\tau)$ is always finite. An example of a currently unsolved problem for a restricted class is that of finding a decision-procedure for the Hilbert-style logic T_\perp whose axioms are the principal types of **B**, **B'**, **I**, **W** (see 7D11(ii)). This is equivalent to finding an algorithm to decide whether a given type τ has any inhabitants that are applicative combinations of these four terms.

8D11 Comment (Efficiency) At first glance the algorithms in this section may seem remarkably efficient, as they terminate in only $|\tau| \times \|\tau\|$ steps. But this impression is totally misleading, as each one of these steps involves listing a set $\mathcal{A}(\tau, d)$ which, although finite, may be very large indeed. In fact any counting algorithm will decide as a special case whether $N\text{habs}(\tau)$ is non-empty, which is equivalent to deciding whether τ is provable in Intuitionist logic (by 6B7.2), and this decision problem is polynomial-space complete (Statman 1979a Prop. 4).

8E The structure of a nf-scheme

The earlier sections left the stretching and shrinking lemmas unproved (8D2, 8D3), as well as the “completeness” part of the search theorem for $\text{Long}(\tau)$ (8C5 (iii)). The aim of the next section will be to fill in these gaps, and the present one

8E4 Definition (Argument-branch) If \underline{Z} is a subargument of a typed or untyped nf-scheme X , the *argument-branch from \underline{X} to \underline{Z}* is the sequence

$$\langle \underline{Z}_0, \underline{Z}_1, \dots, \underline{Z}_k \rangle \quad (k \geq 1)$$

such that $\underline{Z}_0 \equiv \underline{X}$ and \underline{Z}_i is an argument of \underline{Z}_{i-1} for $i = 1, \dots, k$, and $\underline{Z}_k \equiv \underline{Z}$. It is called *unextendable* iff \underline{Z} is an atom or abstracted atom. Its *length* is k (not $k + 1$).

8E4.1 Lemma For any typed or untyped nf-scheme X :

(i) the depth in X of a subargument \underline{Z} is the same as the length of the argument-branch from \underline{X} to \underline{Z} ;

(ii) $\text{Depth}(X)$ is the maximum of the lengths of all the argument-branches in X .

Proof For (i) use induction on $|X|$; for (ii) use 8E3.1. □

8E5 Definition (IA, CA) Let \underline{Z} be a subargument of a typed or untyped nf-scheme X ; say

$$\underline{Z} \equiv \lambda x_1 \dots x_m \cdot y Z_1 \dots Z_n \quad (m \geq 0, n \geq 0).$$

The *Initial Abstractors sequence* $IA(\underline{Z})$ is the (possibly empty) sequence

$$IA(\underline{Z}) = \langle x_1, \dots, x_m \rangle.$$

The *Covering Abstractors sequence* $CA(\underline{Z}, X)$ is defined to be

$$CA(\underline{Z}, X) = \langle z_1, \dots, z_q \rangle,$$

where $\underline{\lambda}z_1, \dots, \underline{\lambda}z_q$ are the abstractors in X whose scopes contain \underline{Z} , written in the order they occur in X from left to right. Also define:

$$\text{Length}(IA(\underline{Z})) = m, \quad \text{Length}(CA(\underline{Z}, X)) = q.$$

8E5.1 Note (i) If X has no bound-variable clashes the member of $IA(\underline{Z})$ are distinct and so are those of $CA(\underline{Z}, X)$.

(ii) $IA(\underline{Z})$ and $CA(\underline{Z}, X)$ are sequences of variables not components.

(iii) For typed nf-schemes each variable in $IA(\underline{Z})$ or $CA(\underline{Z}, X)$ is typed.

(iv) If the argument-branch from \underline{X} to \underline{Z} is $\langle \underline{Z}_0, \dots, \underline{Z}_k \rangle$ ($k \geq 1$), then

$$CA(\underline{Z}, X) = IA(\underline{Z}_0) * \dots * IA(\underline{Z}_{k-1})$$

where “ $*$ ” denotes concatenation of sequences. (Because the abstractors whose scopes contain \underline{Z} are exactly the initial abstractors of $\underline{Z}_0, \dots, \underline{Z}_{k-1}$.)

The next definition and lemma will have meaning for typed nf-schemes only. The lemma will be the strengthened form of the subformula property mentioned at the start of the section, and will lead to a computation of upper bounds for several key sequences and sets. Part of it will use notation from 9E.

8E6 Definition (IAT) Let \underline{Z}^σ be a subargument of a typed nf-scheme X^τ ; say

$$\underline{Z}^\sigma \equiv \lambda x_1^{\sigma_1} \dots x_m^{\sigma_m} \cdot y Z_1 \dots Z_n \quad (m \geq 0, n \geq 0).$$

The *Initial Abstractors' Types sequence* $IAT(Z^\sigma)$ is defined to be

$$IAT(Z^\sigma) = \langle \sigma_1, \dots, \sigma_m \rangle;$$

also define

$$Length(IAT(Z^\sigma)) = m.$$

8E7 Enhanced Subformula Lemma (cf. Ben-Yelles 1979 Lemma 3.31.) *If \underline{Z}^σ is a subargument of a closed long typed nf-scheme X^τ , then*

- (i) σ occurs as a positive subpremise in τ (as defined in 9E6–8),
- (ii) if σ is an atom, $IAT(Z^\sigma) = \emptyset$,
- (iii) if σ is composite, $IAT(Z^\sigma) \in NSS(\tau)$ (defined in 9E9),
- (iv) $NSS(\sigma) \subseteq NSS(\tau)$.

Proof Since Z^σ is long, $IAT(Z^\sigma)$ coincides with the sequence of all premises of σ , so (ii) holds. Also if σ is composite we have

$$(1) \quad IAT(Z^\sigma) \in NSS(\sigma)$$

by the definition of $NSS(\sigma)$ in 9E9. Now (i) implies (iv) by 9E9.2(iii), and (iv) and (1) imply (iii). Hence only (i) remains to be proved.

The proof of (i) is an induction on $|X^\tau|$. To make this work we shall prove

$$(2) \quad \left\{ \begin{array}{l} \text{If } X^\tau \text{ is a long member of } TNS(\Gamma) \text{ (defined in 8C3) and} \\ \quad \Gamma = \{u_1:\theta_1, \dots, u_p:\theta_p, V_1:\phi_1, \dots, V_q:\phi_q\} \\ \text{and } \underline{Z}^\sigma \text{ is a subargument of } X^\tau, \text{ then } \sigma \text{ occurs as a positive} \\ \text{subpremise of} \end{array} \right. \quad \theta_1 \rightarrow \dots \rightarrow \theta_p \rightarrow \tau.$$

Basis. If X^τ is an atom the conclusion of (2) holds vacuously.

Induction step. Let X^τ have form

$$(3) \quad (\lambda x_1^{\tau_1} \dots x_m^{\tau_m} \cdot (y^{(\rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow e)} X_1^{\rho_1} \dots X_n^{\rho_n})^e)(\tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow e)$$

where $m, n \geq 0$ and $\tau \equiv \tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow e$. Then either $y \equiv x_i$ for some $i \leq m$ or $y \equiv u_i$ for some $i \leq p$. If $y \equiv x_i$ then

$$(4) \quad \tau_i \equiv \rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow e$$

and if $y \equiv u_i$ then

$$(5) \quad \theta_i \equiv \rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow e.$$

In both cases each of ρ_1, \dots, ρ_n occurs as a positive subpremise of

$$(6) \quad \theta_1 \rightarrow \dots \rightarrow \theta_p \rightarrow \tau.$$

Now \underline{Z}^σ must be in an $\underline{X}_j^{\rho_j}$ for some $j \leq n$. If $\underline{Z}^\sigma \equiv \underline{X}_j^{\rho_j}$ then $\sigma \equiv \rho_j$ and the conclusion of (2) follows by the above. Next, suppose \underline{Z}^σ is a subargument of $\underline{X}_j^{\rho_j}$. Note that

$$X_j^{\rho_j} \in TNS(\{x_1:\tau_1, \dots, x_m:\tau_m\} \cup \Gamma).$$

Hence, by the induction hypothesis, σ occurs as a positive subpremise of

$$\tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow \theta_1 \rightarrow \dots \rightarrow \theta_p \rightarrow \rho_j.$$

Thus σ occurs as a positive subpremise of (6), giving (2). □

8E7.1 *Corollary* If X^τ is a closed long typed nf-scheme, the type of each meta-variable in X^τ either occurs as a positive subpremise of τ or is τ itself.

Proof By 8E2.2(i) and 8E7(i). □

The main effect of 8E7 is to connect $IAT(Z^\sigma)$, which in general depends on the structure of Z^σ and hence implicitly on that of X^τ , with $NSS(\tau)$ which depends on τ and nothing else. The next corollary will use this to deduce reasonably neat bounds for $IA(\underline{Z}^\sigma)$ and $CA(\underline{Z}^\sigma, X^\tau)$.

8E7.2 *Corollary* If X^τ is a closed long typed nf-scheme and \underline{Z}^σ is a subargument of X^τ or $\underline{Z}^\sigma \equiv \underline{X}^\tau$, then

- (i) $Length(IA(Z^\sigma)) = Length(IAT(Z^\sigma)) \leq |\tau| - 1$,
- (ii) $Length(CA(\underline{Z}^\sigma, X^\tau)) \leq (|\tau| - 1) \times Depth(X^\tau)$.

Further, if $\underline{\lambda v}_1^{\rho_1}, \dots, \underline{\lambda v}_r^{\rho_r}$ are all the abstractors in X^τ (not just its initial ones), then

- (iii) $\{\rho_1, \dots, \rho_r\}$ has $\leq |\tau| - 1$ distinct members.

Proof For (i): $Length(IAT(Z^\sigma)) \leq |\tau| - 1$ by 8E7(iii) and 9E9.3(iv).

For (ii): If $\underline{Z} \equiv \underline{X}$ the left side of (ii) is 0. If $\underline{Z} \not\equiv \underline{X}$ let $\langle \underline{Z}_0, \dots, \underline{Z}_k \rangle$ ($k \geq 1$) be the argument-branch from \underline{X} to \underline{Z} ; then by 8E5.1(iv)

$$\begin{aligned} Length(CA(\underline{Z}, X)) &= Length(IA(Z_0)) + \dots + Length(IA(Z_{k-1})) \\ &\leq k(|\tau| - 1) \end{aligned}$$

by (i). But $Depth(X) \geq k$ by 8E4.1(ii), so (ii) holds.

For (iii): Each ρ_i is in $IAT(X^\tau)$ or in $IAT(Y^\theta)$ for some subargument Y^θ of X^τ ; and in both cases $\rho_i \in \bigcup NSS(\tau)$ (in the former case trivially, and in the latter case by 8E7(iii)). Then use 9E9.3(iii). □

8E7.3 *Exercise** Show that if τ is composite and $d \geq 1$ and $\mathcal{A}(\tau, d)$ is defined, then

- (i) each $X^\tau \in \mathcal{A}(\tau, d)$ contains $\leq |\tau|^d$ meta-variables,
- (ii) $\#(\mathcal{A}(\tau, d)) < (d \times |\tau|)^{(|\tau|^{d-1})} \times \#(\mathcal{A}(\tau, d-1))$,
- (iii) $\#(\mathcal{A}(\tau, d)) < 1 \times 2^{|\tau|} \times 3^{(|\tau|^2)} \times \dots \times d^{(|\tau|^{d-1})} \times |\tau|^{(1+|\tau|+|\tau|^2+\dots+|\tau|^{d-1})}$.

8F Stretching, shrinking and completeness

This section fills in the three gaps that were left in the verification of the counting algorithm in 8C–D: the stretching and shrinking lemmas and the “completeness” part of the search theorem.

8F1 Search-Completeness Lemma Part (iii) of the search theorem 8C5 holds; i.e. if τ is composite and $d \geq 0$, then

$$Long(\tau, d) \subseteq \mathcal{A}(\tau, \leq d + 1).$$

Proof The lemma will be proved by induction on d but to make the induction work we must prove something slightly stronger. Recall that $Long(\tau, d)$ is the set of all long inhabitants of τ with depth $\leq d$ (8A7.2). Let $\mathbb{L}^*(\tau, d)$ be the set of all long typed closed nf-schemes X^τ such that $Depth(X^\tau) = d$ and

- (1) $\left\{ \begin{array}{l} \text{(a) } X^\tau \text{ is proper and all its meta-variables have depth } d \text{ in } X^\tau, \\ \text{(b) all subarguments with depth } d \text{ in } X^\tau \text{ are meta-variables.} \end{array} \right.$

We shall prove both the inclusions

- (2) $\mathbb{L}^*(\tau, d) \subseteq \mathcal{A}(\tau, \leq d),$
 (3) $Long(\tau, d) \subseteq \mathcal{A}(\tau, \leq d + 1),$

where (2) is understood modulo renaming of meta-variables (i.e. (2) says that if $X^\tau \in \mathbb{L}^*(\tau, d)$ then $\mathcal{A}(\tau, \leq d)$ contains either X^τ or a nf-scheme that differs from X^τ only by replacing its meta-variables by distinct others.)

Basis: $d = 0$. For (2): the only nf-schemes with depth 0 are meta-variables; also $\mathcal{A}(\tau, 0) = \{V^\tau\}$ by Step 0 of the search algorithm (8C6), so (2) holds modulo renaming.

For (3): let $\tau \equiv \tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow e$ ($m \geq 1$), and let $M^\tau \in Long(\tau, 0)$. Then M^τ is a term and has form

$$(4) \quad \lambda y_1^{\tau_1} \dots y_m^{\tau_m} \cdot y_i^{\tau_i} \quad (1 \leq i \leq m, \tau_i \equiv e).$$

Now $\mathcal{A}(\tau, 0) = \{V^\tau\}$. To construct $\mathcal{A}(\tau, 1)$ apply 8C6 Step 1: Part IIa1 therein will output (4) as a suitable replacement for V^τ because the tail of τ_i is isomorphic to that of τ . Hence $\mathcal{A}(\tau, 1)$ is defined and contains M^τ .

Induction step: d to $d + 1$. For (2): let $X \in \mathbb{L}^*(\tau, d + 1)$. (Types will not be displayed from now on.) Then $Depth(X) = d + 1$, so by 8E3.1(ii) X has a subargument whose depth in X is $d + 1$, and so by 8E4.1 X has one whose depth in X is d . List all such subarguments (without repetitions); say they are $\underline{W}_1, \dots, \underline{W}_r$ ($r \geq 1$). Clearly $\underline{W}_1, \dots, \underline{W}_r$ are disjoint components since they all have the same depth d in X ; and since $Depth(X) = d + 1$ we have $Depth(W_i) \leq 1$ for each i . Also since X satisfies (1a) and (1b) relative to $d + 1$, no W_i can be a meta-variable; hence each W_i must have form

$$(5) \quad W_i \equiv \lambda x_{i,1} \dots x_{i,m_i} \cdot y_i V_{i,1} \dots V_{i,n_i} \quad (m_i, n_i \geq 0).$$

Let X' be the result of replacing each \underline{W}_i in X by a distinct new meta-variable \underline{V}_i with the same type as \underline{W}_i .

Then X' is a nf-scheme. (It satisfies restriction (iii) in Definition 8C1 because each \underline{W}_i is a subargument.) And it is clearly long and closed and has depth d . Also, by its construction X' satisfies condition (1b) for membership of $\mathbb{L}^*(\tau, d)$. It also satisfies (1a), because if it contained a meta-variable-occurrence \underline{V} at a depth $< d$ such a \underline{V} could not be a \underline{V}_i and hence would occur also in X at a depth $< d$, contrary to the assumption that X satisfies (1a) relative to $d + 1$.

Hence $X' \in \mathbb{L}^*(\tau, d)$. Therefore by the induction hypothesis there is an $X'' \in \mathcal{A}(\tau, \leq d)$ that is identical to X' except perhaps for alphabetic variation of meta-variables. Apply Step $d + 1$ of Algorithm 8C6 to each V_i in X'' . Since W_i has form (5) and is part of a closed nf-scheme, namely X , it is easy to see that \underline{W}_i is a suitable replacement for \underline{V}_i (modulo renamings in W_i). Hence the algorithm will give X as an extension of X'' , so $X \in \mathcal{A}(\tau, \leq d + 1)$, giving (2) for $d + 1$.

For (3): let $M \in \text{Long}(\tau, d+1)$. Then by 8E3.1 M has a subargument whose depth in M is $d+1$. List all such subarguments without repetitions; say they are $\underline{U}_1, \dots, \underline{U}_r$ ($r \geq 1$). Clearly $\underline{U}_1, \dots, \underline{U}_r$ are disjoint; and since $\text{Depth}(M) = d+1$ each \underline{U}_i must have depth 0 and hence must have form

$$(6) \quad U_i \equiv \lambda x_{i,1} \dots x_{i,m_i}. y_i \quad (m_i \geq 0).$$

Define M' to be the result of replacing each \underline{U}_i in M by a distinct new meta-variable \underline{V}_i with the same type as \underline{U}_i . Then M' is a genuine nf-scheme. And it is clearly long and closed and has depth $d+1$. Also M' satisfies condition (1b) for membership of $\mathbb{L}^*(\tau, d+1)$, and satisfies (1a) because all its meta-variables have been introduced by the above replacements. Hence $M' \in \mathbb{L}^*(\tau, d+1)$.

Therefore by (2) for $d+1$ there is an M'' , differing from M' only by renaming meta-variables, such that

$$M'' \in \mathcal{A}(\tau, \leq d+1).$$

Apply Step $d+1$ of Algorithm 8C6 to M'' : it is easy to check that \underline{U}_i is a suitable replacement for \underline{V}_i in M'' , so the algorithm will give M as an extension of M'' . Hence $M \in \mathcal{A}(\tau, \leq d+2)$, proving (3) for $d+1$. \square

8F2 Detailed Stretching Lemma (cf. 8D2.) *If Long (τ) has a member M^τ with depth $\geq \|\tau\|$ then*

- (i) *there exists $M^{*\tau} \in \text{Long}(\tau)$ with $\text{Depth}(M^{*\tau}) \geq \text{Depth}(M^\tau) + 1$,*
- (ii) *Long (τ) is infinite.*

Proof [Ben-Yelles 1979 §§3.20–3.21.] We shall prove (i) and then (ii) will follow by repetition. Types of typed terms will be omitted.

To prove (i), let M be a typed closed long β -nf with type τ and without bound-variable clashes. Let $d = \text{Depth}(M) \geq \|\tau\| \geq 1$. By 8E4.1, M has at least one argument-branch with length d . Choose any such branch, call it

$$(1) \quad \langle \underline{N}_0, \dots, \underline{N}_d \rangle,$$

where $\underline{N}_0 \equiv \underline{M}$ and \underline{N}_{i+1} is an argument of \underline{N}_i for $i = 0, \dots, d-1$. Each of $\underline{N}_0, \dots, \underline{N}_d$ must have form

$$(2) \quad N_i \equiv \lambda x_{i,1} \dots x_{i,m_i}. y_i P_{i,1} \dots P_{i,n_i} \quad (m_i, n_i \geq 0),$$

and for $i \leq d-1$ we have $\underline{N}_{i+1} \equiv \underline{P}_{i,k_i}$ for some $k_i \leq n_i$ (and hence $n_i \geq 1$). Since $d = \text{Depth}(M)$ the last member of (1) has no arguments, so $n_d = 0$.

For $i = 0, \dots, d$ let \underline{B}_i be the body of \underline{N}_i ; that is

$$(3) \quad \underline{B}_i \equiv \underline{y}_i \underline{P}_{i,1} \dots \underline{P}_{i,n_i}.$$

Since \underline{N}_i is long, the type of \underline{B}_i is an atom. And this atom occurs in τ by 2B3(i). But $\|\tau\| \leq d$ and there are $d+1$ components $\underline{B}_0, \dots, \underline{B}_d$, so at least two of these must have the same type.

Choose any pair $\langle p, p+r \rangle$ such that \underline{B}_p and \underline{B}_{p+r} have the same type (and $r \geq 1$). Note that \underline{B}_p properly contains \underline{B}_{p+r} and

$$\text{Depth}(\underline{B}_p) \geq r + \text{Depth}(\underline{B}_{p+r}).$$

Define M^* to be the result of replacing \underline{B}_{p+r} in M by a copy of \underline{B}_p (after changing bound variables in this copy to avoid clashes). Then M^* has an argument-branch with length $d + r$. (Its members are

$$\underline{N}_0^*, \dots, \underline{N}_{p+r}^*, \underline{N}_{p+1}^\circ, \dots, \underline{N}_d^\circ,$$

where for $i = 0, \dots, p + r$ each \underline{N}_i^* has the same position in M^* as \underline{N}_i had in M , and for $j = p + 1, \dots, d$ we have $\underline{N}_j^\circ \equiv \underline{N}_j$.) Hence by 8E4.1,

$$\text{Depth}(M^*) \geq d + r \geq d + 1.$$

To complete the proof of (i) it only remains to show that M^* is a genuine typed term. This will be done by applying Lemma 5B2.1(ii) on replacement in typed terms.

First, for $i = 0, \dots, d$ let Γ_i be the context that assigns to the initial abstractors of N_i the types they have in M . Since M has no bound-variable clashes the variables in $\Gamma_0, \dots, \Gamma_d$ are all distinct, so

$$(4) \quad \Gamma_0 \cup \dots \cup \Gamma_d \text{ is consistent.}$$

Also every variable free in B_i is bound in one of N_0, \dots, N_i because M is closed and \underline{B}_i is in \underline{N}_i . Hence, by the definitions of typed term (5A1) and *Con* (5A4),

$$(5) \quad B_i \in \mathbb{T}(\Gamma_0 \cup \dots \cup \Gamma_i), \quad \text{Con}(B_i) \subseteq \Gamma_0 \cup \dots \cup \Gamma_i.$$

To apply 5B2.1(ii) it is enough to show that the set

$$(6) \quad \text{Con}(B_p) \cup \text{Con}(M) \cup \Gamma_0 \cup \dots \cup \Gamma_{p+r}$$

is consistent. (The abstractors in M whose scopes contain \underline{B}_{p+r} are exactly the initial abstractors of N_0, \dots, N_{p+r} .) But M is closed, so $\text{Con}(M) = \emptyset$. And by (5),

$$\text{Con}(B_p) \subseteq \Gamma_0 \cup \dots \cup \Gamma_p \subseteq \Gamma_0 \cup \dots \cup \Gamma_{p+r}.$$

Thus (6) is consistent by (4). □

8F3 Detailed Shrinking Lemma (cf. 8D3.) *If $\text{Long}(\tau)$ has a member M^τ with depth $\geq \mathbb{D}(\tau)$ then*

(i) *it has a member $M^{*\tau}$ with*

$$\text{Depth}(M^\tau) - \|\tau\| \leq \text{Depth}(M^{*\tau}) < \text{Depth}(M^\tau),$$

(ii) *it has a member N^τ with*

$$\mathbb{D}(\tau) - \|\tau\| \leq \text{Depth}(N^\tau) < \mathbb{D}(\tau).$$

Proof [cf. Ben-Yelles 1979 Thm. 3.28.] Part (ii) is proved by repeating (i) and taking the first output with depth $< \mathbb{D}(\tau)$.

