

Cambridge University Press  
0521018471 - Two-Level Functional Languages  
Flemming Nielson and Hanne Riis Nielson  
Frontmatter  
[More information](#)

---

**TWO-LEVEL FUNCTIONAL LANGUAGES**

Cambridge University Press  
0521018471 - Two-Level Functional Languages  
Flemming Nielson and Hanne Riis Nielson  
Frontmatter  
[More information](#)

## Cambridge Tracts in Theoretical Computer Science

*Managing Editor* Professor C.J. van Rijsbergen, Department of Computing Science,  
University of Glasgow

### Editorial Board

S. Abramsky, Department of Computing Science, Imperial College of Science  
and Technology  
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  
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*
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*
23. E.-R. Olderog *Nets, Terms and Formulas*
27. W.H. Hesselink *Programs, Recursion and Unbounded Choice*
29. P. Gärdenfors (ed) *Belief Revision*
30. M. Anthony & N. Biggs *Computational Learning Theory*
34. F. Nielson & H.R. Nielson *Two Level Functional Languages*

Cambridge University Press  
0521018471 - Two-Level Functional Languages  
Flemming Nielson and Hanne Riis Nielson  
Frontmatter  
[More information](#)

---

---

# TWO-LEVEL FUNCTIONAL LANGUAGES

---

FLEMMING NIELSON & HANNE RIIS NIELSON  
*Department of Computer Science  
Aarhus University, Denmark*



Cambridge University Press  
0521018471 - Two-Level Functional Languages  
Flemming Nielson and Hanne Riis Nielson  
Frontmatter  
[More information](#)

---

CAMBRIDGE UNIVERSITY PRESS  
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press  
The Edinburgh Building, Cambridge CB2 2RU, UK

Published in the United States of America by Cambridge University Press, New York

[www.cambridge.org](http://www.cambridge.org)  
Information on this title: [www.cambridge.org/9780521403849](http://www.cambridge.org/9780521403849)

© Cambridge University Press 1992

This publication 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 1992  
This digitally printed first paperback version 2005

*A catalogue record for this publication is available from the British Library*

ISBN-13 978-0-521-40384-9 hardback  
ISBN-10 0-521-40384-7 hardback

ISBN-13 978-0-521-01847-0 paperback  
ISBN-10 0-521-01847-1 paperback

# Contents

|          |   |            |
|----------|---|------------|
| <b>1</b> | <b>Introduction</b>                                       | <b>1</b>   |
| <b>2</b> | <b>Types Made Explicit</b>                                | <b>7</b>   |
| 2.1      | The Typed $\lambda$ -Calculus . . . . .                   | 8          |
| 2.2      | Type Analysis . . . . .                                   | 12         |
| 2.2.1    | Polytypes . . . . .                                       | 13         |
| 2.2.2    | The algorithm . . . . .                                   | 16         |
| 2.2.3    | Syntactic soundness and completeness . . . . .            | 22         |
| <b>3</b> | <b>Binding Time Made Explicit</b>                         | <b>33</b>  |
| 3.1      | The 2-Level $\lambda$ -Calculus . . . . .                 | 33         |
| 3.2      | Binding Time Analysis . . . . .                           | 46         |
| 3.2.1    | Binding time analysis of types . . . . .                  | 48         |
| 3.2.2    | Binding time analysis of expressions . . . . .            | 55         |
| 3.2.3    | Binding time analysis of programs . . . . .               | 68         |
| 3.3      | Improving the Binding Time Analysis . . . . .             | 72         |
| <b>4</b> | <b>Combinators Made Explicit</b>                          | <b>79</b>  |
| 4.1      | Mixed $\lambda$ -Calculus and Combinatory Logic . . . . . | 80         |
| 4.1.1    | Well-formedness . . . . .                                 | 83         |
| 4.1.2    | Combinator expansion . . . . .                            | 86         |
| 4.2      | Combinator Introduction . . . . .                         | 88         |
| 4.3      | Improving the Combinator Introduction . . . . .           | 102        |
| <b>5</b> | <b>Parameterized Semantics</b>                            | <b>107</b> |
| 5.1      | Types . . . . .   | 107        |
| 5.1.1    | Domain theory . . . . .                                   | 108        |
| 5.1.2    | Interpretation of types . . . . .                         | 116        |
| 5.2      | Expressions . . . . .                                     | 122        |
| 5.3      | Programs . . . . .  | 133        |

