Book review

Basic Simple Type Theory, J. Roger Hindley

Hans-Joerg Tiede
Departments of Computer Science, Cognitive Science and Linguistics, Lindley Hall, Indiana University, Bloomington, IN 47405, USA, htiede@indiana.edu

5/10/1998, To appear in the Journal of Logic, Language, and Information

Basic Simple Type Theory
J. Roger Hindley
Cambridge
1997
Cambridge Tracts in Theoretical Computer Science 42
Cambridge University Press
$39.95/£25.00 (harback)
xi + 198 pages
ISBN: 0-521-46518-4

Type theory has become an increasingly important topic in many areas, including theoretical computer science (Barendregt, 1992), the development of programming languages (Mitchell, 1996), logic (Girard, [et al.], 1989) and linguistics (van Benthem, 1991). The study of type theory was initiated by Russell's attempt to avoid paradoxes in set theory and the foundations of mathematics. In its computational setting, type theory is supposed to give us information about the behavior of programs only with respect to their domain and range, so that if a program has a type $\alpha \to \beta$ (mapping terms of type α to terms of type β) and we apply that program to a term of type α, we can expect to receive a term of type β as the output. In typed λ-calculi, we consider statements of the form: $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash M : \beta$, read "under the assignments of types $\alpha_1, \ldots, \alpha_n$ to the distinct variables $x_1, \ldots, x_n$, possibly occurring free in M, M can be assigned the type β."

Among the different type systems that play a role in the above mentioned areas, the simply typed λ-calculus ($\lambda_\to$) is of central importance. In $\lambda_\to$, only the formation of function types ($\alpha \to \beta$) from a collection of primitive types (possibly including type variables) is allowed. Thus, it does not make use of other type connectives or quantification over types. Its importance is due to the fact that type systems of programming languages usually are extensions of $\lambda_\to$ and many type systems under consideration in logic and linguistics are restrictions of $\lambda_\to$.

The book under review is only concerned with $\lambda_{\rightarrow}$. This feature distinguishes it from many other introductions to type theory (see, for example, the following selection, which is by no means complete or canonical: Barendregt, 1992, Mitchell, 1996, Turner, 1997) which cover a wide range of type theories, of which there certainly are enough to confuse the beginning student of type theory. In particular, this book is concerned with type assignment or Curry typing: algorithms that, given an untyped λ-term, check whether a type can be assigned to that term.

The main topics of the book are the principal type algorithm, the relationship between type theory and logic, the converse principal type algorithm, and an algorithm for counting the number of inhabitants of a type. Much of this material was previously available only in unpublished Ph.D. dissertations (in particular Ben-Yelles' 1979 dissertation), making it available in published form for the first time. Hindley integrates most of the important computational and logical features of $\lambda_{\rightarrow}$ in this book, which should make it appealing to researchers in both areas. He also includes a wealth of historical references and pointers to current research, making this book a useful reference tool.

The first chapter contains a quick review of the necessary background of the untyped λ-calculus. The second chapter introduces types and a type assignment calculus for $\lambda_{\rightarrow}$. The introduction of an assignment calculus leads to the notion of principal types. A principal or most general type is a type that can be assigned to a term, such that all other types that can be assigned to that term are substitution instances of the principal type. This can be illustrated by considering the identity function $\lambda x.x$. The type assignment system of $\lambda_{\rightarrow}$ can assign any type of the form $\alpha \rightarrow \alpha$ to it. Thus, for any type variable a, $a \rightarrow a$ is a principal type of $\lambda x.x$, since for any α, $\alpha \rightarrow \alpha$ can be obtained from $a \rightarrow a$ by the substitution $[a \mapsto \alpha]$. It should be noted that principal types are unique up to renaming of type variables. Hindley's principal type theorem states that every term typable in $\lambda_{\rightarrow}$ has a principal type. This theorem is proved by giving an algorithm that assigns to any term its principal type, if this term is typeable, or else terminates with a correct statement that this term is not typable. This algorithm makes use of unification of types, and is introduced in the book after a good introduction to unification in general.

The converse principal type theorem states that any type that is the type of some term is the principal type of some term. Its proof for $\lambda_{\rightarrow}$ and other systems depends on a variant of the resolution rule of automated theorem proving, called the rule of condensed detachment.

An important contribution of this book is a very clear exposition of the Curry-Howard isomorphism, which describes the relationship

between $\lambda_{\rightarrow}$ and implicational intuitionistic logic. The chapter also includes a discussion of this relationship for restricted classes of λ-terms and substructural logics, which are logics that result from removing some or all of the structural rules of Gentzen's sequent calculus (weakening, exchange, and contraction). The substructural logics which are discussed in the book are relevance logic (lacking weakening), BCK (lacking contraction), and BCI logic (lacking both weakening and contraction), the last of which is the implicational fragment of linear logic.