Part (i) is proved as follows. (Types of typed terms will be omitted.) Let M be a member of $\text{Long}(\tau)$ without bound-variable clashes. Let $d = \text{Depth}(M) \geq \mathbb{D}(\tau)$. Incidentally $\mathbb{D}(\tau) \geq 2$ because τ is composite since atomic types have no inhabitants.

By 8E4.1, M has at least one argument-branch with length d ; to reduce the depth of M we must shrink all these branches. Consider any such branch; just as in the proof of the stretching lemma 8F2 it has form

$$(1) \quad \langle N_0, \dots, N_d \rangle,$$

where $\underline{N}_0 \equiv \underline{M}$ and \underline{N}_{i+1} is an argument of \underline{N}_i for $i = 0, \dots, d-1$. And

$$(2) \quad N_i \equiv \lambda x_{i,1} \dots x_{i,m_i} \cdot y_i P_{i,1} \dots P_{i,n_i} \quad (m_i, n_i \geq 0)$$

for $0 \leq i \leq d-1$. Let the type of N_i be

$$(3) \quad \rho_i \equiv \rho_{i,1} \rightarrow \dots \rightarrow \rho_{i,m_i} \rightarrow a_i.$$

Then since \underline{N}_i is long, the types of $x_{i,1}, x_{i,2}, \dots$ are exactly $\rho_{i,1}, \rho_{i,2}, \dots$; that is, using the “*IAT*” notation introduced in 8E6,

$$IAT(N_i) = \langle \rho_{i,1}, \dots, \rho_{i,m_i} \rangle.$$

Just as in the proof of 8F2 let \underline{B}_i be the body of \underline{N}_i for $i = 0, \dots, d$. Then the type of \underline{B}_i is the tail of the type of \underline{N}_i , namely a_i .

Define a sequence of integers d_0, d_1, \dots thus: $d_0 = 0$ and d_{j+1} is the least $i > d_j$ such that $IAT(N_i)$ differs from all of

$$(4) \quad IAT(N_{d_0}), IAT(N_{d_1}), \dots, IAT(N_{d_j}).$$

Let n be the greatest integer such that d_n is defined. The branch (1) has only d members after \underline{N}_0 , so $n \leq d$ and $d_n \leq d$. Then

$$(5) \quad 0 = d_0 < d_1 < \dots < d_n \leq d,$$

and for $0 \leq i \leq d$, $IAT(N_i)$ is identical to one of

$$(6) \quad IAT(N_{d_0}), IAT(N_{d_1}), \dots, IAT(N_{d_n}).$$

Also the $n+1$ *IAT*'s in (6) are distinct, and by 8E7 they are either empty or members of $NSS(\tau)$. Hence by 9E9.3(ii),

$$(7) \quad n+1 \leq |\tau|.$$

Now d_0, \dots, d_n partition the set $\{0, 1, \dots, d\}$ into the following $n+1$ non-empty sets which will be called *IAT-intervals*:

$$\begin{aligned} \mathbb{I}_j &= \{d_j, d_j + 1, \dots, d_{j+1} - 1\} & (0 \leq j \leq n-1), \\ \mathbb{I}_n &= \{d_n, d_n + 1, \dots, d\}. \end{aligned}$$

If \mathbb{I}_j contains two numbers $p, p+r$ such that $r \geq 1$ and B_p and B_{p+r} have the same type (i.e. $a_p \equiv a_{p+r}$) we shall call $\langle p, p+r \rangle$ a *tail-repetition*. It will be called *minimal* iff there is no other tail-repetition $\langle p', q' \rangle$ with

$$p \leq p' < q' \leq p+r.$$

Now an \mathbb{I}_j that contains no tail-repetitions must have $\leq \|\tau\|$ members. Because for such an \mathbb{I}_j the atoms

$$a_{d_j}, \dots, a_{d_{j+1}-1}$$

must all be distinct, and by (3) each a_i occurs in ρ_i , which occurs in τ by 8E7, and there are only $\|\tau\|$ distinct atoms in τ . This argument also shows that for a minimal tail-repetition $\langle p, p+r \rangle$ we have

$$(8) \quad r \leq \|\tau\|.$$

Now there are $n + 1$ IAT-intervals in the given branch and $n + 1 \leq |\tau|$ by (7), so if no interval contained a tail-repetition the branch would have $\leq |\tau| \times \|\tau\|$ members. But the branch has $d + 1$ members and

$$d + 1 = \text{Depth}(M) + 1 \geq \mathbb{D}(\tau) + 1 > |\tau| \times \|\tau\|.$$

Hence at least one IAT-interval contains a tail-repetition.

We start to build M^* as follows. In the given branch take the last \mathbb{I}_j containing a tail-repetition, choose a minimal tail-repetition $\langle p, p + r \rangle$ in it, and change M to a new term M' by replacing B_p by B_{p+r} .

To see that M' is a genuine typed term we apply 5B2.1(ii) similarly to the proof of 8F2. Using the notation of (4) in that proof, it is enough to show that the set

$$(9) \quad \text{Con}(B_{p+r}) \cup \text{Con}(M) \cup \Gamma_0 \cup \dots \cup \Gamma_p$$

is consistent. But, as in the proof of 8F2, $\text{Con}(M) = \emptyset$ and $\text{Con}(B_{p+r})$ is a subset of $\Gamma_0 \cup \dots \cup \Gamma_{p+r}$, so (9) is consistent by (4) in the proof of 8F2.

It is straightforward to prove that M' is a long β -nf with the same type as M . Also that $|M'| < |M|$.

But M' might not be closed, because the change from M to M' has removed the initial abstractors of $\underline{N}_{p+1}, \dots, \underline{N}_{p+r}$ from M , and so some free variable-occurrences in \underline{B}_{p+r} that were bound in M might now be free in M' . To close M' , apply the following procedure to every such variable-occurrence.

Let \underline{v} be free in the occurrence of B_{p+r} in M' that has replaced \underline{B}_p in M , and let v be also free in M' . Then v does not occur in a covering abstractor of this occurrence of B_{p+r} in M' . But these covering abstractors are exactly the initial abstractors of $\underline{N}_0, \dots, \underline{N}_p$ in M , so

$$v \notin \text{IA}(\underline{N}_0) \cup \dots \cup \text{IA}(\underline{N}_p).$$

But M is closed, so the free \underline{v} in \underline{B}_{p+r} in M must be in the scope of a $\lambda \underline{v}$ in one of $\text{IA}(\underline{N}_0), \dots, \text{IA}(\underline{N}_{p+r})$. Hence v occurs in $\text{IA}(\underline{N}_h)$ for some h with $p + 1 \leq h \leq p + r$; in the notation of (2) we have

$$v \equiv x_{h,k}$$

for some $k \leq m_h$. And the type of v is $\rho_{h,k} \in \text{IAT}(\underline{N}_h)$. Now by the definition of d_0, \dots, d_n , and the fact that the tail-repetition $\langle p, p + r \rangle$ we are eliminating is in an interval \mathbb{I}_j , $\text{IAT}(\underline{N}_h)$ coincides with one of

$$\text{IAT}(\underline{N}_{d_0}), \dots, \text{IAT}(\underline{N}_{d_j});$$

say $\text{IAT}(\underline{N}_h) = \text{IAT}(\underline{N}_{d_q})$ for some $q \leq j$. Hence there is a variable

$$x_{d_q,k} \in \text{IA}(\underline{N}_{d_q})$$

with the same type as v . Replace v by this variable throughout M' . The result will be a long β -nf with the same type and depth as M' and containing one less free variable.

Similarly replace every variable of B_{p+r} that is free in M' by a new one which has the same type but is bound in M' . The result will be a long β -nf M'' with the same type and depth as M' and which is closed.¹

Now M'' has been obtained by removing a type-repetition from an argument-branch in M which originally contained d subarguments. And by (8) the number of subarguments removed is $\leq \|\tau\|$. Hence

$$(10) \quad d - \|\tau\| \leq \text{Depth}(M'') \leq d.$$

If $\text{Depth}(M'') < d$, define $M^* \equiv M''$. If not, select a branch in M'' with length d and apply the above removal procedure to it, then continue shortening branches with length d until there are none left. (This process must terminate because each removal strictly reduces $|M|$.) Define M^* to be the first term produced by this procedure whose depth is less than d . Then

$$d - \|\tau\| \leq \text{Depth}(M^*) < d$$

as required. □

8F3.1 Example Let $\tau \equiv (a \rightarrow a) \rightarrow a \rightarrow a$, and let M^τ be a typed version of the Church numeral for the number four, namely

$$M^\tau \equiv (\lambda u^{a \rightarrow a} v^a. (u(u(u(uv))))^\tau).$$

Then

$$\|\tau\| = 1, \quad |\tau| = 4, \quad \mathbb{D}(\tau) = |\tau| \times \|\tau\| = 4.$$

And $\text{Depth}(M) = 4$, so the above shrinking procedure can be applied to M . There is only one argument-branch in M containing four subarguments, and its members are

$$\underline{\lambda uv} \cdot \underline{u^4 v}, \quad \underline{u^3 v}, \quad \underline{u^2 v}, \quad \underline{uv}, \quad \underline{v}.$$

(Call them $\underline{N}_0, \dots, \underline{N}_4$ respectively.) And

$$\begin{aligned} IAT(N_0) &= \langle a \rightarrow a, a \rangle \\ IAT(N_1) &= IAT(N_2) = IAT(N_3) = IAT(N_4) = \emptyset. \end{aligned}$$

Thus the only change in $IAT(N_i)$ comes at $i = 1$, so in the notation of the proof of 8F3 we have

$$n = 1, \quad d_0 = 0, \quad d_1 = 1, \quad \mathbb{I}_0 = \{0\}, \quad \mathbb{I}_1 = \{1, 2, 3, 4\}.$$

There are three minimal tail-repetitions in \mathbb{I}_1 and the last one is $\langle 3, 4 \rangle$. The proof of 8F3 removes this by replacing \underline{uv} by \underline{v} ; this changes M to

$$M^* \equiv \lambda uv \cdot u^3 v.$$

And $\text{Depth}(M^*) = 3 < \mathbb{D}(\tau)$.

¹ No claim is made that M'' is related to M' by α -conversion or any other semantically significant relation!

8F3.2 *Warning* As mentioned in 8D10(iii) the proof of the shrinking lemma does not necessarily apply to restricted systems of λ -terms, for example the λI -calculus. In fact there is no guarantee that if we shrink a λI -term the result will still be a λI -term, since shrinking may cut out some variables.

Technical details

To avoid interrupting the main lines of thought in the earlier chapters some concepts were defined there only in outline and their main properties were stated without proof. This chapter gives the full definitions and proofs. It should be read only as required to follow the arguments in the other chapters. Its sections are largely independent of each other.

9A The structure of a term

From the viewpoint of logical order this section is best read between 1A3 and 1A4.

As remarked in 1A4, a subterm of a term may have more than one occurrence. The present section introduces a precise notation to distinguish such different occurrences; it is rather clumsy and the reader should avoid using it whenever possible, but in some proofs its precision will be vital. The first step is to define a set of expressions called *positions* that can be assigned to different occurrences of a subterm to show where they occur.

9A1 Definition (Positions) A *position* $p = i_1 \dots i_m$ is any finite (perhaps empty) string of symbols such that i_1, \dots, i_{m-1} are integers and i_m is either an integer or an asterisk, *. Its *length* is m , and if $m = 0$ we say $p = \emptyset$.

- If $m \geq 1$ and $i_m = 1$ we call p a *function* position;
- if $m \geq 1$ and $i_m = 2$ we call p an *argument* position;
- if $m \geq 1$ and $i_m = 0$ we call p a *body* position; and
- if $m \geq 1$ and $i_m = *$ we call p an *abstractor* position.

(Positions containing integers ≥ 3 will be used later but not in the present section.)

The *concatenation* pq of positions $p = i_1 \dots i_m$ and $q = j_1 \dots j_n$ is defined thus: $p\emptyset = p$, $\emptyset q = q$, and if $m, n \geq 1$ and $i_m \neq *$, define

$$pq = i_1 \dots i_m j_1 \dots j_n.$$

(pq is undefined if $m, n \geq 1$ and $i_m = *$.)

A *refinement* of p is any position with form pq ; it is *proper* iff $q \neq \emptyset$.

Two positions $p = i_1 \dots i_m$ and $q = j_1 \dots j_n$ are said to *diverge* iff neither is a refinement of the other, i.e. iff there exists h such that

$$i_h \neq j_h, \quad 1 \leq h \leq \text{Min}\{m, n\}.$$

9A2 Definition (Occurs, occurrence) The phrase “ P occurs in M at position p ” (or “ M contains P at position p ”) is defined by induction on the length of p , thus:

- (i) M occurs in M at position \emptyset ;
- (ii) if P_1P_2 occurs in M at p , then P_i occurs in M at p_i for $i = 1, 2$;
- (iii) if $\lambda x \cdot Q$ occurs in M at p , then Q occurs in M at $p0$ and x occurs in M at $p*$.

An *occurrence of P in M* is a triple $\langle P, p, M \rangle$ such that P occurs in M at p . If $P \equiv x$ and the last symbol of p is $*$ we call $\langle x, p, M \rangle$ a *binding occurrence of x* or an *occurrence of λx* .

An *abstractor* is an occurrence of λx for some x .

A *subterm* of M is any term P that occurs in M .

9A2.1 Note It is easy to prove that at most one term can occur in M at one position. (We assume of course that a term cannot be simultaneously an application and an abstraction, or a composite term and an atom; also that

$$\begin{aligned} P_1P_2 \equiv Q_1Q_2 &\implies P_1 \equiv Q_1 \ \& \ P_2 \equiv Q_2, \\ \lambda x \cdot P \equiv \lambda y \cdot Q &\implies x \equiv y \ \& \ P \equiv Q. \end{aligned}$$

9A2.2 Notation An occurrence $\langle P, p, M \rangle$ may be called simply “ \underline{P} ” when no confusion is likely. A binding occurrence of x may be called either of

$$\underline{x}, \quad \underline{\lambda x}.$$

9A3 Definition (Components) A *component* of a term M is any occurrence in M other than a binding occurrence of a variable, i.e. any occurrence $\langle P, p, M \rangle$ such that the last symbol in p is not $*$.

A component is called *proper* iff it is not $\langle M, \emptyset, M \rangle$.

9A3.1 Note The reason that binding occurrences of variables are denied the name “components” in the above definition is that they play a very different role from other occurrences. In a term such as $\lambda z \cdot xy$ we shall think of \underline{x} and \underline{y} as being the material from which the term is built, but think of $\underline{\lambda z}$ as being part of one of the operations which do the building.

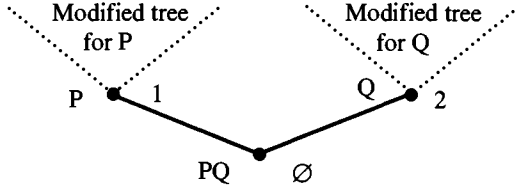
9A4 Definition (The construction-tree of a term) We can display the structure of a term as a tree with each node carrying two labels, a position and a subterm that occurs at that position. This tree is defined for an arbitrary term M as follows.

- (i) If $M \equiv x$, its tree is a single node labelled with x and the empty position, thus:

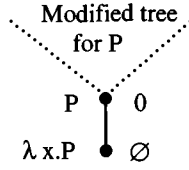
$$x \bullet \emptyset.$$

- (ii) If $M \equiv PQ$, its tree is obtained by first concatenating “1” onto the left end of each position-label in the tree for P , then concatenating “2” onto the left of each

position-label in the tree for Q , and then placing an extra node beneath the two modified trees, as shown below.



(iii) If $M \equiv \lambda x \cdot P$, its tree is obtained by first concatenating “0” onto the left end of each position-label in the tree for P , and then placing an extra node beneath the modified tree, as shown below.



9A4.1 *Example* The construction-tree of $(\lambda x \cdot yx)(\lambda z \cdot x(yx))$ is shown in Fig. 9A4.1a.

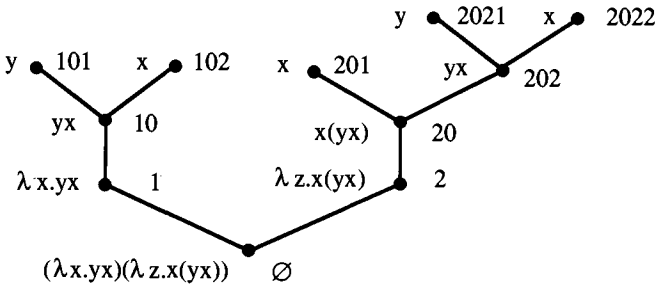


Fig. 9A4.1a. Construction-tree of $(\lambda x \cdot yx)(\lambda z \cdot x(yx))$

9A4.2 *Note* Each component has a node on the construction-tree; but binding occurrences of variables are not components and do not have their own nodes.

9A5 Definition (Relations between components) Let $\underline{P} \equiv \langle P, p, M \rangle$ and $\underline{R} \equiv \langle R, r, M \rangle$ be components of a term M :

(i) iff r is a refinement of p we may say any of

$$\underline{P} \text{ contains } \underline{R}, \quad \underline{R} \text{ is a part of } \underline{P}, \quad \underline{R} \text{ is in } \underline{P};$$

(ii) iff r is a proper refinement of p we may say either of

\underline{P} properly contains \underline{R} , \underline{R} is a proper part of \underline{P} ;

(iii) iff p and r diverge we may say either of

\underline{P} is disjoint from \underline{R} , \underline{P} does not overlap \underline{R} ;

(iv) iff $R \equiv P$ (i.e. $\underline{R}, \underline{P}$ are occurrences of the same term) we may say

\underline{R} is isomorphic to \underline{P} , $\underline{R} \cong \underline{P}$.

9A5.1 Notes (i) If \underline{P} is not disjoint from \underline{R} it is easy to see that either \underline{P} is in \underline{R} or \underline{R} is in \underline{P} . Hence if $\underline{P}_1, \dots, \underline{P}_n$ are components of a term M and no two are disjoint they must be linearly ordered by the relation “is in”; i.e. there must exist a permutation $\langle i_1, \dots, i_n \rangle$ of $\langle 1, \dots, n \rangle$ such that

\underline{P}_{i_1} is in \underline{P}_{i_2} in \underline{P}_{i_3} in ... in \underline{P}_{i_n} .

(ii) If \underline{P} is a proper part of \underline{R} then $P \neq R$.

9A6 Definition (Replacement in a term) Let $\underline{P} \equiv \langle P, p, M \rangle$ be a component of a term M . The result of replacing \underline{P} by a term T will be called

$$\{T/P\}_p M.$$

Its detailed definition proceeds by induction on the length of p , thus:

(i) if $p \equiv \emptyset$: then $P \equiv M$; define

$$\{T/M\}_{\emptyset} M \equiv T;$$

(ii) if $p \equiv q1$: then M contains a term PQ at position q ; define

$$\{T/P\}_{q1} M \equiv \{(TQ)/(PQ)\}_q M;$$

(iii) if $p \equiv q2$: then M contains a term QP at q ; define

$$\{T/P\}_{q2} M \equiv \{(QT)/(QP)\}_q M;$$

(iv) if $p \equiv q0$: then M contains a term $\lambda x \cdot P$ at q ; define

$$\{T/P\}_{q0} M \equiv \{(\lambda x \cdot T)/(\lambda x \cdot P)\}_q M.$$

(The case $p \equiv q*$ is excluded above because then \underline{P} would be a binding occurrence of a variable and not a component; $\{T/P\}_p M$ is undefined in this case.)

9A6.1 Example Let $M \equiv (\lambda x \cdot yx)(\lambda z \cdot x(yx))$, $\underline{P} \equiv \langle yx, 10, M \rangle$, $T \equiv xxyz$. Then

$$\{T/P\}_{10} M \equiv (\lambda x \cdot xxyz)(\lambda z \cdot x(yx)).$$

9A6.2 Note In substitution we change bound variables to avoid clashes but in replacement we do not.

9A6.3 *Note* Of course replacing one component of a term can disturb or destroy other components. To see what happens in two important cases, let $\underline{P} \equiv \langle P, p, M \rangle$ and $\underline{R} \equiv \langle R, r, M \rangle$ and let

$$M^* \equiv \{T/P\}_p M.$$

(i) If \underline{P} is disjoint from \underline{R} , it is not hard to show that M^* contains an occurrence of R at position r (i.e., roughly speaking, replacing \underline{P} does not change its disjoint neighbours.)

(ii) If \underline{P} is in \underline{R} , then $p \equiv rq$ for some position q such that P occurs in R at position q , and it is not hard to show that M^* contains, at position r , the term

$$\{T/P\}_q R.$$

9B Residuals

This section summarises some properties of β -contractions needed in the proof of the weak normalisation theorem in 5C. The full theory of β -reduction is in fact quite deep (see Barendregt 1984 Chapters 3 and 11–14), but none of it is used in this book except the following few very basic ideas.

Everything in this section is valid for both the untyped terms of Chapter 1 and the typed terms of Chapter 5.

9B1 Notation Recall the definitions of β -redex and β -contraction in 1B1. (In this section “ β ” will usually be omitted.) A *redex-occurrence*

$$\underline{R} \equiv \underline{(\lambda x.M)N}$$

is a particular occurrence of a redex in a term. The notations *function-part* and *argument-part* of \underline{R} will be used here for $\underline{\lambda x.M}$ and \underline{N} respectively.

Recall from 1B2 that a *reduction* is a finite or infinite sequence of contractions

$$\langle P_1, \underline{R}_1, Q_1 \rangle, \quad \langle P_2, \underline{R}_2, Q_2 \rangle, \dots$$

where $P_1 \equiv_\alpha P$ and $Q_i \equiv_\alpha P_{i+1}$ for $i = 1, 2, \dots$. This definition allows a reduction to make α -conversions before or after each of its contractions, but the reader may safely ignore these and concentrate on the contractions; the next lemma will say contractions are unaffected by α -conversions in a certain precise sense.

Recall from 1B3 that the *length* of a reduction is the number of its contractions. (And α -conversions are not counted.)

In this section “ ξ ” will denote an arbitrary contraction and “ r ” an arbitrary reduction.

If r_1 is a finite reduction from a term P to a term Q and r_2 is a reduction of Q , the reduction consisting of r_1 followed by r_2 will be called

$$r_1 + r_2.$$

9B1.1 Lemma (α -invariance) If $P \equiv_\alpha P'$ and P contains a β -redex-occurrence $\underline{R} \equiv \langle R, p, P \rangle$ whose contraction changes P to Q , then P' contains a β -redex-occurrence $\langle \underline{R}', p, P' \rangle$ whose contraction changes P' to a term $Q' \equiv_\alpha Q$.

Proof-note Two cases must be considered: (i) P' comes from P by replacing $\lambda x \cdot M$ by $\lambda y \cdot [y/x]M$, (ii) P' comes from P by replacing $\lambda y \cdot [y/x]M$ by $\lambda x \cdot M$ (where $y \notin FV(M)$). The proof is boring but can be slightly simplified by restricting rule (α) as follows (cf. HS 86 Remark 1.21):

(α_0) replace $\lambda x \cdot M$ by $\lambda y \cdot [y/x]M$ if y does not occur at all in M .

(It is straightforward to prove that every re-writing by (α) can be done by a series of re-writings by (α_0) .) The reverse of an (α_0) -replacement is also an (α_0) -replacement, so case (ii) above becomes redundant.