|          |  |            |
|----------|--|------------|
| <b>6</b> | <b>Code Generation</b>                                     | <b>139</b> |
| 6.1      | The Coding Interpretation . . . . .                        | 139        |
| 6.1.1    | First order aspects of the abstract machine . . . . .      | 143        |
| 6.1.2    | First order aspects of the coding interpretation . . . . . | 144        |
| 6.1.3    | Recursion . . . . .  | 148        |
| 6.1.4    | Higher-order aspects . . . . .                             | 152        |
| 6.2      | The Substitution Property . . . . .                        | 154        |
| 6.3      | Well-behavedness of the Code . . . . .                     | 163        |
| 6.3.1    | Definition of the well-behavedness predicates . . . . .    | 165        |
| 6.3.2    | Operations on execution sequences . . . . .                | 168        |
| 6.3.3    | Well-behavedness of operators . . . . .                    | 170        |
| 6.4      | Correctness of the Code . . . . .                          | 185        |
| 6.4.1    | Definition of the correctness predicates . . . . .         | 185        |
| 6.4.2    | Correctness of operators . . . . .                         | 188        |
| <b>7</b> | <b>Abstract Interpretation</b>                             | <b>207</b> |
| 7.1      | Strictness Analysis . . . . .                              | 207        |
| 7.1.1    | Types . . . . .  | 208        |
| 7.1.2    | The safety property . . . . .                              | 214        |
| 7.1.3    | Expressions . . . . .                                      | 219        |
| 7.1.4    | Proof of safety . . . . .                                  | 222        |
| 7.2      | Improved Strictness Analysis . . . . .                     | 232        |
| 7.2.1    | The <b>Case</b> construct . . . . .                        | 232        |
| 7.2.2    | Tensor products . . . . .                                  | 237        |
| <b>8</b> | <b>Conclusion</b>  | <b>253</b> |
| 8.1      | Optimized Code Generation . . . . .                        | 253        |
| 8.1.1    | Using local strictness information . . . . .               | 254        |
| 8.1.2    | Using right context information . . . . .                  | 257        |
| 8.1.3    | Using left context information . . . . .                   | 261        |
| 8.1.4    | Pre-evaluation of arguments . . . . .                      | 266        |
| 8.2      | Denotational Semantics . . . . .                           | 268        |
| 8.2.1    | The language <b>Imp</b> . . . . .                          | 269        |
| 8.2.2    | Transformations on the semantic specification . . . . .    | 272        |
| 8.2.3    | Towards a compiler . . . . .                               | 278        |
| 8.3      | Research Directions . . . . .                              | 280        |
|          | <b>Bibliography</b>  | <b>285</b> |
|          | <b>Summary of Transformation Functions</b>                 | <b>293</b> |
|          | <b>Index</b>   | <b>295</b> |

## List of Figures

|     |  |     |
|-----|--|-----|
| 1.1 | Overview of Chapters 2, 3 and 4 . . . . .                                      | 5   |
| 1.2 | Overview of Chapters 5, 6, 7 and 8 . . . . .                                   | 6   |
| 3.1 | The structure of $\mathcal{L}$ binding times . . . . .                         | 35  |
| 3.2 | Comparison of the $\mathcal{L}$ -level types for <code>reduce</code> . . . . . | 48  |
| 8.1 | Compiler generation . . . . .  | 279 |

# List of Tables

|      |   |     |
|------|---|-----|
| 2.1  | The typed $\lambda$ -calculus . . . . .   | 9   |
| 2.2  | The untyped $\lambda$ -calculus . . . . .   | 9   |
| 2.3  | Well-formedness of the typed $\lambda$ -calculus . . . . .                                    | 11  |
| 2.4  | $\mathcal{E}_{\text{TA}}^C$ : Type analysis of expressions (part 1) . . . . .                 | 18  |
| 2.5  | $\mathcal{E}_{\text{TA}}^C$ : Type analysis of expressions (part 2) . . . . .                 | 19  |
|      |   |     |
| 3.1  | The 2-level $\lambda$ -calculus . . . . .   | 34  |
| 3.2  | Well-formed 2-level types for <b>reduce</b> . . . . .   | 37  |
| 3.3  | Well-formedness of the 2-level types . . . . .  | 38  |
| 3.4  | Well-formedness of the 2-level $\lambda$ -calculus (part 1) . . . . .                         | 41  |
| 3.5  | Well-formedness of the 2-level $\lambda$ -calculus (part 2) . . . . .                         | 42  |
| 3.6  | $\mathcal{T}_{\text{BTA}}$ : Binding time analysis of types . . . . .                         | 49  |
| 3.7  | $\mathcal{E}_{\text{BTA}}^C$ : Binding time analysis of expressions (part 1) . . . . .        | 57  |
| 3.8  | $\mathcal{E}_{\text{BTA}}^C$ : Binding time analysis of expressions (part 2) . . . . .        | 60  |
| 3.9  | $\mathcal{E}_{\text{BTA}}^C$ : Binding time analysis of expressions (part 3) . . . . .        | 62  |
| 3.10 | $\mathcal{E}_{\text{BTA}}^C$ : Binding time analysis of expressions (part 4) . . . . .        | 65  |
| 3.11 | $\mathcal{E}_{\text{BTA}}^C$ : Binding time analysis of expressions (part 5) . . . . .        | 66  |
|      |   |     |
| 4.1  | The mixed $\lambda$ -calculus and combinatory logic . . . . .                                 | 81  |
| 4.2  | Well-formedness of the mixed $\lambda$ -calculus and combinatory logic (1) . . . . .          | 84  |
| 4.3  | Well-formedness of the mixed $\lambda$ -calculus and combinatory logic (2) . . . . .          | 85  |
| 4.4  | $\mathcal{E}_{\text{CI}}^{penv}$ : Combinator introduction for expressions (part 1) . . . . . | 93  |
| 4.5  | $\mathcal{E}_{\text{CI}}^{penv}$ : Combinator introduction for expressions (part 2) . . . . . | 94  |
| 4.6  | $\mathcal{E}_{\text{CI}}^{penv}$ : Combinator introduction for expressions (part 3) . . . . . | 95  |
| 4.7  | $\mathcal{E}_{\text{CI}}^{penv}$ : Combinator introduction for expressions (part 4) . . . . . | 96  |
| 4.8  | $\mathcal{E}_{\text{CI}}^{penv}$ : Combinator introduction for expressions (part 5) . . . . . | 97  |
|      |   |     |
| 5.1  | Operators and their actual types . . . . .  | 125 |
|      |   |     |
| 6.1  | Configurations of the abstract machine . . . . .  | 140 |
| 6.2  | Transition relation for the abstract machine . . . . .  | 141 |
| 6.3  | The coding interpretation <b>K</b> (part 1) . . . . .   | 145 |
| 6.4  | The coding interpretation <b>K</b> (part 2) . . . . .   | 146 |
| 6.5  | The coding interpretation <b>K</b> (part 3) . . . . .   | 147 |