Finally, the book introduces tools for studying the inhabitants of a type, i.e. the terms that can be assigned a given type. This chapter includes algorithms for counting the number of inhabitants and for deciding whether the number of inhabitants of a type is finite or infinite. While the question of how many terms can be assigned a given type may not seem important at first, the algorithms presented in this chapter are very interesting for two reasons. First, using the correspondence between logic and type theory, we can use this algorithm to answer the question of how many different normal form proofs there are for a given formula in intuitionistic logic, including if there are only finitely many, infinitely many, or none. Also, the algorithms for counting the number of inhabitants and deciding whether this number is finite or infinite have interesting similarities to pumping lemmas in formal language theory and to algorithms deciding the finiteness of regular languages. These algorithms relate the, suitably defined, depth of special inhabitants of a type with the number of distinct type variables occurring in that type. It can be shown that if a type has special inhabitants of a depth that is larger than the number of distinct type variables occurring in that type, it has inhabitants of depth greater than n, for all n, and hence infinitely many inhabitants.

Also included are chapters on type theory with equality and its semantics, as well as type theory based on typed terms rather than type assignment (the so-called "Church approach" to typing).

The book contains many exercises with solutions to some of these, making it useful as a textbook. However, the strength of the book, i.e. its thorough discussion of one type system rather than a massive accumulation of different type systems, makes it less appealing to use as a textbook on its own for classes in type theory for logicians or computer scientists. If used as an introduction to type theory for logicians, more material on the relation between cut elimination and normalization should be included. For computer scientists, the importance of other type systems makes it difficult to justify teaching a whole class only on $\lambda_{\rightarrow}$. In either case, the situation can be remedied by supplementing the course with more advanced material, for which the book is excellent preparation.

While the book either includes or refers to most of the important results about $\lambda_{\rightarrow}$, I would like to point out the following omissions. Although all of these topics are referred to in Hindley and Seldin (1986), their inclusion in this book would have been warranted by their importance:

- A proof of strong normalization for $\lambda_{\rightarrow}$. Although Hindley points to relevant places to find such a proof, its inclusion would have made the book more self contained.
- The connection with cartesian closed categories. While Hindley discusses some semantic aspects of $\lambda_{\rightarrow}$, category theoretic models of type theory, and in particular of $\lambda_{\rightarrow}$, have become very important in computer science and logic. On the other hand, good introductions to this topic are available both for logicians (Lambek & Scott, 1986) and for computer scientists (Crole, 1993). However, these texts are not referred to in the book.
- The third omission concerns the question which number theoretic functions are definable in $\lambda_{\rightarrow}$. While strong normalization implies that only total functions, and therefore not all computable functions, are definable, Schwichtenberg's (1976) characterization of the functions definable in $\lambda_{\rightarrow}$ as coinciding with the extended polynomials should have been included or at least referred to in this book. The reason that this result appears so important is that researchers in programming languages would, of course, like to know how much of the computational power of the untyped λ-calculus is lost by the restriction to typed λ-calculi.

All in all, however, this book is an extremely useful introduction to type theory for non-specialists and an equally useful reference for researchers.

References

Barendregt, H.P., 1992, "Lambda calculi with types," in: S. Abramsky [et al.], eds., *Handbook of Logic in Computer Science*. Oxford: Clarendon Press: 117-309.

van Benthem, J., 1991, *Language in Action*. Amsterdam: North Holland.

Ben-Yelles, C.-B., 1979, *Type-assignment in the lambda-calculus; syntax and semantics*. Unpublished Ph.D. Thesis. Mathematics Dept., University of Wales Swansea.

Crole, R.L., 1993, *Categories for Types*. Cambridge: Cambridge University Press.

Girard, J.-Y, Y. Lafont, and P. Taylor, 1989, *Proofs and Types*. Cambridge: Cambridge University Press.

Hindley, J.R. and J.P. Seldin, 1986, *Introduction to Combinators and λ-calculus*. Cambridge: Cambridge University Press.

Lambek, J. and P.J. Scott, 1986, *Introduction to Higher Order Categorical Logic*. Cambridge: Cambridge University Press.

Mitchell, J. C., 1996, *Foundations for Programming Languages*. Cambridge, MA: MIT Press.

Schwichtenberg, H., 1976, "Definierbare Funktionen im λ-Kalkül mit Typen," in: *Archiv für mathematische Logik und Grundlagenforschung* 17: 113-114.

Turner, R., 1997, "Types," in: J. van Benthem and A. ter Meulen, eds., *Handbook of Logic and Language*. Amsterdam: Elsevier: 535-586.