9B1.2 Warning The above lemma does not claim that $R' \equiv_\alpha R$. In fact the following is an example where this fails:

$$P \equiv \lambda x \cdot R, \quad R \equiv (\lambda u \cdot ux)z, \quad P' \equiv \lambda y \cdot [y/x]R.$$

The next definition will describe what happens to a redex-occurrence \underline{S} in a term P when another redex-occurrence \underline{R} in P is contracted.

9B2 Definition (Residuals) Let $\underline{R} \equiv \langle R, p, P \rangle$ and $\underline{S} \equiv \langle S, q, P \rangle$ be redex-occurrences in a term P , say

$$R \equiv (\lambda x \cdot M)N, \quad S \equiv (\lambda y \cdot H)L.$$

Let P change to P' by contracting R . The *residuals of \underline{S} with respect to \underline{R}* are certain redex-occurrences in P' defined as follows (from Curry and Feys 1958 §4B1).

Case 1: \underline{R} does not overlap \underline{S} . Then contracting \underline{R} leaves \underline{S} unchanged; more precisely, by 9A6.3(i), P' contains an occurrence of S at position q ; we call this occurrence the residual of \underline{S} .

Case 2: $\underline{R} \equiv \underline{S}$. Then contracting \underline{R} is the same as contracting \underline{S} ; we say \underline{S} has no residual.

Case 3: $\underline{R} \not\equiv \underline{S}$ and \underline{R} is in \underline{S} . Then \underline{R} must be in \underline{H} or in \underline{L} , and contracting \underline{R} will change $(\lambda y \cdot H)L$ to a component $(\lambda y \cdot H')L$ or $(\lambda y \cdot H)L'$ of P' . We call this the residual of \underline{S} .

Case 4: $\underline{R} \not\equiv \underline{S}$ and \underline{S} is in \underline{R} . Then \underline{S} must be in \underline{M} or in \underline{N} .

Subcase 4a: \underline{S} is in \underline{M} . The contraction of \underline{R} changes $(\lambda x \cdot M)N$ to $[N/x]M$ and changes \underline{S} to a redex-occurrence \underline{S}' with one of the forms

$$[N/x]\underline{S}, \quad [N/x][z_1/y_1] \dots [z_n/y_n]\underline{S}, \quad \underline{S}.$$

(The second form will apply if some changes of bound variables are needed during the substitution $[N/x]M$; the third if \underline{S} is in the scope of an occurrence of λx in \underline{M} .) We call this \underline{S}' the residual of \underline{S} .

Subcase 4b: \underline{S} is in \underline{N} . Let \underline{M} contain $k \geq 0$ free occurrences of x . The contraction of \underline{R} changes $(\lambda x \cdot M)N$ to $[N/x]M$ which contains k substituted occurrences of N , each containing an occurrence of S . We call these occurrences of S the residuals of \underline{S} .

9B2.1 Notation The set of all the residuals of \underline{S} with respect to \underline{R} will be called “ $\underline{S}/\underline{R}$ ” (or “ \underline{S}/ξ ” if ξ is the contraction $\langle P, \underline{R}, P' \rangle$).

A redex-occurrence \underline{S} will be called the *parent* of its residuals.

9B2.2 Note (i) If \underline{S} is not a part of \underline{R} then $\underline{S}/\underline{R}$ has exactly one member (because then Case 2 and Subcase 4b cannot happen).

(ii) Each member of $\underline{S}/\underline{R}$ is an occurrence of a term $(\lambda z \cdot H')L'$, where $z \equiv y$ except possibly in Case 4a if changes of bound variables are needed, and H' and L' are either H and L or terms obtained from them by substitutions.

(iii) The definition of residuals is meaningful for the typed-term system in Chapter 5 as well as for untyped terms. If P is a typed term then by (ii) each residual $(\lambda z \cdot H')L'$ of \underline{S} in P' has the same type as \underline{S} . Further, its function-part $\lambda z \cdot H'$ has the same type as the function-part $\lambda y \cdot H$ of \underline{S} . This fact is needed in the proof of the weak normalisation theorem in 5C.

9B3 Definition Let r be a reduction from P to Q and let \underline{S} be a redex-occurrence in P . The set \underline{S}/r of the *residuals of \underline{S} with respect to r* is a set of redex-occurrences in Q defined thus:

(i) If r has no contractions and no α -conversions (so $P \equiv Q$), define

$$\underline{S}/r = \{\underline{S}\}.$$

(ii) If $r = r^\circ + \xi$ where r° reduces P to a term Q° and ξ is a contraction reducing Q° to Q , and $\underline{S}/r^\circ = \{\underline{S}_1, \dots, \underline{S}_n\}$, define

$$\underline{S}/(r^\circ + \xi) = \underline{S}_1/\xi \cup \underline{S}_2/\xi \cup \dots \cup \underline{S}_n/\xi.$$

(iii) If r ends with an α -conversion $Q' \equiv_\alpha Q$ and the residuals of \underline{S} in Q' are $\underline{S}_1, \dots, \underline{S}_n$, define the residuals of \underline{S} in Q to be the redex-occurrences in Q with the same positions as $\underline{S}_1, \dots, \underline{S}_n$ in Q' . (These exist by 9B1.1.)

Finally, if $\mathbb{S} = \{\underline{S}_1, \dots, \underline{S}_m\}$ is a set of redex-occurrences in P , define

$$\mathbb{S}/r = \underline{S}_1/r \cup \dots \cup \underline{S}_m/r.$$

9B4 Definition (M.C.D.'s) Let $\mathbb{S} = \{\underline{S}_1, \dots, \underline{S}_m\}$ be a set of redex-occurrences in a term P . A *development* of \mathbb{S} is a finite or infinite reduction $\xi_1 + \xi_2 + \xi_3 + \dots$ (with α -conversions allowed), such that $\xi_1 \in \mathbb{S}$ and for $n \geq 1$,

$$\xi_{n+1} \in \mathbb{S}/(\xi_1 + \dots + \xi_n).$$

A *complete development* of \mathbb{S} is a finite development $\xi_1 + \dots + \xi_n$ such that

$$\mathbb{S}/(\xi_1 + \dots + \xi_n) = \emptyset.$$

A *minimal* member of \mathbb{S} is one that does not contain another member. A *minimal complete development (m.c.d.)* of \mathbb{S} is a complete development $\xi_1 + \dots + \xi_n$ such that ξ_1 is minimal in \mathbb{S} and ξ_{i+1} is minimal in $\mathbb{S}/(\xi_1 + \dots + \xi_i)$ for $i = 1, \dots, n-1$ (and the only α -conversions are at the end of ξ_n).

If P changes to Q by an m.c.d. of some set of redex-occurrences in P we shall say

$$P \triangleright_{\text{mcd}} Q.$$

9B4.1 Lemma Every set \mathbb{S} of redex-occurrences in a term has an m.c.d..

Proof Let $\mathbb{S} = \{\underline{S}_1, \dots, \underline{S}_m\}$. Since \mathbb{S} is finite it has at least one minimal member; choose one, say \underline{S}_j , and contract it; by 9B2.2(i) each \underline{S}_i ($i \neq j$) will have just one residual, so $\mathbb{S}/\underline{S}_j$ will have only $n-1$ members. By continuing this process we must eventually reach a term containing no residuals of \mathbb{S} . \square

9B4.2 Notes (i) If \mathbb{S} is empty an m.c.d. of \mathbb{S} has no contractions.

(ii) A single contraction is an m.c.d. of a one-member set.

(iii) The main properties of developments are described in Barendregt 1984 §11.3. In particular it is shown there that infinite developments are impossible even in the untyped λ -calculus. Also the concept of m.c.d. is used in HS86 Appendix 1 as the basis of a proof of the Church-Rosser theorem.

9B5 Definition (Newly created redex-occurrences) If P changes to P' by contracting a redex-occurrence \underline{R} , and P' contains a redex-occurrence S that is not a residual of a redex-occurrence in P , we say \underline{S} has been *newly created* by the contraction of \underline{R} .

9B5.1 Discussion The proof of the weak normalisation theorem in 5C depends on knowing how a newly created redex-occurrence can be produced, so the possible ways of doing this will be analysed here. Let \underline{R} be a redex-occurrence at a position p in a term P , say

$$R \equiv ((\lambda x \cdot M)N).$$

Let P change to P' by contracting \underline{R} ; then at position p in P' there is an occurrence of

$$[N/x]M.$$

Let \underline{S} be any newly created redex-occurrence in P' ; say

$$S \equiv ((\lambda z \cdot H)L).$$

Now \underline{S} must overlap $[N/x]M$, otherwise it would be the residual of a redex-occurrence in P that did not overlap \underline{R} . Hence either \underline{S} properly contains $[N/x]M$ or \underline{S} is in $[N/x]M$.

Case 1: \underline{S} properly contains $[N/x]M$. If $[N/x]M$ were in \underline{H} or \underline{L} then \underline{S} would not be new; hence $[N/x]M \equiv \lambda z \cdot \underline{H}$. Thus $\underline{S} \equiv ([N/x]M)L$ and P must contain \underline{RL} . Now the only way to obtain $[N/x]M \equiv \lambda z \cdot \underline{H}$ is either to have $M \equiv x$ and $N \equiv \lambda z \cdot H$ or to have $M \equiv \lambda y \cdot H'$ for some y and H' .

Subcase 1a: $M \equiv x$, $N \equiv \lambda z \cdot H$. Then $R \equiv (\lambda x \cdot x)N$ and the contraction of \underline{R} proceeds thus:

$$RL \equiv (\lambda x \cdot x)NL \triangleright_{\beta} NL \equiv (\lambda z \cdot H)L \equiv S.$$

Subcase 1b: $M \equiv \lambda y \cdot H'$. Then $R \equiv (\lambda x \cdot (\lambda y \cdot H'))N$ and the contraction of \underline{R} proceeds thus:

$$RL \equiv (\lambda x \cdot (\lambda y \cdot H'))NL \triangleright_{\beta} ([N/x](\lambda y \cdot H'))L \equiv S.$$

Case 2: \underline{S} is in $[N/x]M$. Let x_1, \dots, x_k be the free occurrences of x in \underline{M} in P , and let $\underline{N}_1, \dots, \underline{N}_k$ be the corresponding occurrences of N in $[N/x]M$ in P' . If \underline{S} were in an \underline{N}_i or were disjoint from all of $\underline{N}_1, \dots, \underline{N}_k$, it is easy to see that \underline{S} would not be new. Hence at least one \underline{N}_i must be in \underline{S} . If each such \underline{N}_i was in \underline{H} or \underline{L} , \underline{S} would be a residual of a redex-occurrence in \underline{M} , so one \underline{N}_i must coincide with $\underline{\lambda z} \cdot \underline{H}$. This can only happen if \underline{M} contains a component with form xL' for some L' , and

$$N \equiv \lambda z \cdot H, \quad \underline{S} \equiv [N/x](\underline{xL'}), \quad L \equiv [N/x]L'.$$

Conclusion: in each of the above cases the function-part of the newly created redex \underline{S} is an occurrence of N or of $[N/x]M$.

9C The structure of a TA_λ -deduction

This section should be read in parallel with the outline definition of “ TA_λ -deduction” in 2A8. That outline is filled in here and some lemmas are proved for use in Chapters 5 and 8.

A TA_λ -deduction as defined below will be a slightly more elaborate object than the trees shown in the examples in Chapter 2, and each deduction-tree in that chapter should be regarded as an abbreviation for a deduction as defined below.

9C1 Definition (TA_λ -deductions) TA_λ -deductions are labelled trees defined as follows, each node carrying two labels, a TA_λ -formula and a position.

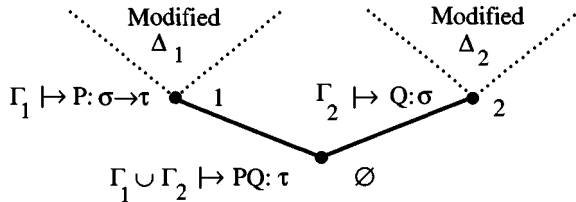
(i) An **atomic** deduction is a single node labelled with an axiom and the empty position, thus:

$$x:\tau \mapsto x:\tau \bullet \emptyset.$$

(ii) If Δ_1 and Δ_2 are deductions whose bottom nodes are labelled by the position \emptyset and the formulae, respectively,

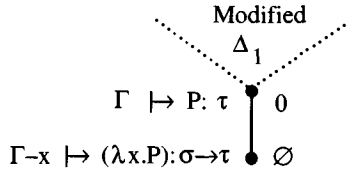
$$\Gamma_1 \mapsto P:(\sigma \rightarrow \tau), \quad \Gamma_2 \mapsto Q:\sigma,$$

and $\Gamma_1 \cup \Gamma_2$ is consistent, then a new deduction called $(\Delta_1 \Delta_2)$ is constructed by first putting “1” on the left end of each position-label in Δ_1 , then putting “2” on the left end of each position-label in Δ_2 , and then placing an extra node beneath the two modified deductions, as shown below.



(iii) If Δ_1 is a deduction whose bottom node is labelled by the position \emptyset and the formula $\Gamma \mapsto P:\tau$, and Γ is consistent with $x:\sigma$, then a new deduction called $(\lambda x \cdot \Delta_1)$

is constructed by first putting “0” on the left end of each position-label in Δ_1 and then placing an extra node beneath the modified deduction, as shown below.



9C1.1 Example For examples of deductions see 2A8.2–5 and the answers to Exercises 2A8.7–8. (The deductions in those examples omit all position-labels and are displayed in the standard space-saving way using horizontal lines instead of the node-and-line way shown in the diagrams above, but this is merely a matter of representation and is not intended to conflict with the definition.)

9C1.2 Note If we remove contexts and types from all the TA_λ -formulae in a deduction-tree whose conclusion is $\Gamma \mapsto M:\tau$, it will be transformed into the construction-tree of M as defined in 9A4.

9C2 Definition The *length*, $|\Delta|$, of a TA_λ -deduction Δ is defined thus: $|\Delta| = 1$ for atomic Δ , and for composite Δ

$$|\Delta_1\Delta_2| = |\Delta_1| + |\Delta_2|, \quad |\lambda x.\Delta_1| = 1 + |\Delta_1|.$$

Clearly $|\Delta| = |M|$ if the conclusion of Δ is $\Gamma \mapsto M:\tau$. (For $|M|$ see 1A2.)

9C3 Definition The set of all type-variables occurring in a deduction Δ will be called

$$Vars(\Delta).^1$$

9C4 Discussion (Replacement in a deduction) Let M contain a component $\underline{P} \equiv \langle P, p, M \rangle$ and let Δ be a TA_λ -deduction of

$$\Gamma \mapsto M:\tau.$$

Then from 9C1 it is easy to see that Δ contains a deduction $\Delta_{\underline{P}}$ of a formula

$$(1) \quad \Gamma_{\underline{P}} \mapsto P:\sigma_{\underline{P}}$$

for some $\Gamma_{\underline{P}}$ and $\sigma_{\underline{P}}$.

It is natural to expect that replacing P by a new term T with the same type will leave the type of M unchanged. But the types of P , T and M all depend on contexts: Γ for M , $\Gamma_{\underline{P}}$ for P , and an unspecified one for T . So our expectation is imprecise. It will be made precise and proved in the two replacement lemmas below.

¹ Warning: deductions Δ contain term-variables as well as type-variables but only the latter are included in $Vars(\Delta)$.

The first step towards precision is to clarify the relationship between $\Gamma_{\underline{P}}$ and Γ . If \underline{P} is not in the body of a λ -abstract in M then clearly

$$(2) \quad \Gamma_{\underline{P}} \subseteq \Gamma.$$

But now suppose \underline{P} is in the bodies of some λ -abstracts in M ; say there are n distinct such abstracts:

$$\underline{\lambda x_1 N_1}, \dots, \underline{\lambda x_n N_n},$$

with \underline{P} in N_i for $i = 1, \dots, n$. By 9A5.1 these components must be linearly ordered by the relation “is in”; by renaming them if necessary, we can assume that

$$\underline{P} \text{ is in } \underline{N_1} \text{ in } \underline{\lambda x_1 N_1} \text{ in } \underline{N_2} \text{ in } \underline{\lambda x_2 N_2} \dots \text{ in } \underline{N_n} \text{ in } \underline{\lambda x_n N_n}.$$

By the subject-construction theorem (2B2) Δ must contain n applications of (\rightarrow I) below (1), and each of these has one of the following forms, depending on whether $x_i \in FV(N_i)$ or not:

$$(3) \quad \frac{\Gamma_i, x_i : \xi_i \mapsto N_i : \eta_i}{\Gamma_i \mapsto (\lambda x_i. N_i) : \xi_i \rightarrow \eta_i}, \quad \frac{\Gamma_i \mapsto N_i : \eta_i}{\Gamma_i \mapsto (\lambda x_i. N_i) : \xi_i \rightarrow \eta_i}.$$

(Here $1 \leq i \leq n$.) Hence

$$(4) \quad \Gamma_{\underline{P}} \subseteq \Gamma \cup \{x_1 : \xi_1, \dots, x_n : \xi_n\}.$$

If M has no bound-variable clashes then x_1, \dots, x_n are distinct and the right-hand side of (4) is consistent. (But if M has bound-variable clashes an $x_i : \xi_i$ might be inconsistent with an $x_j : \xi_j$ ($j \neq i$) or even with a member of Γ .)

The replacement lemmas will now be stated. The first one is easy and is used in proving the subject-reduction theorem (2C1). The second is more complex to state but gives a stronger result; it is applied in 5B2.1(ii) which is used in 8F2.

9C5 First Replacement Lemma for Deductions *Let Δ be a TA_λ -deduction of the formula $\Gamma \mapsto M : \tau$ and let M contain a component \underline{P} with a position p . Then Δ contains a deduction $\Delta_{\underline{P}}$ of a formula with form*

$$\Gamma_{\underline{P}} \mapsto P : \sigma_{\underline{P}}$$

for some $\Gamma_{\underline{P}}$ and $\sigma_{\underline{P}}$; let $\{T/P\}_p M$ be the result of replacing P at p by a term T such that

$$\Gamma_T \vdash_\lambda T : \sigma_{\underline{P}}$$

for some $\Gamma_T \subseteq \Gamma_{\underline{P}}$. Then

$$\Gamma \vdash_\lambda \{T/P\}_p M : \tau.$$

Proof Let $\Gamma_T^* \equiv \Gamma_T \upharpoonright T$. Then $\text{Subjects}(\Gamma_T^*) \doteq FV(T)$ and by 2A11(i) there is a deduction Δ_T of

$$\Gamma_T^* \mapsto T : \sigma_{\underline{P}}.$$

Replace $\Delta_{\underline{P}}$ by Δ_T in Δ and modify the contexts and subjects in all the formulae

below Δ_P accordingly. Since $\Gamma_T \subseteq \Gamma_P$ none of these modifications enlarges a context, so no inconsistencies are introduced. Therefore

$$\Gamma \vdash_{\lambda} \{T/P\}_p M : \tau. \quad \square$$

9C6 Second Replacement Lemma for Deductions *Let Δ, M, P be as in the first replacement lemma, and let $\{T/P\}_p M$ be the result of replacing P at p by a term T such that*

$$(i) \quad \Gamma_T \vdash_{\lambda} T : \sigma_P$$

for some Γ_T such that

$$(ii) \quad \Gamma_T \cup \Gamma \cup \{x_1 : \xi_1, \dots, x_n : \xi_n\} \text{ is consistent,}$$

where x_1, \dots, x_n ($n \geq 0$) are the binding variables of the λ -abstracts in M whose bodies contain P , and ξ_1, \dots, ξ_n are their types in Δ (see 9C4 for details). Then

$$(iii) \quad \Gamma \cup \Gamma_T \vdash_{\lambda} \{T/P\}_p M : \tau.$$

Proof Just as in the previous lemma, by 2A11(i) we can assume $\text{Subjects}(\Gamma_T) = FV(T)$. Replace Δ_P by Δ_T in Δ and modify the contexts and subjects in all the formulae below Δ_P accordingly. No inconsistencies will be introduced, because by 9C4(4) the only variables not in Γ that could cause problems are x_1, \dots, x_n . \square

9C6.1 Corollary *The conclusion of Lemma 9C6 also holds if instead of 9C6(ii) we assume that $\Gamma \cup \Gamma_T$ is consistent and none of x_1, \dots, x_n occurs free in T .*

Proof If $\text{Subjects}(\Gamma_T) = FV(T)$ the above assumption implies that Γ_T satisfies 9C6(ii). \square

9D The structure of a type

This section and the next give two alternative approaches to the structure of an arbitrary type. The approach in the next section is used in Chapter 8; that in this one is more basic and is included mainly as a contrast and introduction to the other approach.

From the viewpoint of logical order the present section fits between 2A2 and 2A3.

9D1 Definition (Occurs, occurrence) Here *positions* are the same as in 9A1; however, positions containing $*$ are not needed in the present section. The phrase “ σ occurs in τ at position p ” is defined thus (cf. 9A2):

- (i) τ occurs in τ at position \emptyset ;
- (ii) if $\sigma_1 \rightarrow \sigma_2$ occurs in τ at p , then σ_i occurs in τ at pi ($i = 1, 2$).

A triple $\langle \sigma, p, \tau \rangle$ such that σ occurs in τ at p is called an *occurrence* of σ in τ , or a *component* of τ . A type that occurs in τ is called a *subtype* of τ .

9D1.1 Notation An occurrence $\langle \sigma, p, \tau \rangle$ may be called simply “ σ ” when no confusion is likely.

Recall from 2A2 that the number of occurrences of variables in τ is called $|\tau|$ and

the set of these variables is called $Vars(\tau)$. The set of all variables occurring in a finite sequence $\langle \tau_1, \dots, \tau_n \rangle$ will be called

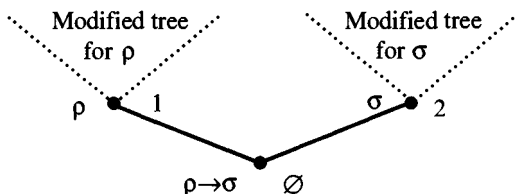
$$Vars(\tau_1, \dots, \tau_n).$$

9D2 Definition (Construction-tree of a type) The *construction-tree* of a type τ is a labelled tree showing how τ is built up (cf. the tree of a term in 9A4). Each node has two labels, a type and a position, and the tree is defined thus:

- (i) If τ is an atom, say $\tau \equiv e$, its tree has just one node:

$$e \bullet \emptyset.$$

- (ii) If $\tau \equiv \rho \rightarrow \sigma$, its tree is built from the trees for ρ and σ by first putting “1” on the left end of each position-label in the tree for ρ , next putting “2” on the left end of each position-label in the tree for σ , and then placing an extra node beneath the two modified trees, thus:



9D2.1 Example Let $\tau \equiv (a \rightarrow (b \rightarrow c)) \rightarrow ((a \rightarrow b) \rightarrow (a \rightarrow c))$; its tree is as shown in Fig. 9D2.1a.