---

|     |  |     |
|-----|--|-----|
| 7.1 | Type part of <b>A</b> . . . . .                              | 208 |
| 7.2 | Safety predicate for types of kind <b>r</b> . . . . .        | 214 |
| 7.3 | Expression part of <b>A</b> (part 1) . . . . .               | 219 |
| 7.4 | Expression part of <b>A</b> (part 2) . . . . .               | 220 |
| 7.5 | Expression part of <b>A</b> (part 3) . . . . .               | 221 |
| 8.1 | The optimizing interpretation <b>O</b> (part 1) . . . . .    | 254 |
| 8.2 | The optimizing interpretation <b>O</b> (part 2) . . . . .    | 255 |
| 8.3 | The optimizing interpretation <b>O</b> (part 3) . . . . .    | 256 |
| 8.4 | Semantics of <b>Imp</b> -expressions . . . . .               | 271 |
| 8.5 | Semantics of <b>Imp</b> -statements . . . . .                | 272 |
| 8.6 | Semantics of <b>Imp</b> -declarations and programs . . . . . | 273 |

# Preface

The subject area of this book concerns the implementation of functional languages. The main perspective is that part of the implementation process amounts to

*making computer science concepts explicit*

in order to facilitate the application, and the development, of general frameworks for program analysis and code generation.

This is illustrated on a specimen functional language patterned after the  $\lambda$ -calculus:

- *Types* are made explicit in Chapter 2 by means of a Hindley/Milner/Damas type analysis.
- *Binding times* are made explicit in Chapter 3 using an approach inspired by the one for type analysis. The binding times of chief interest are *compile-time* and *run-time*.
- *Combinators* are made explicit in Chapter 4 but only for run-time computations whereas the compile-time computations retain their  $\lambda$ -calculus syntax.

The advantages of this approach are illustrated in the remainder of the book where the emphasis also shifts from a ‘syntactic perspective’ to a more ‘semantic perspective’:

- A notion of *parameterized semantics* is defined in Chapter 5 and this allows a wide variety of semantics to be given.
- It is illustrated for *code generation* in Chapter 6. Code is generated for a structured abstract machine and the correctness proof exploits Kripke-logical relations and layered predicates.
- It is illustrated for *abstract interpretation* in Chapter 7. We generalize Wadler’s strictness analysis to general lists, show the correctness using logical relations, and illustrate the similarity between tensor products and Wadler’s case analysis.

Finally, Chapter 8 discusses possible ways of extending the development. This includes the use of abstract interpretation to obtain an improved code generation that may still be proved correct. We also illustrate the role of the mixed  $\lambda$ -calculus and combinatory logic as a metalanguage for denotational semantics; this allows a systematic approach to compiler generation from semantic specifications.

### Notes for the Reader

This book is intended for researchers and for students who already have some formal training. Much of the work reported here has been documented elsewhere in the scientific literature and we have therefore aimed at a style of exposition where we concentrate on the main insights and methods, including proofs and proof techniques, but where we feel free to refer to the literature for technically complex generalizations and details of tedious proofs. To facilitate this, we provide bibliographic notes covering variations of the technical development. Our notation is mostly standard but we find ‘ $\hookrightarrow$ ’ a more readable notation for ‘partial functions’ than ‘ $\rightarrow$ ’.

### Acknowledgements

The research reported here has been supported by The Danish Natural Sciences Research Council. The presentation has benefited from comments from our students and colleagues, in particular Torben Amtoft, Poul Christiansen, Fritz Henglein, Torben Lange, Jens Mikkelsen, Thorleif Nielson, Jens Palsberg, Hans J. Pedersen, Anders Pilegaard, Kirsten L. Solberg, Bettina B. Sørensen and Phil Wadler. Finally, David Tranah made a number of suggestions concerning how to improve the presentation.

Aarhus, January 1992

Flemming Nielson  
Hanne Riis Nielson