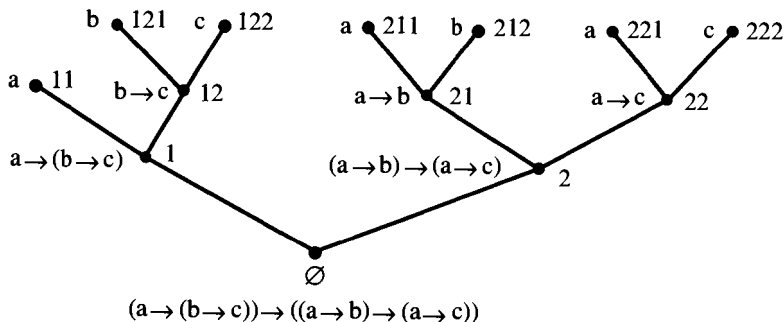


Fig. 9D2.1a.

9D3 Definition (Positive and negative occurrences) Let $\underline{\sigma} \equiv \langle \sigma, p, \tau \rangle$ be an occurrence of σ . We call $\underline{\sigma}$ *positive* (and say that σ *occurs positively* in τ) iff $p = \emptyset$ or the number of 1's in p is even. We call $\underline{\sigma}$ *negative* and say that σ *occurs negatively* in τ iff the number of 1's in p is odd.

9D3.1 *Example* The positive occurrences in the type τ shown in Example 9D2.1 are

$$\begin{aligned} \langle \tau, \emptyset, \tau \rangle, \quad \langle a, 11, \tau \rangle, \quad \langle b, 121, \tau \rangle, \quad \langle (a \rightarrow b) \rightarrow (a \rightarrow c), 2, \tau \rangle, \\ \langle a, 211, \tau \rangle, \quad \langle a \rightarrow c, 22, \tau \rangle, \quad \langle c, 222, \tau \rangle. \end{aligned}$$

All other occurrences in this type τ are negative.

The reader familiar with the usual concept of positive and negative occurrences in propositional logic can easily check that the above definition agrees with it.

9E The condensed structure of a type

The present section analyses the structure of an arbitrary type in a more compact way than the last, and computes a bound on the number of its components of a certain kind. This bound is used in the proof of 8F3.

The starting point is a remark in 8B1 that every type τ can be expressed uniquely in the form

$$(1) \quad \tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow e$$

where $m \geq 0$ and e is an atom. In this section we shall view a type as having been built up from its atoms by the operation of constructing (1) from τ_1, \dots, τ_m and e instead of the more usual operation of constructing $(\rho \rightarrow \sigma)$ from ρ and σ . Corresponding to this new view a new construction-tree of a type can be defined that is more condensed than the usual one and in which there is no bound on the number of branches that may start from a node. The following definitions formalize this view.

9E1 Notation Just as in 9A1, a *position* is any finite (perhaps empty) string $p = i_1 \dots i_m$ of integers and *'s such that $i_1, \dots, i_{m-1} \neq *$. The present section will use the notation of 9A1, with the following exceptions:

if $m \geq 1$ and $i_m \neq *$, p will be called a *premise* position;

if $m \geq 1$ and $i_m = *$, p will be called a *conclusion* or *tail* position.

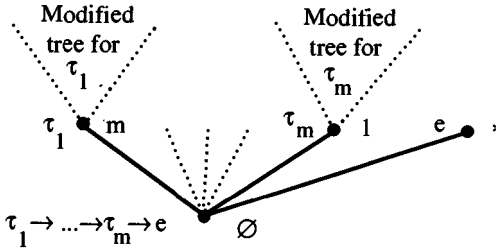
9E2 Definition (Condensed tree of a type) The *condensed construction-tree* of a type τ is defined by induction on $|\tau|$, thus. (Each of its nodes is labelled with a type and a position.)

(i) If τ is an atom e , its condensed tree is a single node:

$$e \bullet \emptyset.$$

(ii) If $\tau \equiv \tau_1 \rightarrow \dots \tau_m \rightarrow e$ ($n \geq 1$), construct its condensed tree from the condensed trees of τ_1, \dots, τ_m by first replacing each position-label p in the tree of τ_i by

$(m + 1 - i)p$ (for $i = 1, \dots, m$), and then combining the modified trees as follows:



9E2.1 Note The use of $(m + 1 - i)p$ in (ii) above has the effect of assigning positions to τ_1, \dots, τ_m backwards, giving position 1 to τ_m , 2 to τ_{m-1}, \dots , and m to τ_1 . This makes it easier to relate positions in the condensed tree of $\sigma \rightarrow \tau$ to those of σ and τ , though we shall not need this facility in this book.

As an example, Fig. 9E2.1a shows the condensed tree for the type

$$\begin{aligned}\tau &\equiv (a \rightarrow (b \rightarrow c)) \rightarrow ((a \rightarrow b) \rightarrow (a \rightarrow c)) \\ &\equiv (a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c.\end{aligned}$$

9E3 Definition (S-subtypes) The *significant subtypes* or *s-subtypes* of a type τ are the types that label the nodes in the condensed tree of τ . A *proper* s-subtype of τ is an s-subtype $\neq \tau$.

9E3.1 Example The s-subtypes of the type τ in Fig. 9E2.1a are

$$a, \quad b, \quad c, \quad a \rightarrow b \rightarrow c, \quad a \rightarrow b, \quad \tau.$$

Note that although $b \rightarrow c$ is a subtype of τ in the usually accepted sense (9D1) it is not an s-subtype because it does not correspond to a node on the condensed tree.

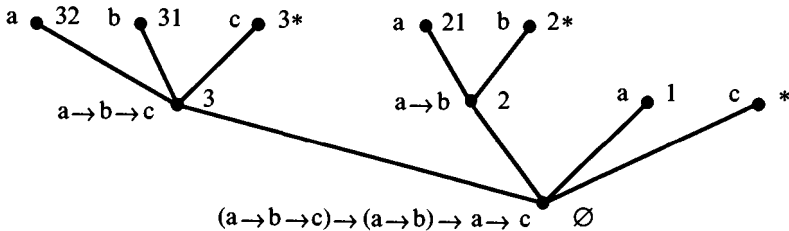


Fig. 9E2.1a.

9E4 Definition (S-components) If a node on the condensed tree of τ is labelled with a type σ and a position p we call the triple $\langle \sigma, p, \tau \rangle$ an *s-component* of τ . (Thus an s-component is a particular occurrence of an s-subtype.)

9E4.1 Notation S-components are distinguished from s-subtypes by underlining their names. The phrases “ $\underline{\rho}$ contains $\underline{\sigma}$ ”, “ $\underline{\rho}$ properly contains $\underline{\sigma}$ ” etc. are defined here as in 9A5.

9E5 Definition (Premises, tail) If $\underline{\rho}$ is a composite s-component of a type τ and $\rho \equiv \rho_1 \rightarrow \dots \rightarrow \rho_m \rightarrow a$ ($m \geq 1$), the s-components $\underline{\rho}_1, \dots, \underline{\rho}_m$ are called the *premises* of $\underline{\rho}$ and \underline{a} is called the *conclusion* or *tail-component* of $\underline{\rho}$, or just

$$\text{Tail}(\underline{\rho}).$$

9E5.1 Lemma Two distinct s-components of a type τ cannot have the same tail-component.

Proof Induction on $|\tau|$, using the fact that if $\tau \equiv \tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow e$ the only s-components containing \underline{e} are τ and \underline{e} , and \underline{e} is not the tail of itself because atoms do not have tails (by 9E5). \square

9E5.2 Warning The above lemma does not say that the tails of two distinct s-components $\underline{\rho}$ and $\underline{\sigma}$ cannot be occurrences of the same atom. That is, using the \cong -notation of (iv), the lemma forbids $\text{Tail}(\underline{\rho}) \equiv \text{Tail}(\underline{\sigma})$ but does not forbid $\text{Tail}(\underline{\rho}) \cong \text{Tail}(\underline{\sigma})$.

9E6 Definition (Subpremises, subtails) An s-component of τ is called a *subpremise* or *subtail* of τ according as it is a premise or tail of another s-component of τ . The sets of all subpremises and all subtails of τ will be called, respectively,

$$\text{Subpremises}(\tau), \quad \text{Subtails}(\tau).$$

9E6.1 Example The type $\tau \equiv (a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$ in Fig. 9E2.1a has six subpremises, namely all three \underline{a} 's and

$$\langle a \rightarrow b \rightarrow c, 3, \tau \rangle, \quad \langle a \rightarrow b, 2, \tau \rangle, \quad \langle b, 31, \tau \rangle.$$

It has three subtails, namely

$$\langle b, 2^*, \tau \rangle, \quad \langle c, 3^*, \tau \rangle, \quad \langle c, *, \tau \rangle.$$

9E6.2 Notes (i) A proper s-component $\underline{\sigma} = \langle \sigma, p, \tau \rangle$ ($p \neq \emptyset$) is a subpremise iff p is a premise position and a subtail iff p is a tail position.

(ii) Each s-component of τ is either a subtail, an atomic subpremise, a composite subpremise, or τ itself, and cannot be more than one of these.

(iii) If τ is composite, its leftmost atom-occurrence is a subpremise and its rightmost is a subtail.

(iv) An atom has no subpremises or subtails.

9E6.3 Lemma *If τ is composite, then*

- (i) $\#(\text{Subtails}(\tau))$ = no. of composite s-components of τ ,
- (ii) = 1 + no. of composite subpremises of τ ,
- (iii) = $|\tau|$ - no. of atomic subpremises of τ ,
- (iv) $\leq |\tau| - 1$,
- (v) $\#(\text{Subpremises}(\tau))$ = $|\tau| - 1$,
- (vi) no. of s-components of $\tau \leq 2|\tau| - 1$.

Proof For (i): use 9E5.1. For (ii): each composite s-component is either a subpremise or τ itself. For (iii): use 9E6.2(ii). For (iv): use 9E6.2(iii). For (v): subtract (iii) from (ii). For (vi): use 9E6.2(ii), adding (iv) to (v) and adding 1 for τ itself. \square

9E7 Definition *Order*(τ), the *order* of τ , is 1 + the length of the longest position on the condensed tree of τ . In detail: *Order*(e) = 1 for atoms e , and for composite types

$$\text{Order}(\tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow e) = 1 + \text{Max}\{\text{Order}(\tau_1), \dots, \text{Order}(\tau_m)\}.$$

9E7.1 Example *Order*(($a \rightarrow b \rightarrow c$) \rightarrow ($a \rightarrow b$) $\rightarrow a \rightarrow c$) = 3.

9E8 Definition (Positive and negative s-components) An s-component $\underline{\sigma}$ of τ is called *positive* or *negative* according as the number of non-asterisk symbols in its position is even or odd. If $\underline{\sigma}$ is positive we say σ *occurs positively* in τ , otherwise σ *occurs negatively* in τ .

9E8.1 Example If $\tau \equiv (a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$, see Fig. 9E2.1a, its positive s-components are

$$\langle \tau, \emptyset, \tau \rangle, \quad \langle c, *, \tau \rangle, \quad \langle a, 32, \tau \rangle, \quad \langle b, 31, \tau \rangle, \quad \langle a, 21, \tau \rangle.$$

9E8.2 Notes (i) It is straightforward to show that an s-component $\underline{\sigma}$ is positive or negative according as the corresponding component in the more usual sense (9D3) is positive or negative.

(ii) A subpremise of τ is positive iff its position has even length. (Because the position of a subpremise contains no *'s.)

The following set plays a role in Chapter 8.

9E9 Definition (*NSS*(τ)) (cf. Ben-Yelles 1979 Def. 3.36.) If τ is composite, *NSS*(τ) is the set of all finite sequences $\langle \sigma_1, \dots, \sigma_n \rangle$ ($n \geq 1$) such that τ contains a positive composite s-component with form

$$\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow a$$

for some atom a . Each member of *NSS*(τ) is called a *negative subpremise-sequence* (because it is a sequence of terms that have occurrences as negative subpremises in τ).

The set of all the members of the sequences in *NSS*(τ) will be called

$$\bigcup \text{NSS}(\tau).$$

9E9.1 *Example* If $\tau \equiv (a \rightarrow (b \rightarrow d \rightarrow c) \rightarrow d) \rightarrow (a \rightarrow b \rightarrow c) \rightarrow d \rightarrow d$, we have

$$\begin{aligned} NSS(\tau) &= \{ \langle a \rightarrow (b \rightarrow d \rightarrow c) \rightarrow d, a \rightarrow b \rightarrow c, d \rangle, \langle b, d \rangle \}, \\ \bigcup NSS(\tau) &= \{ a \rightarrow (b \rightarrow d \rightarrow c) \rightarrow d, a \rightarrow b \rightarrow c, d, b \}. \end{aligned}$$

9E9.2 *Notes* (i) If τ is composite, say $\tau \equiv \tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow e$ ($m \geq 1$), and each τ_i has form

$$\tau_i \equiv \tau_{i,1} \rightarrow \dots \rightarrow \tau_{i,m_i} \rightarrow e_i \quad (m_i \geq 0),$$

then

$$NSS(\tau) = \{ \langle \tau_1, \dots, \tau_m \rangle \} \cup \bigcup \{ NSS(\tau_{i,j}) : 1 \leq j \leq m_i, 1 \leq i \leq m \}.$$

(ii) If τ is an atom, $NSS(\tau) = \emptyset$.

(iii) If ρ occurs positively in τ then $NSS(\rho) \subseteq NSS(\tau)$.

9E9.3 *Lemma* If τ is composite, then

- (i) $\#(NSS(\tau)) \leq \text{no. of positive composite s-components of } \tau$,
- (ii) $\leq |\tau| - 1$,
- (iii) $\#(\bigcup NSS(\tau)) \leq |\tau| - 1$,
- (iv) $\langle \sigma_1, \dots, \sigma_n \rangle \in NSS(\tau) \implies n \leq |\tau| - 1$.

Proof For (i): each $\langle \sigma_1, \dots, \sigma_n \rangle \in NSS(\tau)$ is obtained from a distinct positive composite s-component of τ . For (ii): use 9E6.3 (i) and (iv). For (iii): note that

$$\bigcup NSS(\tau) \subseteq \text{Subpremises}(\tau)$$

and use 9E6.3 (v). For (iv): use the definition of $NSS(\tau)$. □

9F Imitating combinatory logic in λ -calculus

This section is used in 6D and 7D. The converse PT theorems in 7A can be seen as completeness theorems for the rule (D) described in 7D1; but that rule is formulated for Hilbert-style axiom-based versions of logic not Natural Deduction versions, and in the correspondence between logic and types the Hilbert style corresponds more closely to combinatory logic than to λ -calculus. It is possible to describe this correspondence in a λ -context but to do so we must first use λ -terms to imitate a little of standard combinatory logic technique; this is done in the present section.

Recall the following closed λ -terms:

$$\begin{aligned} \mathbf{B} &\equiv \lambda xyz \cdot x(yz), & \mathbf{B}' &\equiv \lambda xyz \cdot y(xz), & \mathbf{C} &\equiv \lambda xyz \cdot xzy, & \mathbf{I} &\equiv \lambda x \cdot x, \\ \mathbf{K} &\equiv \lambda xy \cdot x, & \mathbf{S} &\equiv \lambda xyz \cdot xz(yz), & \mathbf{W} &\equiv \lambda xy \cdot xyy. \end{aligned}$$

9F1 Definition (\mathbb{S} -combinations) If \mathbb{S} is a set of λ -terms, an \mathbb{S} -*combination*, or *applicative combination of members of \mathbb{S}* , is a λ -term built from some or all of the members of \mathbb{S} by application only. An \mathbb{S} -*and-variables combination* is an applicative combination of members of \mathbb{S} and variables.

For subsets of $\{\mathbf{B}, \mathbf{B}', \mathbf{C}, \mathbf{I}, \mathbf{K}, \mathbf{S}, \mathbf{W}\}$ the \mathbb{S} -combinations will be called **BCK-combinations**, **BCIW-combinations**, etc.

9F1.1 Example If $\mathbb{S} = \{\mathbf{B}, \mathbf{C}, \mathbf{K}\}$ then \mathbf{CKK} and \mathbf{B} are \mathbb{S} -combinations and $\mathbf{CK}x, xy$ and \mathbf{CKK} are \mathbb{S} -and-variables combinations. But $\lambda x. \mathbf{BC}$ is not an \mathbb{S} -combination or an \mathbb{S} -and-variables combination.

The theory called combinatory logic was originally developed from the idea that λ -abstraction can be imitated by building combinations of a very limited set of operators called *basic combinators*. (See HS 86 Ch. 2.) In many accounts of the theory just two basic combinators called \mathbf{S} and \mathbf{K} are assumed, with similar reduction-properties to the λ -terms \mathbf{S} and \mathbf{K} above. But the version below uses $\mathbf{B}, \mathbf{C}, \mathbf{K}$ and \mathbf{W} instead; this makes it easier to discuss particular subsystems.

9F2 Definition For λ -terms P and Q , we shall say that P *β -reduces to Q with strong type-invariance* iff

- (i) $P \triangleright_{\beta} Q$,
- (ii) $FV(P) = FV(Q)$,
- (iii) for all Γ and τ , $\Gamma \vdash_{\lambda} P : \tau \iff \Gamma \vdash_{\lambda} Q : \tau$.

We shall say P *β -converts to Q with strong type-invariance* iff $P =_{\beta} Q$ and (ii) and (iii) hold.

9F3 Combinatory Completeness Theorem *There is an algorithm which accepts any λ -term M and constructs a \mathbf{BCKW} -and-variables combination M^* which β -reduces to M with strong type-invariance.*

Proof The algorithm will construct M^* by induction on $|M|$. If M is a variable choose $M^* \equiv M$.

If $M \equiv NP$ and N^* and P^* have been constructed, choose $M^* \equiv N^*P^*$. (To see that N^*P^* satisfies 9F2 (iii), use the induction hypothesis and the subject-construction theorem, 2B2(iii).)

Now assume $M \equiv \lambda x.N$ and N^* has been constructed. By the induction hypothesis,

$$(1) \quad FV(\lambda x.N^*) = FV(M), \quad \lambda x.N^* \triangleright_{\beta} M.$$

Also, by the subject-construction theorem (2B2), $\Gamma \vdash_{\lambda} M : \tau$ iff $\tau \equiv \rho \rightarrow \sigma$ for some ρ and σ such that

$$\Gamma, x:\rho \vdash_{\lambda} N : \sigma$$

and by the induction hypothesis the latter is equivalent to

$$(2) \quad \Gamma, x:\rho \vdash_{\lambda} N^* : \sigma.$$

We cannot simply choose $M^* \equiv \lambda x.N^*$ because this is not an applicative combination. Instead, an applicative combination called “ $[x] \cdot N^*$ ” will be constructed, such that

$$(3) \quad FV([x] \cdot N^*) = FV(N^*) - \{x\},$$

$$(4) \quad [x] \cdot N^* \triangleright_{\beta} \lambda x.N^*,$$

$$(5) \quad \Gamma \vdash_{\lambda} ([x] \cdot N^*) : \tau \iff \tau \equiv \rho \rightarrow \sigma \text{ and } \Gamma, x:\rho \vdash_{\lambda} N^* : \sigma.$$

Then M^* will be chosen to be $[x] \cdot N^*$ and 9F2 (i)–(iii) will follow from (1)–(5).

Before constructing $[x] \cdot N^*$, define

$$(6) \quad I^* \equiv \mathbf{CKK}, \quad S^* \equiv \mathbf{B(B(BW)C)(BB)}.$$

It is routine to check that I^* and S^* β -reduce to **I** and **S** with strong type-invariance.

Then $[x] \cdot N^*$ is constructed by the following algorithm (from Curry and Feys 1958 §6A3, omitting (c)).

- | | |
|---|---|
| (a) If $x \notin FV(N^*)$: | $[x] \cdot N^* \equiv \mathbf{KN}^*;$ |
| (b) if $N^* \equiv x$: | $[x] \cdot N^* \equiv I^*;$ |
| (d) if $N^* \equiv PQ, x \notin FV(P), x \in FV(Q)$: | $[x] \cdot N^* \equiv \mathbf{BP}([x] \cdot Q);$ |
| (e) if $N^* \equiv PQ, x \in FV(P), x \notin FV(Q)$: | $[x] \cdot N^* \equiv \mathbf{C}([x] \cdot P)Q;$ |
| (f) if $N^* \equiv PQ, x \in FV(P), x \in FV(Q)$: | $[x] \cdot N^* \equiv S^*([x] \cdot P)([x] \cdot Q).$ |

It is easy to see that $[x] \cdot N^*$ is defined for every applicative combination N^* of **B**, **C**, **K**, **W** and variables, and a routine induction on $|N^*|$ shows that (3)–(5) hold. \square

9F4 Definition A set \mathbb{S} of typable closed λ -terms is called a *typable basis* for a set \mathbb{L} of λ -terms iff there is an algorithm that accepts any member of \mathbb{L} and constructs an \mathbb{S} -and-variables combination M^* which β -reduces to M with strong type-invariance.

9F5 Partial Completeness Theorem (i) $\{\mathbf{B}, \mathbf{C}, \mathbf{I}, \mathbf{W}\}$ is a typable basis for the set of all $\lambda\mathbf{I}$ -terms (defined in 1D1).

(ii) $\{\mathbf{B}, \mathbf{C}, \mathbf{K}\}$ is a typable basis for the set of all \mathbf{BCK} λ -terms (1D2).

(iii) $\{\mathbf{B}, \mathbf{C}, \mathbf{I}\}$ is a typable basis for the set of all \mathbf{BCI} λ -terms (1D3).

Proof (i) Modify the algorithm given in 9F3 by omitting (a) and replacing I^* by **I** in (b).

(ii) Modify the algorithm in 9F3 by omitting (f).

(iii) In 9F3, omit (a) and (f) and replace I^* by **I** in (b). \square

Answers to starred exercises

2A8.7 Recall that (i) $\mathbf{B}' \equiv \lambda xyz. y(xz)$; (ii) $\mathbf{C} \equiv \lambda xyz. xzy$; (iii) $\mathbf{S} \equiv \lambda xyz. xz(yz)$; (iv) $\mathbf{W} \equiv \lambda xy. xyy$; the deductions are:

$$\begin{array}{c}
 \frac{x:a \rightarrow b \mapsto x:a \rightarrow b \quad z:a \mapsto z:a}{\mapsto} (\rightarrow E) \\
 \frac{y:b \rightarrow c \mapsto y:b \rightarrow c \quad x:a \rightarrow b, z:a \mapsto xz:b}{\mapsto} (\rightarrow E) \\
 \frac{x:a \rightarrow b, y:b \rightarrow c, z:a \mapsto y(xz):c}{\mapsto} (\rightarrow I) \quad 3 \text{ times} \\
 \mapsto (\lambda xyz. y(xz)):(a \rightarrow b) \rightarrow (b \rightarrow c) \rightarrow a \rightarrow c \\
 \\
 \frac{x:a \rightarrow b \rightarrow c \mapsto x:a \rightarrow b \rightarrow c \quad z:a \mapsto z:a}{\mapsto} (\rightarrow E) \\
 \frac{x:a \rightarrow b \rightarrow c, z:a \mapsto xz:b \rightarrow c \quad y:b \mapsto y:b}{\mapsto} (\rightarrow E) \\
 \frac{x:a \rightarrow b \rightarrow c, y:b, z:a \mapsto xzy:c}{\mapsto} (\rightarrow I) \quad 3 \text{ times} \\
 \mapsto (\lambda xyz. xzy):(a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c \\
 \\
 \frac{x:a \rightarrow b \rightarrow c \mapsto x:a \rightarrow b \rightarrow c \quad z:a \mapsto z:a}{\mapsto} (\rightarrow E) \quad \frac{y:a \rightarrow b \mapsto y:a \rightarrow b \quad z:a \mapsto z:a}{\mapsto} (\rightarrow E) \\
 \frac{x:a \rightarrow b \rightarrow c, z:a \mapsto xz:b \rightarrow c \quad y:a \rightarrow b, z:a \mapsto yz:b}{\mapsto} (\rightarrow E) \\
 \frac{x:a \rightarrow b \rightarrow c, y:a \rightarrow b, z:a \mapsto xz(yz):c}{\mapsto} (\rightarrow I) \quad 3 \text{ times} \\
 \mapsto (\lambda xyz. xz(yz)):(a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c \\
 \\
 \frac{x:a \rightarrow a \rightarrow b \mapsto x:a \rightarrow a \rightarrow b \quad y:a \mapsto y:a}{\mapsto} (\rightarrow E) \\
 \frac{x:a \rightarrow a \rightarrow b, y:a \mapsto xy:b \quad y:a \mapsto y:a}{\mapsto} (\rightarrow E) \\
 \frac{x:a \rightarrow a \rightarrow b, y:a \mapsto xyy:b}{\mapsto} (\rightarrow I) \quad \text{twice} \\
 \mapsto (\lambda xy. xyy):(a \rightarrow a \rightarrow b) \rightarrow a \rightarrow b
 \end{array}$$

2A8.8 The deductions are too wide to show in full here, so the context and arrow will be omitted from each formula. (This cannot be done for arbitrary deductions

without ambiguity but those below are simple enough.) First note that, by deductions like 2A8.3,

$$\vdash \mathbf{I} : (a \rightarrow b) \rightarrow a \rightarrow b, \quad \vdash \mathbf{I} : c \rightarrow c$$

(i) Let $M \equiv \lambda vxyz.v(y(vxz))$ and $P \equiv M\mathbf{I}$.

$$\begin{array}{c}
 \frac{v:(a \rightarrow b) \rightarrow a \rightarrow b \quad x:a \rightarrow b}{vx:a \rightarrow b} (\rightarrow E) \\
 \frac{vx:a \rightarrow b \quad z:a}{v(xz):b} (\rightarrow E) \\
 \frac{y:b \rightarrow a \rightarrow b \quad v(xz):b}{y(vxz):a \rightarrow b} (\rightarrow E) \\
 \frac{v:(a \rightarrow b) \rightarrow a \rightarrow b \quad y(vxz):a \rightarrow b}{v(y(vxz)):a \rightarrow b} (\rightarrow E) \\
 \frac{v(y(vxz)):a \rightarrow b}{M:((a \rightarrow b) \rightarrow a \rightarrow b) \rightarrow (a \rightarrow b) \rightarrow (b \rightarrow a \rightarrow b) \rightarrow a \rightarrow a \rightarrow b} (\rightarrow I) \quad 4 \text{ times} \\
 \frac{M:((a \rightarrow b) \rightarrow a \rightarrow b) \rightarrow (a \rightarrow b) \rightarrow (b \rightarrow a \rightarrow b) \rightarrow a \rightarrow a \rightarrow b \quad \mathbf{I}:(a \rightarrow b) \rightarrow a \rightarrow b}{M\mathbf{I}:(a \rightarrow b) \rightarrow (b \rightarrow a \rightarrow b) \rightarrow a \rightarrow a \rightarrow b}
 \end{array}$$

(ii) Let $Q \equiv \lambda xyz.\mathbf{I}(y(\mathbf{I}xz))$.

$$\begin{array}{c}
 \frac{\mathbf{I}:(a \rightarrow b) \rightarrow a \rightarrow b \quad x:a \rightarrow b}{\mathbf{I}x:a \rightarrow b} (\rightarrow E) \\
 \frac{\mathbf{I}x:a \rightarrow b \quad z:a}{\mathbf{I}xz:b} (\rightarrow E) \\
 \frac{y:b \rightarrow c \quad \mathbf{I}xz:b}{y(\mathbf{I}xz):b} (\rightarrow E) \\
 \frac{\mathbf{I}:c \rightarrow c \quad y(\mathbf{I}xz):b}{\mathbf{I}(y(\mathbf{I}xz)):c} (\rightarrow E) \\
 \frac{\mathbf{I}(y(\mathbf{I}xz)):c}{(\lambda xyz.\mathbf{I}(y(\mathbf{I}xz))):(a \rightarrow b) \rightarrow (b \rightarrow c) \rightarrow a \rightarrow c} (\rightarrow I) \quad 3 \text{ times}
 \end{array}$$

2B3.2 (T. Coquand, informal correspondence 1993.) Let $\Gamma \mapsto M:\tau$ be the conclusion of Δ and Δ' . Then M has a β -nf M^* by WN (2D5), and by 1B9 there is a leftmost reduction ρ from M to M^* . Such a reduction consists of the following successive parts (each of which may be empty):

(1) A reduction (called a *head reduction of order 0*) in which each step has form

$$(\lambda y.P)QR_1 \dots R_k \triangleright_1 ([Q/y]P)R_1 \dots R_k \quad (k \geq 0).$$

(2) A series of reductions ρ_1, \dots, ρ_m ($m \geq 0$), where ρ_i has form

$$\lambda x_1 \dots x_i. N \triangleright \lambda x_1 \dots x_i. N'$$

where $N \triangleright N'$ by a head reduction of order 0. (We call ρ_i a *head reduction of order i*.)

(3) A reduction (called an *internal reduction*) with form

$$\lambda x_1 \dots x_m. yN_1 \dots N_n \triangleright \lambda x_1 \dots x_m. yN_1^* \dots N_n^* \equiv M^*,$$

where $N_i \triangleright N_i^*$ by a leftmost reduction, for $i = 1, \dots, n$.

By the proof of the Subject-reduction theorem 2C1, Δ and Δ' reduce to deductions of

$$\Gamma^* \mapsto M^*:\tau, \quad \Gamma'^* \mapsto M^*:\tau,$$

for some $\Gamma^* \subseteq \Gamma$ and $\Gamma'^* \subseteq \Gamma$. (Reducing a deduction is like reducing a typed term, 5B8.) But $\Gamma^* = \Gamma'^* = \Gamma$, because $FV(M^*) = FV(M)$ since reductions of λ I-terms do not cancel, and so

$$\text{Subjects}(\Gamma^*) = \text{Subjects}(\Gamma'^*) = FV(M^*) = FV(M) = \text{Subjects}(\Gamma).$$

Hence by 2B4, Δ and Δ' both reduce to the same deduction.

Now prove $\Delta \equiv \Delta'$ by induction on $|M^*|$ with an induction on the length of the reduction ρ in both the basis and induction steps. (The case $\rho = \emptyset$ is 2B4.)

Basis: $M^* \equiv y$. Then (2) and (3) are empty. For each step (1) in ρ , if the deduction for the right side of (1) is unique, then the types and deductions for Q, P, R_1, \dots, R_k are uniquely determined. (Note that Q occurs in $[Q/y]P$ because y occurs in P .) Hence the deduction for the left side is uniquely determined.

Induction step: $M^* \equiv \lambda x_1 \dots x_m. y N_1^* \dots N_n^*$, $m + n \geq 1$. For steps (1) use the above argument and for (2) and (3) use the hypothesis of the induction on $|M^*|$.

3A6.1 Let $\mathbf{B} \equiv \lambda xyz. x(yz)$ and let Δ be any TA_λ -deduction of a type ξ for \mathbf{B} . By 2B2(iv) the last three steps in Δ must be applications of rule $(\rightarrow I)$, with form

$$\begin{array}{c} x:\alpha, y:\beta, z:\sigma \mapsto x(yz):\tau \\ \hline x:\alpha, y:\beta \mapsto \lambda z. x(yz):\sigma \rightarrow \tau \\ \hline x:\alpha \mapsto \lambda yz. x(yz):\beta \rightarrow \sigma \rightarrow \tau \\ \hline \mapsto \lambda xyz. x(yz):\alpha \rightarrow \beta \rightarrow \sigma \rightarrow \tau \end{array}$$

for some $\alpha, \beta, \sigma, \tau$. Then by 2B2(iii) the step above these must be $(\rightarrow E)$, and its premises must have form

$$x:\rho \rightarrow \tau \mapsto x:\rho \rightarrow \tau, \quad y:\beta, z:\sigma \mapsto yz:\rho$$

for some ρ . Hence $\alpha \equiv \rho \rightarrow \tau$. Also by 2B2(iii) the step above $yz:\rho$ must be $(\rightarrow E)$, with premises of form

$$y:\sigma \rightarrow \rho \mapsto y:\sigma \rightarrow \rho, \quad z:\sigma \mapsto z:\sigma.$$

Hence $\beta \equiv \sigma \rightarrow \rho$, and the type assigned to \mathbf{B} must have form

$$(\rho \rightarrow \tau) \rightarrow (\sigma \rightarrow \rho) \rightarrow \sigma \rightarrow \tau.$$

3B5.2 (i) $[a/b] \circ [b/a] \equiv [a/b, a/a]$. For example if $\tau \equiv a \rightarrow b$, we have

$$([a/b] \circ [b/a])(\tau) \equiv a \rightarrow a, \quad [a/b]([b/a](\tau)) \equiv [a/b](b \rightarrow b) \equiv a \rightarrow a.$$

(Warning: $[a/b] \circ [b/a] \neq [a/a]$! To see this, apply $[a/a]$ to τ .)

(ii) Let $s \equiv [\sigma_1/a_1, \dots, \sigma_n/a_n]$. If $\text{Range}(s) \cap \text{Dom}(s) = \emptyset$, define

$$s_\tau^* \equiv [\sigma_1/a_1] \circ \dots \circ [\sigma_n/a_n]$$

If $\text{Range}(s) \cap \text{Dom}(s) \neq \emptyset$, choose distinct $b_1, \dots, b_n \notin \text{Range}(s) \cup \text{Dom}(s) \cup \text{Vars}(\tau)$, and define $\sigma_i^* \equiv [b_i/a_i]\sigma_i$ for each $i \leq n$; then define

$$s_\tau^* =_{\text{ext}} [a_1/b_1] \circ \dots \circ [a_n/b_n] \circ [\sigma_1^*/a_1] \circ \dots \circ [\sigma_n^*/a_n].$$

(Warning: The dependence of s_τ^* on τ is real; the definition of s_τ^* includes the clause “ $b_1, \dots, b_n \notin \text{Vars}(\tau)$ ” and this clause is more important than it looks: no claim is made that there is a composition s^* of single substitutions such that $s^*(\tau) \equiv s(\tau)$ for all τ .)

3C3.1 Let $\rho \equiv a \rightarrow (b \rightarrow c)$, $\tau \equiv (a \rightarrow b) \rightarrow a$, $v_0 \equiv ((b \rightarrow c) \rightarrow d) \rightarrow (b \rightarrow c)$. By 3C2.1, v_0 is a c.i. of ρ and τ . To show v_0 is most general, let v be any other c.i., say

$$v \equiv s_1(a) \rightarrow (s_1(b) \rightarrow s_1(c)) \equiv (s_2(a) \rightarrow s_2(b)) \rightarrow s_2(a).$$

Then

$$s_1(a) \equiv s_2(a) \rightarrow s_2(b), \quad s_2(a) \equiv s_1(b) \rightarrow s_1(c),$$

so

$$s_1(a) \equiv (s_1(b) \rightarrow s_1(c)) \rightarrow s_2(b),$$

and hence v is an instance of v_0 .

3D2.1 Let $\rho \equiv a \rightarrow (b \rightarrow b)$, $\tau \equiv (c \rightarrow c) \rightarrow a$, $\mathfrak{u} \equiv [(b \rightarrow b)/a, b/c]$. First, \mathfrak{u} clearly unifies $\{\rho, \tau\}$. To prove that \mathfrak{u} is most general, let s be any other unifier, say

$$s \equiv [\alpha/a, \beta/b, \gamma/c],$$

where

$$\alpha \rightarrow (\beta \rightarrow \beta) \equiv (\gamma \rightarrow \gamma) \rightarrow \alpha.$$

Then, by comparing the two sides of this identity, we have

$$\alpha \equiv \gamma \rightarrow \gamma, \quad \alpha \equiv \beta \rightarrow \beta.$$

Hence $\beta \equiv \gamma$ and

$$s \equiv [(\beta \rightarrow \beta)/a, \beta/b, \beta/c] \equiv [\beta/b] \circ \mathfrak{u}.$$

6B4.3 Respectively, $(\Delta_5)_\lambda$ and $(\Delta_6)_\lambda$ are

$$\frac{\frac{x:a \mapsto x:a}{x:a \mapsto (\lambda y.x):a \rightarrow a} (\rightarrow I)_{\text{vac}}}{\mapsto (\lambda xy.x):a \rightarrow a \rightarrow a} (\rightarrow I)_{\text{main}} \qquad \frac{\frac{x:a \mapsto x:a}{\mapsto (\lambda x.x):a \rightarrow a} (\rightarrow I)_{\text{main}}}{\mapsto (\lambda yx.x):a \rightarrow a \rightarrow a} (\rightarrow I)_{\text{vac}}$$

7A3.2 (M. W. Bunder) Choose $P \equiv (\lambda vxyz.v(y(vxz)))\mathbf{I}$, $P' \equiv (\lambda vxyz.vy(x(vz)))\mathbf{I}$. Then $P =_\beta P'$, and by an easy computation

$$\begin{aligned} PT(P) &\equiv (a \rightarrow b) \rightarrow (b \rightarrow a \rightarrow b) \rightarrow a \rightarrow a \rightarrow b, \\ PT(P') &\equiv ((c \rightarrow d) \rightarrow c) \rightarrow (c \rightarrow d) \rightarrow (c \rightarrow d) \rightarrow d. \end{aligned}$$

These two types have no common substitution-instance, because if such an instance were obtained by substitutions $[\alpha/a, \beta/b]$ and $[\gamma/c, \delta/d]$ we would get

$$\alpha \equiv \gamma \rightarrow \delta, \quad \alpha \rightarrow \beta \equiv \delta,$$

which would imply the impossible identity $\alpha \equiv \gamma \rightarrow \alpha \rightarrow \beta$.

7C7.2 (i) Let $M \equiv \mathbf{I}$ and $\tau \equiv (b \rightarrow b) \rightarrow b \rightarrow b$. Define $\tau^\circ \equiv (a \rightarrow b) \rightarrow c \rightarrow d$; then τ° is skeletal and changes to τ when we make the identifications $[b/a], [b/c], [b/d]$. The algorithm begins by applying the proof of 7C2, which begins by applying the proof of 7C1 to build a term whose PT is $\tau^\circ \rightarrow \tau^\circ$; and the procedure in that proof gives

$$I_{\tau^\circ} \equiv \lambda xy. (\lambda uv. \mathbf{I}(u(\mathbf{I}v)))(x((\lambda uv. \mathbf{I}(u(\mathbf{I}v)))y)).$$

Then, following the proof of 7C2, the algorithm defines

$$M^+ \equiv I_{\tau^\circ} M \equiv I_{\tau^\circ} \mathbf{I}.$$

It then applies the PT algorithm to compute $\tau^+ \equiv PT(M^+)$; in fact in this case $\tau^+ \equiv \tau^\circ$ and the identifications s_1, \dots, s_k in 7C7 are $[b/a], [b/c], [b/d]$ (and $k = 3$). The next step is to apply the proof of 7C5 to build three terms N_1, N_2, N_3 in turn, such that

$$\begin{aligned} PT(N_1) &\equiv [b/d]\tau^+ && \equiv (a \rightarrow b) \rightarrow c \rightarrow b, \\ PT(N_2) &\equiv [b/c][b/d]\tau^+ && \equiv (a \rightarrow b) \rightarrow b \rightarrow b, \\ PT(N_3) &\equiv [b/a][b/c][b/d]\tau^+ && \equiv (b \rightarrow b) \rightarrow b \rightarrow b. \end{aligned}$$

To do this, the algorithm first applies 7C4 to obtain three terms R_1, R_2, R_3 such that

$$\begin{aligned} PT(R_1) &\equiv (d \rightarrow f) \rightarrow (g \rightarrow b) \rightarrow ((a \rightarrow b) \rightarrow c \rightarrow d) \rightarrow ((a \rightarrow g) \rightarrow c \rightarrow f), \\ PT(R_2) &\equiv (f \rightarrow c) \rightarrow (g \rightarrow b_1) \rightarrow ((a \rightarrow b_1) \rightarrow c \rightarrow b_2) \rightarrow ((a \rightarrow g) \rightarrow f \rightarrow b_2), \\ PT(R_3) &\equiv (a \rightarrow f) \rightarrow (g \rightarrow b_1) \rightarrow ((a \rightarrow b_1) \rightarrow b_2 \rightarrow b_3) \rightarrow ((f \rightarrow g) \rightarrow b_2 \rightarrow b_3). \end{aligned}$$

Then the algorithm defines

$$N_1 \equiv (\lambda x. R_1 x x) \mathbf{I}(I_{\tau^\circ} \mathbf{I}), \quad N_2 \equiv (\lambda x. R_2 x x) \mathbf{I}N_1, \quad N_3 \equiv (\lambda x. R_3 x x) \mathbf{I}N_2.$$

Finally it defines $M^* \equiv N_3$. (By the way, the algorithm is not claimed to be efficient! There exists a much shorter M^* than the one above, namely $M^* \equiv \lambda xy. x(xy)$.)

(ii) Let $M \equiv \mathbf{K}$, $\tau \equiv b \rightarrow b \rightarrow b$. Define $\tau^\circ \equiv a \rightarrow b \rightarrow c$; then τ° has the 1-property and changes to τ under the identifications $[b/a], [b/c]$. The algorithm begins by applying the proof of 7C1 to build a term whose PT is $\tau^\circ \rightarrow \tau^\circ$; this term is

$$I_{\tau^\circ} \equiv \lambda xy. (\lambda uv. \mathbf{I}(u(\mathbf{I}v)))(x(\mathbf{I}y)).$$

Then, following the proof of 7C2, the algorithm defines

$$M^+ \equiv I_{\tau^\circ} M \equiv I_{\tau^\circ} \mathbf{K}$$

and computes $\tau^+ \equiv PT(M^+)$; in this case $\tau^+ \equiv a \rightarrow b \rightarrow a$ and for the identifications s_1, \dots, s_k in 7C7 we have $k = 1$ and $s_1 \equiv [b/a]$. The next step is to apply 7C5's proof to build N such that

$$PT(N) \equiv [b/a]\tau^+ \equiv b \rightarrow b \rightarrow b.$$

To do this, the algorithm first applies 7C4 to obtain a term R such that

$$PT(R) \equiv (f \rightarrow a_1) \rightarrow (g \rightarrow b) \rightarrow (a_1 \rightarrow b \rightarrow a_2) \rightarrow (f \rightarrow g \rightarrow a_2),$$

and then defines $N \equiv (\lambda x. Rx x)(I_{\tau} \mathbf{K})$.

8A12.1 The eight regions contain the following terms in order from left to right.

Top row: $\lambda xy. xy$, $\lambda xyz. x(xyyy)z$, $\lambda xyz u. x(xyyy)zu$, $\lambda xyz u. x y z u$;
 bottom row: $\lambda x. x$, $\lambda xyz. xz(xyyy)$, $\lambda xyz u. xu(xyyy)z$, $\lambda xyz u. uxyz$.

8B7 For items 1, 6, 8, 11 in Table 8B7a see 8B3–8B6. For the rest, see the answer to 8C6.4. (In item 12, $P_0 \notin Nprinc(\tau)$ since the PT of P_0 is $((a \rightarrow a) \rightarrow b) \rightarrow b$ by the PT algorithm, 3E1.)

8C6.4 For rows 6, 8 and 11 of Table 8B7a see Examples 8C6.1–3. The other rows are dealt with below. (For ease of reading, types are omitted and $x^d M$ is used for $x(x \dots (xM) \dots)$ with d x 's.)

1. $\mathcal{A}(\tau, 0) = \{V\}$, $\mathcal{A}(\tau, 1) = \emptyset$.
2. $\mathcal{A}(\tau, 0) = \{V\}$, $\mathcal{A}(\tau, 1) = \{\lambda x_1. x_1\}$.
3. $\mathcal{A}(\tau, 0) = \{V\}$, $\mathcal{A}(\tau, 1) = \{\lambda x_1 x_2. x_1\}$.
4. $\mathcal{A}(\tau, 0) = \{V\}$, $\mathcal{A}(\tau, 1) = \{\lambda x_1 x_2 x_3. x_1 V_1\}$, $\mathcal{A}(\tau, 2) = \{\lambda x_1 x_2 x_3. x_1(x_2 V_2)\}$,
 $\mathcal{A}(\tau, 3) = \{\lambda x_1 x_2 x_3. x_1(x_2 x_3)\}$.
5. $\mathcal{A}(\tau, 0) = \{V\}$, $\mathcal{A}(\tau, 1) = \{\lambda x_1 x_2 x_3. x_1 V_1 V_2\}$, $\mathcal{A}(\tau, 2) = \{\lambda x_1 x_2 x_3. x_1 x_3 x_2\}$.
7. $\mathcal{A}(\tau, 0) = \{V\}$, $\mathcal{A}(\tau, 1) = \{\lambda x_1 x_2. x_1 V_1 V_2\}$, $\mathcal{A}(\tau, 2) = \{\lambda x_1 x_2. x_1 x_2 x_2\}$.
9. $\mathcal{A}(\tau, 0) = \{V\}$, $\mathcal{A}(\tau, 1) = \{\lambda x_1 x_2. x_1 V_1, \lambda x_1 x_2. x_2\}$
 $\mathcal{A}(\tau, d) = \{\lambda x_1 x_2. x_1^d V_d, \lambda x_1 x_2. x_1^{d-1} x_2\}$ for all $d \geq 2$.
10. $\mathcal{A}(\tau, 0) = \{V\}$, $\mathcal{A}(\tau, 1) = \{\lambda x_1 x_2. x_1 V_1\}$, $\mathcal{A}(\tau, 2) = \{\lambda x_1 x_2. x_1 x_2\}$.
12. $\mathcal{A}(\tau, 0) = \{V\}$, $\mathcal{A}(\tau, 1) = \{\lambda x. x V_1\}$
 $\mathcal{A}(\tau, 2) = \{\lambda x. x(\lambda y_1. x V_2), \lambda x. x(\lambda y_1. y_1)\}$
 $\mathcal{A}(\tau, 3) = \{\lambda x. x(\lambda y_1. y_1), \lambda x. x(\lambda y_1. x(\lambda y_2. x V_3))\}$
 $\lambda x. x(\lambda y_1. x(\lambda y_2. y_1)), \lambda x. x(\lambda y_1. x(\lambda y_2. y_2))\}$
 $\mathcal{A}(\tau, d) = \{\lambda x. x(\lambda y_1. x(\dots(\lambda y_{d-1}. x V_d) \dots)),$
 $\lambda x. x(\lambda y_1. x(\dots(\lambda y_{d-2}. x(\lambda y_{d-1}. y_1)) \dots)), \dots$
 $\lambda x. x(\lambda y_1. x(\dots(\lambda y_{d-2}. x(\lambda y_{d-1}. y_{d-1})) \dots))\}$ for all $d \geq 4$.

8E7.3 For (iii), use (ii) and the fact that $\#(\mathcal{A}(\tau, 0)) = 1$ (since $\mathcal{A}(\tau, 0) = \{V^\tau\}$ by Step 0 of Algorithm 8C6).

For (i), use induction on d . The basis is trivial since $\mathcal{A}(\tau, 0) = \{V^\tau\}$.

For the induction step (d to $d+1$), let $X^\tau \in \mathcal{A}(\tau, d)$ contain q metavariables where $1 \leq q \leq |\tau|^d$, and let V^ρ be one of these.

Consider Part IIa1 of Step $d+1$ of Algorithm 8C6: using the notation of IIa1, note that each suitable replacement Y_i^ρ generated by IIa1 for V^ρ contains $\leq n_i$ metavariables, where n_i is the arity of σ_i . But σ_i occurs in ρ which occurs in τ by 8E7.1, so

$$n_i \leq |\sigma_i| - 1 \leq |\tau| - 1 < |\tau|.$$

Next consider Part IIa2 in 8C6. Using the notation of IIa2, note that each suitable replacement Z_j generated by IIa2 for V^ρ contains $\leq h$ metavariables, where h_j is the arity of ζ_j . But ζ_j occurs in τ by 2B3(i), so

$$h_j \leq |\zeta_j| - 1 \leq |\tau| - 1 < |\tau|.$$

Thus each metavariable in X^τ is replaced by $< |\tau|$ new ones, so the total number of metavariables in the resulting extension of X^τ is $< q|\tau|$. Hence (i) holds.

To prove (ii), look at the above induction step in more detail. When IIa1 in 8C6 is applied to V^ρ the number of suitable replacements it generates is $\leq m$, where m is the arity of ρ . We have

$$m \leq |\rho| - 1 \leq |\tau| - 1.$$

When IIa2 in 8C6 is applied to V^ρ the number of suitable replacements it generates is $\leq t$, and by 8E7.2(ii),

$$t \leq (|\tau| - 1)\text{Depth}(X^\tau).$$

Thus the total number of suitable replacements for V^ρ is less than or equal to

$$|\tau| - 1 + (|\tau| - 1)d, < |\tau|(d + 1).$$

But there are q metavariables in X^τ , so when we apply IIa2 the total number of resulting extensions of X^τ is less than $(|\tau|(d + 1))^q$. Hence

$$\#(\mathcal{A}(\tau, d + 1)) < (|\tau|(d + 1))^q \times \#(\mathcal{A}(\tau, d)).$$

Then (ii) follows by (i).

Bibliography

References to unpublished manuscripts have been avoided as far as possible, as they are likely to be unavailable to most readers. A few have been included for historical and other reasons, however.

- AHO, A. V., SETHI, R., ULLMAN, J. D. [1986] **Compilers**, Addison-Wesley Co., USA 1986.
- ANDERSON, A. R., BELNAP, N. D. [1975] **Entailment, Vol. I**, Princeton University Press, USA 1975.
- ANDERSON, A. R., BELNAP, N. D., DUNN, J. M. [1992] **Entailment, Vol. II**, Princeton University Press, USA 1992.
- ANDREWS, P. B. [1965] **A transfinite type theory with type variables**, North-Holland Co., Netherlands 1965.
- ANDREWS, P. B. [1971] *Resolution in type theory*, **J. Symbolic Logic** 36 (1971), 414–432.
- ANDREWS, P. B. [1986] **An Introduction to Mathematical Logic and Type Theory; to Truth through Proof**, Academic Press, USA and UK 1986.
- AVRON, A. [1988] *The semantics and proof theory of linear logic*, **Theoretical Computer Science** 57 (1988), 161–184.
- AVRON, A. [1992] *Axiomatic systems, deduction and implication*, **J. Logic and Computation** 2 (1992), 51–98.
- BAADER, F., SIEKMANN, J. H. [1994] *Unification theory*, in **Handbook of Logic in Artificial Intelligence, Vol. 2: Deduction Methodologies**, ed. D. M. Gabbay, C. J. Hogger, J. A. Robinson, Oxford University Press, UK 1994, pp. 41–126.
- BARENDREGT, H. P. [1984] **The Lambda Calculus**, North-Holland Co., Netherlands, 2nd. edition 1984.
- BARENDREGT, H. P. [1992] *Lambda calculi with types*, in **Handbook of Logic in Computer Science, Vol. 2**, ed. S. Abramsky et al., Clarendon Press, UK 1992, pp. 117–309.
- BARENDREGT, H. P., COPPO, M., DEZANI, M. [1983] *A filter lambda model and the completeness of type assignment*, **J. Symbolic Logic** 48 (1983), 931–940.
- BEN-YELLES, C.-B. [1979] **Type-assignment in the lambda-calculus; syntax and semantics**, thesis 1979, Mathematics Dept., University of Wales Swansea, Swansea SA2 8PP, UK.
- BLOK, W. J., PIGOZZI, D. [1989] **Algebraizable Logics**, Memoirs of the American Mathematical Society No. 396 (1989), Amer. Math. Soc., Providence, R.I. 02901, USA.
- de BRUIJN, N. G. [1980] *A survey of the project AUTOMATH*, in **To H. B. Curry**, ed. J. P. Seldin, J. R. Hindley, Academic Press, UK 1980, pp. 579–606.
- BUNDER, M. W. [1982] *Deduction theorems for weak implicational logics*, **Studia Logica** 41 (1982), 95–108.
- BUNDER, M. W. [1986] Review no. 03004, in **Zentralblatt für Mathematik** 574 (1986), 10–11, of Bunder, M. W. and Meyer, R. K. *A result for combinators, BCK logics and BCK algebras*, **Logique et Analyse** 28 (1985), 33–40.
- BUNDER, M. W. [1991] *Corrections to some results for BCK logics and algebras*, **Logique et Analyse** 31 (1991), 115–122.

- BUNDER, M. W. [1996] *Standardization of proofs in propositional logic*, in **Logic and Reality** (Proceedings of the 1989 conference in Christchurch, N. Z. in memory of A. N. Prior), ed. J. Copeland, Oxford University Press, UK, 1996.
- BUNDER, M. W. [1993] *Theorems in classical logic are substitution instances of theorems in condensed BCI logic*, in **Substructural logics**, ed. P. Schroeder-Heister, K. Dosen, Clarendon Press, UK 1993, pp. 43–62.
- CHURCH, A. [1940] *A formulation of the simple theory of types*, **J. Symbolic Logic** 5 (1940), 56–68.
- CHURCH, A. [1941] **The Calculi of Lambda Conversion**, Princeton University Press, 1941. Reprinted by University Microfilms, Inc., Ann Arbor, Michigan, USA 1963.
- CHURCH, A. [1951] *The weak theory of implication*, in **Kontrolliertes Denken, Untersuchungen zum Logikkalkül und zur Logik der Einzelwissenschaften (Festgabe zum 60. Geburtstag von Prof. W. Britzelmayr)**, ed. A. Menne, A. Wilhelmy, H. Angstl; publisher Kommissionsverlag Karl Alber, München 1951 (Sonderdruck); pp. 22–37.
- CHURCH, A. [1976] *Schröder's anticipation of the simple theory of types*, **Erkenntnis** 10 (1976), 407–411. (Presented at the 5th Internat. Congress for the Unity of Science, 1939, and intended for publication in **J. Unified Science (Erkenntnis)** Vol. 9 which did not appear.)
- CONSTABLE, R. L. [1991] *Type theory as a foundation for computer science*, in **Theoretical Aspects of Computer Software**, ed. by T. Ito, A. R. Meyer, **Lecture Notes in Computer Science**, Springer-Verlag, Germany, No. 526 (1991), 226–243.
- COPPO, M. [1984] *Completeness of type-assignment in continuous lambda models*, **Theoretical Computer Science** 29 (1984), 309–324.
- COPPO, M., DEZANI, M. [1978] *A new type-assignment for lambda terms*, **Archiv für Mathematische Logik** 19 (1978), 139–156.
- CROSSLEY, J. N., SHEPHERDSON, J. C. [1993] *Extracting programs from proofs by an extension of the Curry-Howard process*, in **Logical Methods**, ed. J. N. Crossley, J. B. Remuel, R. A. Shore, M. E. Sweedler, Birkhäuser, Boston, Mass., USA 1993, pp. 222–288.
- CURRY, H. B. [1934] *Functionality in combinatory logic*, **Proc. National Academy of Sciences of the USA** 20 (1934), 584–590.
- CURRY, H. B. [1942] *The combinatory foundations of mathematical logic*, **J. Symbolic Logic** 7 (1942), 49–64.
- CURRY, H. B. [1966] *Technique for evaluating principal functional character*, Notes dated March 17th 1966, in Curry archive, Pennsylvania State University, Univ. Park, PA 16802, USA
- CURRY, H. B. [1969] *Modified basic functionality in combinatory logic*, **Dialectica** 23 (1969), 83–92.
- CURRY, H. B., FEYS, R. [1958] **Combinatory Logic**, Vol. I, North-Holland Co., Netherlands 1958.
- CURRY, H. B., HINDLEY, J. R., SELDIN, J. P. [1972] **Combinatory Logic**, Vol. II, North-Holland Co., Netherlands 1972.
- van DALEN, D. [1980] **Logic and Structure**, Springer-Verlag, Germany, 1980.
- DAMAS, L. [1984] *Type Assignment in Programming Languages*, Ph.D. thesis, Computer Science Dept., University of Edinburgh, King's Buildings, Mayfield Rd., Edinburgh, UK 1984.
- DAMAS, L., MILNER, R. [1982] *Principal type-schemes for functional programming languages*, in **Proc. Ninth Annual A.C.M. Symposium on the Principles of Programming Languages (P.O.P.L.)**, 1982, Assoc. for Computing Machinery, New York, USA 1982, pp. 207–212.
- DEKKERS, W. [1988] *Reducibility of types in typed lambda calculus*, **Information and Computation** 77 (1988), 131–137.
- DILLER, J. [1968] *Zur Berechenbarkeit primitiv-rekursiver Funktionale endlicher Typen*, in **Contributions to Mathematical Logic**, ed. H.A. Schmidt et al., North-Holland Co., Netherlands 1968, pp. 109–120.

- DOSEN, K. [1992a] *Modal translations in substructural logics*, **J. Philosophical Logic** 21 (1992), 283–336.
- DOSEN, K. [1992b] *The first axiomatization of relevant logic*, **J. Philosophical Logic** 21 (1992), 339–356.
- DRAGALIN, A. G. [1968] *The computation of primitive recursive terms of finite type, and primitive recursive realization*, **Zapiski Nauchnyh Seminarov Leningradskogo Otdelenija Matematicheskogo Instituta im V.I. Steklova Akademii Nauk S.S.S.R. (L.O.M.I.)** 8 (1968), 32–45.
- DWORK, C., KANELAKIS, P. C., MITCHELL, J. C. [1984] *On the sequential nature of unification*, **J. Logic Programming** 1 (1984), 35–50.
- DYCKHOFF, R. [1992] *Contraction-free sequent calculi for intuitionistic logic*, **J. Symbolic Logic** 57 (1992), 795–807.
- FITCH, F. B. [1936] *A system of formal logic without an analogue to the Curry W operator*, **J. Symbolic Logic** 1 (1936), 92–100.
- FORTUNE, S., LEIVANT, D., O'DONNELL, M. [1983] *The expressiveness of simple and second-order type structures*, **J. Assoc. for Computing Machinery** 30 (1980) 151–185.
- GALLIER, J. H. [1990] *On Girard's "Candidats de déductibilité"*, in Odifreddi 1990, pp. 123–203.
- GALLIER, J. H. [1993] *Constructive logics. Part 1: a tutorial on proof systems and typed λ -calculi*, **Theoretical Computer Science** 110 (1993), 249–339.
- GANDY, R. O. [1977] *The simple theory of types*, in **Logic Colloquium** 76, ed. R. O. Gandy, M. Hyland, North-Holland Co., Netherlands 1977, pp. 173–181.
- GANDY, R. O. [1980a] *An early proof of normalization by A. M. Turing*, in **To H. B. Curry**, ed. J. R. Hindley, J. P. Seldin, Academic Press, UK 1980, pp. 453–455.
- GANDY, R. O. [1980b] *Proofs of strong normalization*, in **To H. B. Curry**, ed. J. R. Hindley, J. P. Seldin, Academic Press, UK. 1980, pp. 457–477.
- GENTZEN, G. [1935] *Untersuchungen über das logische Schliessen*, **Math. Zeitschrift** 39 (1935), 176–210, 405–431. English transl. as *Investigations into logical deduction in The Collected Papers of Gerhard Gentzen*, ed. M. E. Szabo, North-Holland Co., Netherlands 1969.
- GIANNINI, P., HONSELL, F., RONCHI DELLA ROCCA, S. [1993] *Type inference: some results, some problems*, **Fundamenta Informaticae** 19 (1993), 87–125.
- GIRARD, J. -Y. [1987] *Linear logic*, **Theoretical Computer Science** 50 (1987), 1–101.
- GIRARD, J. -Y., LAFONT, Y., TAYLOR, P. [1989] **Proofs and Types**, Cambridge University Press, UK 1989.
- GLADSTONE, M. D. [1965] *Some ways of constructing a propositional calculus of any required degree of unsolvability*, **Trans. American Math. Soc.** 118 (1965), 192–210.
- GRZEGORCZYK, A. [1953] *Some Classes of Recursive Functions*, **Rozprawy Matematyczne**, Państwowe Wydawnictwo Naukowe, Warszawa, Poland 1953.
- HAMILTON, A. [1988] **Logic for Mathematicians**, Cambridge University Press, UK 1978, revised edition 1988.
- HANATANI, Y. [1966] *Calculabilité des fonctionnels récursives primitives de type fini sur les nombres naturels*, **Ann. Japan Assoc. for the Philosophy of Science** 3 (1966), 19–30. Revised version: *Calculability of the primitive recursive functionals of finite type over the natural numbers*, in **Proof Theory Symposium, Kiel 1974**, ed. by J. Diller, G. H. Müller, **Lecture Notes in Mathematics** 500 (1975), Springer-Verlag, Germany, pp. 152–163.
- HANKIN, C. [1994] *Lambda Calculi*, Clarendon Press, Oxford, UK 1994.
- HENKIN, L. [1950] *Completeness in the theory of types*, **J. Symbolic Logic** 15 (1950), 81–91.
- HERBRAND, J. [1930] *Recherches sur la théorie de la démonstration*, **Trav. Soc. Sci. Lett. Varsovie, Classe III Sci. Math. Phys.**, No. 33 (1930), English transl. in **Jacques Herbrand Logical Writings**, ed. W.D. Goldfarb, Harvard University Press, USA 1971, pp. 44–202.
- HEYTING, A. [1955] **Intuitionism, an Introduction**, North-Holland Co., Netherlands 1955 (3rd edition 1971).

- HINATA, S. [1967] *Calculability of primitive recursive functionals of finite type*, **Sci. Report Tokyo Kyoiku Daigaku**, Section A, 9 (226) (1967), 42–59.
- HINDLEY, J. R. [1969] *The principal type-scheme of an object in combinatory logic*, **Trans. American Math. Soc.** 146 (1969), 29–60.
- HINDLEY, J. R. [1983a] *The completeness theorem for typing λ -terms*, **Theoretical Computer Science** 22 (1983), 1–17.
- HINDLEY, J. R. [1983b] *Curry's type-rules are complete with respect to the F-semantics too*, **Theoretical Computer Science** 22 (1983), 127–133.
- HINDLEY, J. R. [1989] *BCK-combinators and linear λ -terms have types*, **Theoretical Computer Science** 64 (1989), 97–106.
- HINDLEY, J. R. [1992] *Types with intersection, an introduction*, **Formal Aspects of Computing** 4 (1992), 470–486.
- HINDLEY, J. R. [1993] *BCK and BCI logics, condensed detachment and the 2-property*, **Notre Dame J. Formal Logic** 34 (1993), 231–250.
- HINDLEY, J. R., MEREDITH, D. [1990] *Principal type-schemes and condensed detachment*, **J. Symbolic Logic** 55 (1990), 90–105.
- HINDLEY, J. R., SELDIN, J. P. [1986] [HS 86] **Introduction to Combinators and λ -calculus**, Cambridge University Press, UK 1986.
- HIROKAWA, S. [1991a] *Principal type assignment to lambda terms*, **International Journal of the Foundations of Computer Science** 2 (1991), 149–162.
- HIROKAWA, S. [1991b] *Principal type-schemes of BCI-lambda-terms*, in **Theoretical Aspects of Computer Software**, ed. by T. Ito, A. R. Meyer, **Lecture Notes in Computer Science**, Springer-Verlag, Germany, No. 526 (1991), 633–650.
- HIROKAWA, S. [1991c] *BCK-formulas having unique proofs*, in **Category Theory and Computer Science**, ed. by D. H. Pitt et al., **Lecture Notes in Computer Science**, Springer-Verlag, Germany, No. 530 (1991), 106–120.
- HIROKAWA, S. [1992a] *The converse principal type-scheme theorem in lambda calculus*, **Studia Logica** 51 (1992), 83–95.
- HIROKAWA, S. [1992b] *Balanced formulas, BCK-minimal formulas and their proofs*, in **Logical Foundations of Computer Science – Tver '92**, ed. by A. Nerode, M. Taitslin, **Lecture Notes in Computer Science**, Springer-Verlag, Germany, No. 620 (1992), 198–208.
- HIROKAWA, S. [1993a] *Principal types of BCK-lambda-terms*, **Theoretical Computer Science** 107 (1993), 253–276.
- HIROKAWA, S. [1993b] *The relevance graph of a BCK-formula*, **J. Logic and Computation** 3 (1993), 269–285.
- HIROKAWA, S. [1993c] *The number of proofs for an implicational formula*, **J. Symbolic Logic** 58 (1993), 1117 (abstract only; fuller MS informally circulated 1991).
- HOPCROFT, J. E., ULLMAN, J. D., [1979] **Introduction to Automata Theory, Languages, and Computation**, Addison-Wesley, USA 1979.
- HOWARD, W. [1969] *The formulae-as-types notion of construction*, MS 1969, publ. in **To H. B. Curry**, ed. J. R. Hindley, J. P. Seldin, Academic Press, UK 1980, pp. 479–490.
- HOWARD, W. [1970] *Assignment of ordinals to terms for primitive recursive functionals of finite type*, in **Intuitionism and Proof Theory (Proc. Buffalo Conference 1968)**, ed. A. Kino, J. Myhill, R. Vesley, North-Holland Co., Netherlands, 1970, pp. 443–458.
- HUDELMAIER, J. [1993] *An $O(n \log n)$ -space decision procedure for intuitionistic propositional logic*, **J. Logic and Computation** 3 (1993), 63–75.
- HUET, G. [1975] *A unification algorithm for typed λ -calculus*, **Theoretical Computer Science** 1 (1975), 27–57.
- ISEKI, K., TANAKA, S. [1978] *An introduction to the theory of BCK-algebras*, **Mathematica Japonica** 23 (1978), 1–26.
- JASKOWSKI, S. [1963] *Über Tautologien, in welchen keine Variable mehr als zweimal vorkommt*, **Zeitschrift für Mathematische Logik** 9 (1963), 219–228.
- KALMAN, I. A. [1982] *The two-property and condensed detachment*, **Studia Logica** 41 (1982), 173–179.

- KALMAN, J. A. [1983] *Condensed detachment as a rule of inference*, **Studia Logica** 42 (1983), 443–451.
- KANELLAKIS, P., MAIRSON, H., MITCHELL, J. C. [1991] *Unification and ML-type reconstruction*, in **Computational Logic, Essays in Honour of Alan Robinson**, ed. J. -L. Lassez, G. Plotkin, M.I.T. Press, Cambridge, Mass., USA 1991, pp. 444–478.
- KANELLAKIS, P., MITCHELL, J. C. [1989] *Polymorphic unification and ML typing*, in **Proceedings of the 16th Annual A.C.M. Symposium on the Principles of Programming Languages (P.O.P.L.)**, 1989, Assoc. for Computing Machinery, New York, USA, 1989 pp. 105–113.
- KFOURY, A. J., TIURYN, J., URZYCZYN, P. [1990] *ML typability is DEXPTIME-complete*, **Lecture Notes in Computer Science** 431 (1990), Springer-Verlag, Germany, pp. 206–220.
- KNIGHT, K. [1989] *Unification: a multidisciplinary survey*, **A.C.M. Computing Surveys** 21 (1989), 93–124.
- KOMORI, Y. [1987] *BCK algebras and lambda calculus*, in **Proc. 10th Symposium on Semigroups, Sakado 1986**, publisher Josai University, Sakado, Saitama, Japan, 1987, pp. 5–11.
- KOMORI, Y. [1989] *Illative combinatory logic based on BCK-logic*, **Mathematica Japonica** 34 (1989), 585–596.
- KOMORI, Y., HIROKAWA, S. [1993] *The number of proofs for a BCK-formula*, **J. Symbolic Logic** 58 (1993), 626–628.
- KRIVINE, J. -L. [1990] *Lambda-calcul, types et modèles*, Masson, France, 1990. English transl. **Lambda-Calculus, Types and Models**, Ellis-Horwood, USA 1993.
- LAMBEK, J., SCOTT, P. J. [1986] *Introduction to Higher-order Categorical Logic*, Cambridge University Press, UK, 1986.
- LÄUCHLI, H. [1965] *Intuitionistic propositional calculus and definably non-empty terms* (abstract), **J. Symbolic Logic** 30 (1965), 263.
- LEIVANT, D. [1979] *Assumption classes in natural deduction*, **Zeitschrift für Mathematische Logik** 25 (1979), 1–4.
- LEMMON, E. J., MEREDITH, C. A., MEREDITH, D., PRIOR, A. N., THOMAS, I. [1957] *Calculus of pure strict implication*, mimeographed 1957, publ. in **Philosophical logic**, ed. Davis, Hockney, Wilson, D. Reidel Co., Netherlands 1969, pp. 215–250.
- LUKASIEWICZ, J. [1939] *Der Äquivalenzenkalkül*, **Collectanea Logica** 1 (1939), 145–169. English transl. as *The equivalential calculus*, in **Jan Lukasiewicz Selected Works**, ed. L. Borkowski, North-Holland Co., Netherlands 1970, pp. 250–277.
- MAIRSON, H. [1990] *Deciding ML typability is complete for deterministic exponential time*, in **Proceedings of the 17th Annual A.C.M. Symposium on the Principles of Programming Languages (P.O.P.L.)**, 1990, Assoc. for Computing Machinery, New York, USA 1990, pp. 382–401.
- MARTELLI, A., MONTANARI, U. [1982] *An efficient unification algorithm*, **A.C.M. Transactions on Programming Languages and Systems** 4 (1982), 258–282.
- MARTIN-LÖF, P. [1975] *An intuitionistic theory of types: predicative part*, in **Logic Colloquium '73**, ed. H. E. Rose et al., North-Holland Co., Netherlands 1975, pp. 73–118.
- MASLOV, S. [1964] *An inverse method of establishing deducibilities in the classical predicate calculus*, **Soviet Mathematics** 5 (1964), 1420–1424. (English transl. of **Doklady Akad. Nauk S.S.S.R.** 159 (1964), 17–20.)
- MEGILL, N. D., BUNDER, M. W. [1996] *Weaker D-complete logics*, **Journal of the Interest Group on Propositional Logics** 4 (1996), 215–225.
- MEREDITH, C. A., PRIOR, A. N. [1963] *Notes on the axiomatics of the propositional calculus*, **Notre Dame J. Formal Logic** 4 (1963), 171–187.
- MEREDITH, D. [1977] *In memoriam Carew Arthur Meredith*, **Notre Dame J. Formal Logic** 18 (1977), 513–516.
- MEYER, R. K., BUNDER, M. W. [1988] *Condensed detachment and combinators*, Tech. Report TR-ARP-8/88 (1988), Research School of the Social Sciences, Australian National University, Canberra, Australia.

- MILNER, R. [1978] *A theory of type polymorphism in programming*, **J. Computer and System Sciences** 17 (1978), 348–375.
- MINTS, G. [1979] *A primitive recursive bound of strong normalization for predicate calculus*, **Zapiski Nauchnyh Seminarov Leningradskogo Otdelenija Matematicheskogo Instituta im V. I. Steklova Akademii Nauk S.S.S.R. (L.O.M.I.)** 88 (1979), 131–135 (in Russian with English summary).
- MINTS, G., TAMMET, T. [1991] *Condensed detachment is complete for relevance logic: a computer-aided proof*, **J. Automated Reasoning** 7 (1991), 587–596.
- MITCHELL, J. C. [1988] *Polymorphic type inference and containment*, **Information and Computation** 76 (1988), 211–249.
- MITCHELL, J. C. [1990] *Type systems for programming languages*, in **Handbook of Theoretical Computer Science Vol. B, Formal Models and Semantics**, ed. J. van Leeuwen, Elsevier, Netherlands 1990, pp. 365–458.
- MITCHELL, J. C. [1996] **Foundations for Programming Languages**, M.I.T. Press, USA 1996.
- MITSCHKE, G. [1979] *The standardization theorem for the λ -calculus*, **Zeitschrift für Mathematische Logik** 25 (1979), 29–31.
- MOH, S.-K. [1950] *The deduction theorems and two new logical systems*, **Methodos** 2 (1950), 56–75.
- MORRIS, J. H. [1968] **Lambda-calculus Models of Programming Languages**, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Mass., USA 1968.
- NEDERPELT, R. P. [1973] **Strong Normalization in a Typed Lambda-calculus with Lambda-structured Types**, Ph.D. thesis, Tech. Hogeschool, Eindhoven, Netherlands 1973.
- NERODE, A., ODIFREDDI, P., [199-] **Lambda Calculi and Constructive Logics**, Cornell University Press, USA, to appear 1997.
- ODIFREDDI, P. [1990] (editor) **Logic and Computer Science**, Academic Press, USA 1990 (Series *APIC Studies in Data Processing* No. 31).
- PATERSON, M., WEGMAN, M. [1978] *Linear Unification*, **J. Computer and System Sciences** 16 (1978), 158–167.
- PLOTKIN, G. D. [1994] *A semantics for static type-inference*, **Information and Computation** 109 (1994), 256–299.
- PRAWITZ, D. [1960] *An improved proof procedure*, **Theoria** 26 (1960), 102–139.
- PRAWITZ, D. [1965] **Natural Deduction**, Almqvist and Wiksell, Sweden 1965.
- PRIOR, A. N. [1955] **Formal Logic**, Clarendon Press, UK 1955 (2nd. edition. 1962).
- RÉVÉSZ, G. [1988] **Lambda Calculus, Combinators and Functional Programming**, Cambridge University Press, UK 1988.
- REYNOLDS, J. [1985] *Three approaches to type structure*, in **Mathematical Foundations of Software Development, Vol. 1 (CAAP '85)**, ed. by H. Ehrig et al., **Lecture Notes in Computer Science**, Springer-Verlag, Germany, No. 185 (1985), 97–138.
- REZUS, A. [1981] **Lambda-conversion and logic**, Ph.D. thesis, Universiteit Utrecht, Math. Inst., Budapestlaan 6, de Uithof, Utrecht, Netherlands.
- ROBINSON, J. A. [1965] *A machine-oriented logic based on the resolution principle*, **J. Assoc. for Computing Machinery** 12 (1965), 23–41.
- ROBINSON, J. A. [1966] Review of Prawitz 1960, **J. Symbolic Logic** 31 (1966), 126–127.
- ROBINSON, J. A. [1979] **Logic: Form and Function. The Mechanization of Deductive Reasoning**, Edinburgh University Press, UK 1979.
- RUSSELL, B. [1903] **Principles of Mathematics**, Allen and Unwin, UK 1903. (1992 edition: Routledge and Kegan Paul, UK.)
- SALLÉ, P. [1978] *Une extension de la théorie des types*, in **Automata, Languages and Programming**, ed. by G. Ausiello, C. Böhm, **Lecture Notes in Computer Science**, Springer-Verlag, Germany, No. 62 (1978), 398–410.
- SANCHIS, L. E. [1967] *Functionals defined by recursion*, **Notre Dame J. Formal Logic** 8 (1967), 161–174.

- SCEDROV, A. [1990] *A guide to polymorphic types*, in Odifreddi 1990, pp. 387–420.
- SCHWICHTENBERG, H. [1991] *An upper bound for reduction sequences in typed lambda-calculus*, *Archiv für Math. Logik* 30 (1991), 405–408.
- SELDIN, J. P. [1968] *Studies in Illative Combinatory Logic*, Ph.D. thesis, University of Amsterdam, Netherlands 1968.
- SELDIN, J. P. [1977] *A sequent calculus for type assignment*, *J. Symbolic Logic* 42 (1977), 11–28.
- SELDIN, J. P. [1978] *A sequent calculus formulation of type assignment with equality rules for the $\lambda\beta$ -calculus*, *J. Symbolic Logic* 43 (1978), 643–649.
- STATMAN, R. [1979a] *Intuitionistic propositional logic is polynomial-space complete*, *Theoretical Computer Science* 9 (1979), 67–72.
- STATMAN, R. [1979b] *The typed λ -calculus is not elementary recursive*, *Theoretical Computer Science* 9 (1979), 73–81.
- STATMAN, R. [1980] *On the existence of closed terms in the typed λ -calculus*; Part 1 in To H.B. Curry, ed. J.R. Hindley and J.P. Seldin, Academic Press, UK 1980, pp. 511–534; Part 2 in *Theoretical Computer Science* 15 (1981), 329–338.
- STOUGHTON, A. [1988] *Substitution revisited*, *Theoretical Computer Science* 59 (1988), 317–325.
- TAIT, W. W. [1965] *Infinitely long terms of transfinite type*, in *Formal Systems and Recursive Functions*, ed. J. N. Crossley, M. A. Dummett, North-Holland Co., Netherlands 1965, pp. 176–185.
- TAIT, W. W. [1967] *Intensional interpretations of functionals of finite type*, *J. Symbolic Logic* 32 (1967), 198–212.
- TAKAHASHI, M. H. [1991] *Theory of Computation, Computability and Lambda-calculus*, Kindai Kagaku Sha, Tokyo, Japan 1991 (in Japanese).
- TAKAHASHI, M. H., AKAMA, Y., HIROKAWA, S. [1994] *Normal proofs and their grammar*, in *Theoretical Aspects of Computer Software (TACS '94)*, ed. by M. Hagiya, J. C. Mitchell, *Lecture Notes in Computer Science*, Springer-Verlag, Germany, No. 789 (1994), 465–493.
- TIURYN, J. [1990] *Type inference problems: a survey*, in *Mathematical Foundations of Computer Science 1990*, ed. by B. Rovan, *Lecture Notes in Computer Science*, Springer-Verlag, Germany, No. 452 (1990), 105–120.
- TROELSTRA, A. S. [1973] (editor) *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, *Lecture Notes in Mathematics*, Springer-Verlag, Germany, No. 344 (1973).
- TROELSTRA, A. S., van DALEN, D. [1988] *Constructivism in Mathematics, an Introduction*, Vols. 1 & 2, North-Holland Co., Netherlands 1988.
- TURING, A. M. [1942] Notes published in Gandy 1980a.
- TYSZKIEWICZ, J. [1988] *Complexity of Type Inference in Finitely Typed Lambda Calculus*, Master's thesis, University of Warsaw, Poland 1988.
- URQUHART, A. [1984] *The undecidability of entailment and relevant implication*, *J. Symbolic Logic* 49 (1984), 1059–1073.
- URQUHART, A. [1995] *The complexity of propositional proofs*, *Bull. Symbolic Logic* 1 (1995), 425–467.
- DE VRIJER, R. [1987] *Exactly estimating functionals and strong normalization*, *Proc. Koninklijke Nederlandse Akademie van Wetenschappen*, Series A, 90 (1987), 479–493.
- WAND, M. [1987] *A simple algorithm and proof for type inference*, *Fundamenta Informaticae* 10 (1987), 115–122.
- ZAIONC, M. [1985] *The set of unifiers in typed λ -calculus as regular expression*, in *Rewriting Techniques and Applications*, ed. by J.-P. Jouannaud, *Lecture Notes in Computer Science*, Springer-Verlag, Germany, No. 202 (1985), pp. 430–440.
- ZAIONC, M. [1987a] *The regular expression descriptions of unifier set in the typed λ -calculus*, *Fundamenta Informaticae* 10 (1987), 309–322.

- ZAIONC, M. [1987b] *Word operations definable in the typed λ -calculus*, **Theoretical Computer Science** 52 (1987), 1–14.
- ZAIONC, M. [1988] *Mechanical procedure for proof construction via closed terms in typed λ -calculus*, **J. Automated Reasoning** 4 (1988), 173–190.
- ZAIONC, M. [1990] *A characterization of lambda definable tree operations*, **Information and Computation** 89 (1990), 35–46.
- ZAIONC, M. [199-] *Fixpoint technique for counting terms in typed λ -calculus*, **Fundamenta Informaticae** (to appear).
- ZUCKER, J. [1974] *The correspondence between cut-elimination and normalization*, **Annals of Mathematical Logic** 7 (1974): *Part I*, pp. 1–112; *Part II*, pp. 113–155.

Table of principal types

This table gives a (not necessarily unique) principal inhabitant of each of some types. It gives a β -nf where possible, but if there is no normal principal inhabitant it gives a non-normal one. It includes most of the PT information in Tables 3E2a and 8B7a.

Type	A principal inhabitant
$a \rightarrow a$	$\lambda x. x \equiv \mathbf{I}$
$a \rightarrow b \rightarrow a$	$\lambda xy. x \equiv \mathbf{K}$
$a \rightarrow a \rightarrow a$	$(\lambda xyz. \mathbf{K}(xy)(xz))\mathbf{I}$ [Nprinc = \emptyset]
$(a \rightarrow b) \rightarrow a \rightarrow b$	$\lambda xy. xy$
$(a \rightarrow b) \rightarrow a \rightarrow a$	$\lambda xy. \mathbf{K}y(xy)$ [Nprinc = \emptyset]
$(a \rightarrow a) \rightarrow a \rightarrow a$	$\lambda xy. x(xy)$ [and $\lambda xy. x^n y$, $n \geq 2$]
$a \rightarrow (a \rightarrow b) \rightarrow a$	$\lambda xy. \mathbf{K}x(yx)$ [Nprinc = \emptyset]
$a \rightarrow a \rightarrow b \rightarrow b$	$(\lambda xyz. \mathbf{K}(xy)(xz))(\mathbf{K}\mathbf{I})$ [Nprinc = \emptyset]
$((a \rightarrow a) \rightarrow a) \rightarrow a$	$\lambda x. x(\lambda y. x\mathbf{I})$ [for others see 8B7a(12)]
$(a \rightarrow a \rightarrow b) \rightarrow a \rightarrow b$	$\lambda xy. xyy \equiv \mathbf{W}$
$(a \rightarrow a \rightarrow a) \rightarrow a \rightarrow a$	$\lambda xy. x(xyy)y$
$(a \rightarrow b) \rightarrow a \rightarrow a \rightarrow b$	$\lambda xyz. \mathbf{K}(xy)(xz)$ [Nprinc = \emptyset]
$(a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c$	$\lambda xyz. xzy \equiv \mathbf{C}$
$(a_1 \rightarrow \dots \rightarrow a_n \rightarrow b \rightarrow c) \rightarrow$ $b \rightarrow a_1 \rightarrow \dots \rightarrow a_n \rightarrow c$	$\lambda uvw_1 \dots w_n. uw_1 \dots w_n v$
$(a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b$	$\lambda xyz. x(yz) \equiv \mathbf{B}$
$(a \rightarrow b) \rightarrow (b \rightarrow c) \rightarrow a \rightarrow c$	$\lambda xyz. y(xz) \equiv \mathbf{B}'$
$(a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$	$\lambda xyz. xz(yz) \equiv \mathbf{S}$
$(a \rightarrow b \rightarrow b \rightarrow c) \rightarrow a \rightarrow b \rightarrow c$	$\lambda xyz. xyz$
$(a \rightarrow b) \rightarrow (c \rightarrow d \rightarrow a) \rightarrow c \rightarrow d \rightarrow b$	$\lambda uvwx. u(vwx)$
$(a \rightarrow b) \rightarrow (c \rightarrow b \rightarrow d) \rightarrow c \rightarrow a \rightarrow d$	$\lambda uvwx. vw(ux)$
$(a \rightarrow b) \rightarrow (c_1 \rightarrow \dots \rightarrow c_n \rightarrow b \rightarrow d) \rightarrow$ $c_1 \rightarrow \dots \rightarrow c_n \rightarrow a \rightarrow d$	$\lambda uvw_1 \dots w_n x. vw_1 \dots w_n(ux)$
$((a \rightarrow b) \rightarrow c \rightarrow a) \rightarrow (a \rightarrow b) \rightarrow c \rightarrow b$	$\lambda uvw. v(uvw)$
$(a \rightarrow a') \rightarrow (b \rightarrow b') \rightarrow (a' \rightarrow b) \rightarrow a \rightarrow b'$	$\lambda uvwx. v(w(ux))$
$(a \rightarrow b) \rightarrow (c \rightarrow d) \rightarrow c \rightarrow (d \rightarrow a) \rightarrow b$	$\lambda uvwx. u(x(vw))$
$(a \rightarrow b \rightarrow c) \rightarrow a \rightarrow (d \rightarrow b) \rightarrow d \rightarrow c$	$\lambda uvwx. uv(wx)$

Table of principal types – continued

Type	A principal inhabitant
$(a \rightarrow b \rightarrow c) \rightarrow a \rightarrow (c \rightarrow d) \rightarrow b \rightarrow d$	$\lambda uvwx \cdot w(ux)$
$(a \rightarrow b \rightarrow c) \rightarrow (d \rightarrow a) \rightarrow b \rightarrow d \rightarrow c$	$\lambda uvwx \cdot u(vx)w$
$((a \rightarrow b) \rightarrow a \rightarrow c) \rightarrow d \rightarrow (b \rightarrow c) \rightarrow d$	$\lambda uv \cdot u(\lambda xy \cdot v(xy))$
$((a \rightarrow b) \rightarrow c \rightarrow b) \rightarrow d \rightarrow (c \rightarrow a) \rightarrow d$	$\lambda uv \cdot u(\lambda xy \cdot x(vy))$
$(a \rightarrow b \rightarrow c) \rightarrow (d \rightarrow a) \rightarrow (d \rightarrow b) \rightarrow d \rightarrow c$	$\lambda uvwx \cdot u(vx)(wx)$
$(a \rightarrow a') \rightarrow (b \rightarrow c \rightarrow c') \rightarrow b \rightarrow (c' \rightarrow a) \rightarrow c \rightarrow a'$	$\lambda uvwxy \cdot u(x(vwy))$
$(a \rightarrow a') \rightarrow (b \rightarrow c \rightarrow c') \rightarrow b \rightarrow (a' \rightarrow c) \rightarrow a \rightarrow c'$	$\lambda uvwxy \cdot vw(x(uy))$
$(a \rightarrow c \rightarrow c') \rightarrow (b \rightarrow c' \rightarrow c'') \rightarrow a \rightarrow b \rightarrow c \rightarrow c''$	$\lambda uvwxy \cdot vx(uwy)$
$a_1 \rightarrow \dots \rightarrow a_n \rightarrow (a_1 \rightarrow \dots \rightarrow a_n \rightarrow b) \rightarrow b$	$\lambda u_1 \dots u_n v \cdot vu_1 \dots u_n$
$(a \rightarrow b_1 \rightarrow \dots \rightarrow b_n \rightarrow c \rightarrow d) \rightarrow c \rightarrow b_1 \dots b_n \rightarrow a \rightarrow d$	$\lambda uvw_1 \dots w_n x \cdot uxw_1 \dots w_nv$
$(a \rightarrow b) \rightarrow ((c_1 \rightarrow \dots \rightarrow c_n \rightarrow a \rightarrow d) \rightarrow e) \rightarrow$ $(c_1 \rightarrow \dots \rightarrow c_n \rightarrow b \rightarrow d) \rightarrow e$	$\lambda uvw \cdot v(\lambda x_1 \dots x_n y \cdot wx_1 \dots x_n(uy))$
$(a \rightarrow b \rightarrow c) \rightarrow ((d_1 \rightarrow \dots \rightarrow d_n \rightarrow a \rightarrow e) \rightarrow f) \rightarrow$ $b \rightarrow (d_1 \rightarrow \dots \rightarrow d_n \rightarrow c \rightarrow e) \rightarrow f$	$\lambda uvwx \cdot v(\lambda y_1 \dots y_n z \cdot xy_1 \dots y_n(uzw))$

Index

- \mathbb{A} -logics (see axiom-based logics)
- $\mathbb{A}^{D\vdash}$ 104
- \mathbb{A}^{\vdash} 89
- $\mathcal{A}(\tau, d)$ 120
- $\mathcal{A}(\tau, \leq d)$ 124
- $\mathcal{A}_{\text{terms}}(\tau, d)$ 120
- $\mathcal{A}_{\text{terms}}(\tau, \leq d)$ 124
- abstract (λ -abstract) 1
- abstracted atom 7
- abstractor 2, 141
 - covering 2, 130
 - initial 110, 130
- abstractor position 140
- algorithms
 - for counting inhabitants 108–139, esp. 125–6
 - Search algo. in 120–2
 - main searching strategy 114–7
 - Converse PT algo. 96–101, esp. 101
 - Principal-type (PT) algo. 30–51, esp. 44–9
 - historical comment on 33–4
 - main step in 38
 - Type-checking algo. (= PT algo.)
 - Unification algo. 42–3
- alphabetic variant 37
- application (term) 1
- application in a λ -model 58
- applicative combination 157
- argument in a normal form 110, 128
 - subargument 129
- argument-branch 130
- argument-part of a redex 144
- argument position 140
- arity of a type 115
- assignment of types (see TA_{λ})
- association rule for parentheses
 - in terms 1
 - in types 12
- atom, in a term 1
 - abstracted 7
 - typed 67
- atom, in a type 12
- atomic deduction 78, 148
- atomic term (see atom in a term)
- axiom-based logics 88–92
 - \mathbb{A} -logics, definition 88–9
 - BCI-logic (also see BCI-) 90
 - D-incompleteness of 106–7
 - BCK-logic (also see BCK-) 90
 - D-incompleteness of 106–7
 - condensed logics 104
 - Intuitionist logic (also see Intuit.) 90
 - D-completeness of 106
 - link with Natural Deduction 92
 - link with types of λ -terms 91
 - R_{\rightarrow} (also see R_{\rightarrow}) 90
 - D-completeness of 105
 - T_{\rightarrow} (logic of ticket entailment) 106
- axioms
 - of TA_{λ} 15
 - (B) (B'), (C), (I), (K), (W) 81, 90
- B** (closed λ -term) 4
 - principal type of 16, 33, 50
 - deduction for PT of 16
- (B) (axiom) 81, 90
- B'** (closed λ -term) 4
 - principal type of 18, 50
 - deduction for PT of 161
- (B') (axiom) 81
- basic combinators, sets \mathbb{S} of 157–8
- basis, typable 159
- BCI-logic
 - Natural Deduction version 87
 - axiom-based version 90
 - D-incompleteness of 106–7
 - link with types of λ -terms 88
- BCIW**-PT 94
- $\text{BCI}\lambda$ -terms 11
- $\text{BCI}\lambda$ -PT 94
- BCK-logic
 - Natural Deduction version 86
 - axiom-based version 90
 - D-incompleteness of 106–7
 - link with types of λ -terms 88
- BCKW**-PT 94
- $\text{BCK}\lambda$ -terms 10
 - typability of all 28
- $\text{BCK}\lambda$ -PT 94
- binding occurrence 2, 141
- body of an abstract 2
- body position 140
- bound variable-occurrence 2
 - changing (see α -conversion)
- bound-variable clash 3
 - in typed terms 68

- C** (closed λ -term) 4
 - principal type of 18, 50
 - deduction for 161
- (C) (axiom) 81, 90
- $CA()$ 130
- cancelled assumptions (see discharging)
- cancelling reduction 11
- cardinality of a set of terms 109
- Changing bound variables (see α -conversion)
- Church numerals 4, 118
- Church-Rosser theorem
 - for β 5
 - for $\beta\eta$ and η 8
 - for typed terms 70
- Church-style type-theories 14
- c.i. (= common instance) 38–9
- clash of bound variables 3
 - in typed terms 68
- Classical logic, D-completeness of 106
- closed nf-scheme 119
- closed term 4
- combination of given terms 157
- combinator (= closed λ -term) 4
 - B**, **B'**, **C**, **I**, **K**, **S**, **W**, **Y** 4, 50
 - fixed-point (see **Y**)
- combinatory logic (CL)
 - imitated in λ -calculus 157–9
 - combinatory completeness 158, 159
 - typable basis for 159
 - type-system TA_C for CL 19
- common instance 38–9
- Commuting lemma for β and η 8
- complete development 146
- complete discharge convention 77
- completeness
 - combinatory 158–9
 - comp. lemma for Search algo. 132
 - of rule (D) (see D-completeness)
- Completeness theorem (semantic) 59
- component of a substitution 35
- component of a term 2, 141
 - contains another component 142
 - does not overlap another 143
 - is a part of another 142
 - is a proper part of another 143
 - is disjoint from another 143
 - is in another 142
 - is isomorphic to another 143
 - proper 141
 - properly contains another 143
- component of a type 151
 - s-component 155
- composite term 1
- composite type 12
- composition of substitutions (\circ) 36
- Composition-extension lemma 36
- $Con()$ (minimum context) 65
- concatenation of positions 140
- conclusion of a type (see tail)
- conclusion (= tail) position 153
- Condensed Detachment (see (D), rule)
- condensed logics 104
 - link with principal types 105
- condensed tree of a type 153–4
- consistent (for contexts) 14, 15
- construction-tree
 - of a term 141–2
 - of a type 152
 - condensed version 153–4
- contains (said of components) 142
- context, type- 14
 - for nf-schemes 119–20
 - M-context 15
 - minimum context, $Con()$ 65
- context-free grammars 123, 124
- contraction (also see β -, $\beta\eta$ -, η -)
 - β -contracts 4
 - for typed terms 69
 - $\beta\eta$ -contracts 8
 - η -contracts 7
 - for typed terms 69
- contraction (kind of substitution) 96
- contractum
 - of a β -redex 4, 69
 - of an η -redex 7, 69
- Converse principal-type algorithm 96–101, esp. 101
- Converse principal-type theorem
 - for **BB'** **IW**-combinations 102
 - for λI 95
 - proof of 96–101
 - for λK 94
- conversion
 - α - (see α -conversion)
 - β - (also see β -) 5
 - $\beta\eta$ - (also see $\beta\eta$ -) 8
 - η - (also see η -) 7
- conversion-sensitivity of $Types()$ 26
- corresponding logic deduction Δ_L 79
- counting (contrast to enumerating) 109
- Counting theorem and algorithm
 - for $Long(\tau)$ 125
 - for $Nhabs(\tau)$ 126
 - for $Nhabs_\eta(\tau)$ 125
 - Search algo. 120–2
 - main search strategy 114–7
- covering abstractor 2
 - $CA()$ 130
- Curry-Howard mapping 79–85, esp. 79, cf. also 74
 - is an isomorphism of proofs 83
 - is not 1-1 for deductions 81
 - one-sided inverse of 82, 84
- Curry-Howard theorem 84
 - for axiom-based logics 91
 - for condensed logics 105
 - for weaker logics 88
- Curry-style type-theory (also see TA_λ) 14
- cut-elimination, to prove normalisn. 72
- (D) (condensed detachment rule) 102–7, esp. 102
 - historical note on 103–4
- $D(\rho \rightarrow \sigma) \tau$ 102–3
- $D()$ 124
- D-completeness 104
 - of intuitionist logic 106
 - of R_\perp 105
 - of some other logics 106
 - preserved by D-extension 105
 - fails for BCI, BCK logics 106–7
- D-deducible, D-deduction 104
- D-extension of a logic 105

- D-incompleteness of BCI, BCK logics 106
- D-proof 104
- D-theorem 104
- Deduction theorem mentioned 92
- deduction-rules (see rules)
- deductions
 - in a Natural Deduction logic 78
 - in axiom-based logics 89
 - in TA_λ (see TA_λ -deduction)
 - substitutions-first 90
- depth of a β -nf 110
- depth of a subargument in a nf 129
- $Depth()$ (see depth of a β -nf)
- detachment rules (see $(\rightarrow E)$, (D))
- development 146
- disagreement pair 43
- discharge convention, complete 77
- discharge convention, partial 77
- discharge label 76, 78
- discharge-equivalent 79
- discharging 16, 76–8
 - vacuous 16, 76–8
- disjoint components 143
- diverging positions 140
- $Dom()$ 35
- duplicating reduction 11
- Enhanced Subformula lemma 131
- enumerating (contrast to counting) 109
- environment, semantic 57
- (Eq_β) , $(Eq_{\beta\eta})$, rules 52
 - (Eq) denotes both 52
- equal (see β -, $\beta\eta$ -, η -conversion)
- Equality-postponement theorem 53
- erasing types 65
- expansion, β - 5
 - Subject-expansion theorem 24
- extendable nf-scheme 122
- extensions of a logic 105
- extensions of a nf-scheme 122
- extensional λ -model 58
- extensionally equivalent substitutions 35
- fixed-point combinator (see Υ)
- formula, implicational 75
- formula, TA_λ - 15
- formulae-as-types (also see Curry-Howard) 74
- free variable-occurrence 2, 68
- function position 140
- function-part of a redex 144
- $FV()$ 2, 68
- $Habs()$, $Habs_r()$, $Habs_u()$ 108
- head of a normal form 110, 128
- Hilbert-Gentzen link theorem 92
- Hilbert-style (see axiom-based logics)
- HS86 (refers to Hindley & Seldin 1986 in Bibliography)
- I (closed λ -term) 4
 - principal type of 17, 50
- (I) (axiom) 81, 90
- $IA()$ 130
- $IAT()$ 130–1
- IAT -intervals 136
- identical modulo renaming 37
- identification (kind of substitution) 96
- Identification lemma 100
- implicational formulae 75
- implicational logics (see logics)
- in (said of components) 142
- inhabitant 108–39, esp. 108–9
 - normal 108–9
 - long 110
 - principal 112
- Inhabitant-counting algorithm
 - for $Long(\tau)$ 125
 - for $Nhabs(\tau)$ 126
 - for $Nhabs_\eta(\tau)$ 125
 - Search algo. in 120–2
 - main search strategy in 114–7
- inhabitants of particular types (tables of) 118, 177–8
- inhabited type 109
- initial abstractors 110, 130
- instance (= substitution-inst.) 31
 - common 38–9
- interpretations in a λ -model
 - of terms 57–8
 - of types 58
- intersection-types 18
- Intuitionist logic (implicational)
 - axiom-based version 90
 - D-completeness of 106
 - Natural Deduction version 76
 - consistency of 117
 - decidability of 85
 - link with types 84
- inverse of a substitution 37
- isomorphic occurrences 115, 143
- K (closed λ -term) 4
 - principal type of 17, 50
 - deduction for 17
- (K) (axiom) 81, 90
- lambda-to-logic (see Curry-Howard)
- leftmost β -reduction 6
- length
 - of a deduction 149
 - of a position 140
 - of a reduction 5
 - of a term 2
 - a typed term 67
 - of a type 13
 - of an argument-branch 130
 - of $CA()$, $IA()$ 130
 - of $IAT()$ 131
 - 2-length of a position 129
- linear logic, implicational fragment 87
- linear λ -terms 11
- listing (= enumerating) 109
- logic deductions (see deductions)
- logic-to-lambda mapping 82
- logics, implicational 74–92, 102–7
 - (also see BCI-, BCK-, Classical, Intuitionist, R_\rightarrow , T_\rightarrow)
 - axiom-based (= A-logics) 88–9
 - Natural Deduction based 76–9, 86–7
 - condensed logics 104
 - link with principal types 105
 - extensions, D-extensions 105

- long β -normal form 110
- Long*() 110
 - counting theorem and algo. for 125
 - search algorithm for 120–2
 - in restricted classes of terms 127
- Long*(, *d*) 111
- Long _{η}* () 113
- m.g.c.i. (= most general common instance) 39
- m.g.c.i.-generator 39
- m.g.u. (= most general unification) 41
- m.g.u. (= most general unifier) 41
- major premise 89
- maximal-length reduction 5
- maximal (= long) normal form 110
- MCD 146
- metavariables 119
- minimal complete development (see MCD)
- minimal logic (see Intuitionist logic)
- minimal member 146
- minimal tail-repetition 136
- minimum context 65
- minor premise 89
- ML (programming language) *x*, 30, 52
- model, λ - 57
 - extensional 58
- modified basic functionality 19
- Modus Ponens (also see $\rightarrow E$) 89
- monovalent (= consistent) 14
- most general common instance (see m.g.c.i.)
- most general unification 41
- most general unifier 41
- Natural Deduction
 - in implicational logics 76–9, 86–7
 - in TA_λ 15–16, 18
- negative occurrence 152
- negative s-component 156
- negative subpremise-sequence 156
- new metavariable or variable 119
- newly-created redex-occurrence 147
- nf (= normal form, see β -nf, $\beta\eta$ -nf, η -nf)
- nf-scheme 119
 - closed 119
 - long typed 120
 - proper 119
 - typed 119
- Nhabs*(), *Nhabs _{η}* (), *Nhabs_u*() 108
 - counting theorem and algo. for 126
 - searching for members 114–22
- Nhabs _{η}* () 113
 - counting 125
- non-cancelling reduction 11
- non-duplicating reduction 11
- non-overlapping components 143
- normal form (see β -, $\beta\eta$ - η -nf)
- normal inhabitant (see *Nhabs*())
- normalisation, strong (see SN)
- normalisation, weak (see WN)
- Nprinc*() 112, 127
- Nprinc _{η}* () 113, 127
- NSS*() 156
- numerals of Church 4
 - principal types of 50, 118
- \bar{n} (see numerals of Church)
- occurrences in a term 2, 141
 - abstractors 2, 141
 - binding occurrences 2, 141
 - occurrences of λx 141
- occurrences in a type 151
 - negative, positive 152, 156
- occurs (in a term) 2, 141
- occurs (in a type) 151
 - negatively, positively 152, 156
- Order of a type, *Order*() 156
- overlaps, does not overlap 143
- parent of a residual 146
- parentheses, omission from terms 1
- parentheses, omission from types 12
- part of (said of components) 142
- Partial completeness theorem 159
- partial discharge convention, partial discharging 76–7
- Peirce's Law (PL) 75
 - intuitionistic unprovability 85, 117
- (PL) (see Peirce's Law)
- positions 140, 153
 - abstractor pos., argument pos., body pos., function pos. 140
 - concatenation of two pos. 140
 - conclusion or tail pos., premise pos. 153
 - refinement of a pos. 140
 - diverging pair of pos. 140
- positive occurrence 152
- positive s-component 156
- Postponement theorem for η 8
- predicate 14, 15
- premise position 153
- premises of a type 115, 155
 - subpremises 155
- Princ*() 112, 127
- principal deduction 32
- principal inhabitants of a type 112
- principal pair 32
- principal type (see PT)
- proofs (see also deductions)
 - in a Natural Deduction logic 79
 - in an axiom-based logic 89
 - in TA_λ 16
- proper component of a term 141
- proper nf-scheme 119
- proper part of a component 143
- proper refinement of a position 140
- proper s-subtype 154
- properly contains 143
- propositional variables 75
- provability problem 89
- provable formula (= theorem) 79, 89
- PT (= principal type)
 - definition in TA_λ 32
 - of **B**, **I**, **K** (with deductions) 16–17
 - of **B'**, **C**, **S**, **W** 18, 161
 - tables of PT's 50, 177–8
 - in $TA_{\lambda+\beta}$ and $TA_{\lambda+\beta\eta}$ 56
 - BCL** λ -PT, **BCKW**-PT, etc. 94
 - λI -PT, λK -PT as is 94
- PT*() (= principal type of) 32
- PT algorithm 30–51, esp. 44–9
 - main step in 38
 - historical note 33–4

PT algorithm, converse (see Converse principal-type algorithm)
 PT theorem
 for TA_λ 33
 for $TA_{\lambda+\beta}$ and $TA_{\lambda+\beta\eta}$ 56
 Pumping lemma 124
 pure λ -calculus 1

R_\perp (relevance logic)
 axiom-based version 90
 D-completeness of 105
 Natural Deduction version 86
 link with types of λI -terms 88

$Range()$ 35

redex
 β - (also see β -redex) 4
 $\beta\eta$ - 8
 η - 7
 typed β -, η - 69

redex-occurrences, notation for 144

reduces, reductions
 β - (also see β -reduction) 4
 $\beta\eta$ - 8
 η - (also see η -reduction) 7
 length of a reduction 5

reduction of a deduction 70

reduction of subjects 24

refinement of a position 140
 proper refinement 140

regular languages 124

reject, in Search algorithm 122

relevance logic (see R_\perp)

replacement in a term 143
 typed 67
 in a deduction 149
 replacement lemmas 150–1

replacements, suitable 121

residuals of a redex-occurrence 145
 of a set 146

rules of deduction
 (D), condensed detachment 102–3
 historical note 103–4
 (Eq), (Eq β), (Eq $\beta\eta$) 52
 (Sub) in propositional logics 89
 (\rightarrow E) in propositional logics 76, 89
 (\rightarrow E) in TA_λ 16
 (\rightarrow I) in TA_λ 16
 (\rightarrow I)_{main}, (\rightarrow I)_{vac} 16

S (closed λ -term) 4
 principal type of 18
 deduction for 161

\mathbb{S} -and-variables combination 157

\mathbb{S} -combination 157

s-component 155
 negative, positive 156

s-subtype 154

satisfaction in a λ -model 58

scope 2

Search algorithm 120–2
 main strategy of 114–7

Search-completeness lemma 132

self-application 17

semantic environment 57

semantics, simple 59

Shrinking lemma, statement 124
 proof 135–8

significant subtype (see s-subtype)

simple semantics 59

simple type theory (see also TA_λ) ix

simultaneous substitution
 in terms 3
 in types 31, 35

single substitution in types 35

skeletal type 96

skeleton of a type 96

SN (strong normalization) 27–8

SN theorem
 for typable terms in TA_λ 28
 for typed terms 72
 historical note 73

Standardization theorem mentioned 7

stratified term (= typable term, q.v.)

Stretching lemma, statement 124
 proof 134–5

strong normalization (see SN)

strong type-invariance 158

structure of β -normal form 7
 typed 110
 long typed 115

(Sub), substitution rule 89

subarguments 129

Subformula lemma, enhanced 131

subformula property 22

subject 14, 15

Subject-construction theorem 20–1

Subject-expansion theorem 24

Subject-reduction theorem
 for β -reduction 24
 for η -reduction in $TA_{\lambda+\beta}$ 56

subpremises, *Subpremises()* 155

substitution-instance (see instance)

substitution into terms
 def. for untyped terms 3
 simultaneous 3
 def. for typed terms 68
 subs. lemmas for deductions 23

substitution rule (Sub) 89

substitutions-first deduction 90

substitutions into types 31, 35
 components of a sub. 35
 composition of two subs. (\circ) 36
 contractions (= identifications, q.v.)
 extensional equivalence ($=_{ext}$) 35
 identifications 96
 inverse of a one-to-one sub. 37
 one-to-one sub. 37
 renamings 37
 single sub. 35
 trivial comps. of a sub. 35
 union of two subs. (\cup) 35
 variable-domain (*Dom()*) 35
 variable-range (*Range()*) 35

subtails, *Subtails()* 155

subterm 2, 141

subtype 151
 s-subtype 154

success made in Search algo. 122

suitable multi-replacements 122

suitable replacements 121

- T_{\rightarrow} , ticket entailment logic 106
- TA_C 19–20
- TA_{λ} 15–16
 - axioms and rules for 15–16
 - basic facts about 12–29
 - comparison with HS86 version 18
- TA_{λ} -
 - deducible formula 16
 - deduction 16
 - detailed definition 148–9
 - length of 149
 - translated to a typed term 66
 - formula 15
 - typable term 27
- $TA_{\lambda+\beta}$, $TA_{\lambda+\beta\eta}$ 52
- $TA_{\lambda+\beta[\eta]}$ (for both $TA_{\lambda+\beta}$, $TA_{\lambda+\beta\eta}$) 52
- $TA_{\lambda+\beta}$ -typable, $TA_{\lambda+\beta\eta}$ -typable 52
- tail of a type, $Tail()$ 115, 155
 - subtail 155
- tail position 153
- term (means λ -term) 1
 - abstract 1
 - application 1
 - closed 4
 - composite 1
 - construction-tree of 141–2
 - subterm 2
 - typable (see typable term)
 - typed (see typed term)
- term models 59
- term-variable 1
- terminates, said of a reduction 5
- theorem (= provable formula)
 - in a Natural Deduction logic 79
 - in an axiom-based logic 89
- ticket entailment logic (T_{\rightarrow}) 106
- TM_{β} , $TM_{\beta\eta}$ (term models) 59
- $TM_{\beta[\eta]}$ (for both TM_{β} , $TM_{\beta\eta}$) 59
- $TNS(\Gamma)$ 119–20
- trivial component of a substitution 35
- $TT(\Gamma)$ 64
- typable basis 159
- typable term
 - in TA_{λ} 27
 - in $TA_{\lambda+\beta}$, $TA_{\lambda+\beta\eta}$ 52
- type-assignment (expression) 14
- type-assignment theory (see TA_{λ})
- Type-checking algorithm (see PT algo.)
- type-context 14
 - for nf-schemes 119
 - minimum, $Con()$ 65
- type-erasure 65
- type-invariance, strong 158
- type-substitutions (see subs. into types)
- type-theories in general x
 - Church-style and Curry-style 13–14
- type-variables 12
 - interpretation in a λ -model 58
- typed
 - nf-scheme 120
 - substitution 68
 - term 63–73, esp. 64
 - encoding a TA_{λ} -deduction 66
 - variable 64
 - α -conversion 69
 - β -, η -redex, reduction 69
 - β -, η -, $\beta\eta$ -normal form 69
 - structure of β -nf 110
- types 12
 - atomic 12
 - composite 12
 - construction-tree of 152
 - condensed 153–4
 - inhabitants, tables of 50, 177–8
 - principal (also see PT) 32
 - with intersection 18
- $Types()$ 26
- $Types_{sno\ eq}()$ 56
- $Types_{\beta}()$, $Types_{\beta\eta}()$ 55
- unextendable argument-branch 130
- unifiable 40
- unification 40
 - most general (= m.g.u.) 41
- Unification algorithm 42–3
 - historical note on 43
 - use in PT algorithm 34, 45–6
- Unification theorem 42
- unifier 40
 - most general (= m.g.u.) 41
- union of substitutions (\cup) 35
- uniqueness of TA_{λ} -deductions
 - holds for nf's and λI -terms 22
- vacuous discharging 16, 76, 78
- valid in a λ -model 58
- variable
 - bound 2, 68
 - changing (see α -conversion)
 - free 2, 68
 - new 119
 - propositional 75
 - term- 1
 - type- 12
 - interpreting in a λ -model 58
 - typed 64
- variable-domain of a substitution 35
- variable-range of a substitution 35
- $Vars()$ 13, 31, 149
- W** (combinator) 4
 - principal type of 18
 - deduction for 161
- (W) (axiom) 81, 90
- weak normalization (see WN)
- Weakening lemma 18
- WN (= weak normalization) 27–8
- WN theorem
 - for TA_{λ} -typable terms 28
 - for $TA_{\lambda+\beta}$ and $TA_{\lambda+\beta\eta}$ 55
 - for typed terms 71
 - historical note 72
- Y** (fixed-point combinator) 4
 - untypability of 28
- α -conversion 3
 - of typed terms 69
- α -invariance lemma for TA_{λ} 23

- β -contraction 4
 - of typed terms 69
- β -conversion 5
 - Church-Rosser theorem for 5
 - decidable in TA_λ 28
 - of typed terms 69
 - rule (Eq_β) in $TA_{\lambda+\beta}$ 52
 - with strong type-invariance 158
- β -equality (see β -conversion)
- β -expansion 5
 - sometimes preserves types 24
- β -nf (see β -normal forms)
- β -nf (set of all β -normal forms) 6
 - structural characterization 7
- β -normal forms 6
 - structure of 7
 - typed 69
 - structure of 110
 - long typed 110
 - structure of long typed 115
- β -redex 4
 - typed 69
- β -redex-occurrence 144
 - argument-part 144
 - function-part 144
 - newly-created 147
 - residuals of 145
- β -reduction 4
 - Church-Rosser theorem for 5
 - leftmost red. 6
 - length of a red. 5
 - non-cancelling red. 11
 - non-duplicating red. 11
 - preserves types (Subject-reduction thm.) 24
 - red. with strong type-invariance 158
 - typed red. 69
- $\beta\eta$ -contraction 8
- $\beta\eta$ -conversion 8
 - Church-Rosser theorem for 8
 - rule ($Eq_{\beta\eta}$) in $TA_{\lambda+\beta\eta}$ 52
- $\beta\eta$ -equality (see $\beta\eta$ -conversion)
- $\beta\eta$ -nf (see $\beta\eta$ -normal forms)
- $\beta\eta$ -nf (set of all $\beta\eta$ -normal forms) 9
- $\beta\eta$ -normal forms 9
 - seeking 10
 - typed 69
- $\beta\eta$ -redex 8
- $\beta\eta$ -reduction 8
 - Church-Rosser theorem for 8
 - Commuting lemma for 8
 - η -Postponement theorem for 8
- Γ (for arbitrary type-contexts) 14
- $\Gamma - x$ 15
- $\Gamma \vdash M$ 15
- Δ_L (see Curry-Howard mapping)
- Δ_λ (logic-to-lambda mapping) 82
- η -contraction, conversion 7
 - typed 69
- η -family 8
 - typed 112
- η -nf (see η -normal forms)
- η -normal forms 9
 - typed 69
- η -Postponement theorem 8
- η -redex 7
 - typed 69
- η -reduction 7
 - Church-Rosser theorem for 8
 - Commuting lemma with β 8
- λ -abstract 1
- λ -calculus, definition 1
 - basic facts 1–11
- λ -model 57
 - extensional 58
- λ -term (see term)
- λI -calculus, λI -terms 10
- λI -PT 94
- λK -calculus, λK -terms 10
- λK -PT 94
- $\bar{0}$ (Church numeral zero) 4
 - principal type of 50
- 1-property 96
- $\bar{1}$ (Church numeral 1) 4
 - principal type of 50
- 2-length of a position 129
- ∞ (possible length of a reduction) 5
- ∞ (possible no. of inhabitants) 109
- $\#()$ (cardinality of a set of terms) 109
- $(\rightarrow E), \rightarrow$ -elimination rule
 - in TA_λ 16
 - in propositional logics 76, 89
- $(\rightarrow I), \rightarrow$ -introduction rule
 - in TA_λ 16
 - in propositional logics 76
 - discharging assumptions 76–9
- $(\rightarrow I)_{\text{main}}, (\rightarrow I)_{\text{vac}}$ 16
- $*$ (for concatenation) 130
- $*$ (symbol in a position) 140, 153
- \cdot_β (meaning β -normal form of) 6
- $\cdot_{\beta\eta}$ (meaning $\beta\eta$ -normal form of) 9
- \cdot_η (meaning η -normal form of) 9
- \equiv (identity of terms etc.) 1
- \equiv_α (α -convertibility) 3
- \cong (isomorphism of components) 115, 143
- $=$ (identity of nos., sets, etc.) 1
- $=_{\text{ext}}$ 35
- $=_\beta$ (see β -conversion)
- $=_{\beta\eta}$ (see $\beta\eta$ -conversion)
- $=_{\beta[\eta]}$ (used for both $=_\beta, =_{\beta\eta}$) 53
- $=_\eta$ (see η -conversion)
- $\triangleright_{1\beta}$ (β -contracts to, see β -contraction)
- \triangleright_β (β -reduces to, see β -reduction)
- $\triangleright_{\beta\eta}$ ($\beta\eta$ -reduces to, see $\beta\eta$ -reduction)
- \triangleright_η (η -reduces to, see η -reduction)

\mapsto 15

contrasted with \vdash 19

\vdash_A 89

\vdash_{AD} 104

\vdash_{int} 79

$\vdash_{\text{no eq}}$ 53

\vdash_p 32

\vdash_β 52

$\vdash_{\beta\eta}$ 52

\vdash_λ (also see TA_λ , A_λ -deduction) 18

\models_β , $\models_{\beta\eta}$ 58

$[\ / \]$ (substitution notation) 3
in typed terms 68

$\{ \ / \ }$ (replacement notation) 67, 143

$\{ \ \}_\eta$ (η -family) 8, 112

$| \ |$ (length of a deduction) 149

$| \ |$ (length of a term) 2

of a typed term 67

$| \ |$ (length of a type) 13

$|| \ ||$ 13

M^\sharp (type-erasing) 65

\cup (union of substitutions) 35

\circ (composition of substitutions) 36

\upharpoonright (restriction of a substitution) 35

\upharpoonright (restriction of a context) 15