Ordinals and Typed Lambda Calculus

Typed Lambda Calculus

Andrés Sicard-Ramírez

Universidad EAFIT

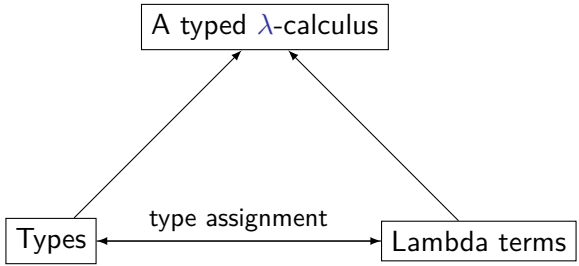
Semester 2018-2

(Last modification: 7th May 2025)

Introduction

General picture

The term 'typed λ-calculus' denotes a family of formal systems. Every typed λ-calculus is a formal theory compound of formulae, axioms and rules of inference.*



*See, e.g. [Bar1992] and [HS2008].

Introduction

Example (Types)

- Mathematics
  $f : \mathbb{Z} \to \{\text{True}, \text{False}\}$
  $f(x) = ...$
- Haskell

```
f :: Bool → Int
f b = ...

map :: (a → b) → [a] → [b]
map f as = ...
```

Introduction

Example (Types)

$$
\begin{aligned}
\sigma, \tau ::= {} & x && \text{(type variables)} \\
\mid {} & c && \text{(constant types)} \\
\mid {} & \bot && \text{(the empty type)} \\
\mid {} & \top && \text{(the unit type)} \\
\mid {} & \sigma \to \tau && \text{(function types)} \\
\mid {} & \sigma \times \tau && \text{(product types)} \\
\mid {} & \sigma + \tau && \text{(disjoint union types)}
\end{aligned}
$$

What is a Type?

- A type is a set.

What is a Type?

- A type is a set.
- Types as ranges of significance of propositional functions. Let $\varphi(x)$ be a (unary) propositional function. The type of $\varphi(x)$ is the range within which x must lie if $\varphi(x)$ is to be a proposition [Rus1938, Appendix B: The Doctrine of Types].

  In modern terminology, Rusell's types are domains of propositional functions.

What is a Type?

- A type is a set.
- Types as ranges of significance of propositional functions. Let $\varphi(x)$ be a (unary) propositional function. The type of $\varphi(x)$ is the range within which x must lie if $\varphi(x)$ is to be a proposition [Rus1938, Appendix B: The Doctrine of Types].

  In modern terminology, Rusell's types are domains of propositional functions.

  Example
  Let $\varphi(x)$ be the propositional function 'x is a prime number'. Then $\varphi(x)$ is a proposition only when its argument is a natural number.

$$\varphi : \mathbb{N} \to \{\mathrm{False}, \mathrm{True}\}$$
$$\varphi(x) = x \text{ is a prime number.}$$

What is a Type?

- Hoare's 'Notes on Data Structuring' [Hoa1972, pp. 92-93]:

What is a Type?

- Hoare's 'Notes on Data Structuring' [Hoa1972, pp. 92-93]:

  Thus there is a high degree of commonality in the use of the concept of type by mathematicians, logicians and programmers. The salient characteristics of the concept of type may be summarised:

  1. *A type determines the class of values which may be assumed by a variable or expression.*
  2. *Every value belongs to one and only one type.*
  3. *The type of a value denoted by any constant, variable, or expression may be deduced from its form or context, without any knowledge of its value as computed at run time.*

(continued on next slide)

What is a Type?

- Hoare's 'Notes on Data Structuring' (continuation)

  4. *Each operator expects operands of some fixed type, and delivers a result of some fixed type (usually the same). Where the same symbol is applied to several different types (e.g. $+$ for addition of integers as well as reals), this symbol may be regarded as ambiguous, denoting several different actual operators. The resolution of such systematic ambiguity can always be made at compile time.*

  5. *The properties of the values of a type and of the primitive operations defined over them are specified by means of a set of axioms.*

(continued on next slide)

What is a Type?

- Hoare's 'Notes on Data Structuring' (continuation)

  6. *Type information is used in a high-level language both to prevent or detect meaningless constructions in a program, and to determine the method of representing and manipulating data on a computer.*

  7. *The types in which we are interested are those already familiar to mathematicians; namely, Cartesian Products, Discriminated Unions, Sets, Functions, Sequences, and Recursive Structures.*

What is a Type?

- 'A type system is a tractable syntactic method for proving the absence of certain program behaviours by classifying phrases according to the kinds of values they compute.' [Pie2002, p. 1]

What is a Type?

- 'A type system is a tractable syntactic method for proving the absence of certain program behaviours by classifying phrases according to the kinds of values they compute.' [Pie2002, p. 1]
- 'A type is an approximation of a dynamic behaviour that can be derived from the form of an expression.' [KS2008, p. 8]

What is a Type?

- 'A type system is a tractable syntactic method for proving the absence of certain program behaviours by classifying phrases according to the kinds of values they compute.' [Pie2002, p. 1]
- 'A type is an approximation of a dynamic behaviour that can be derived from the form of an expression.' [KS2008, p. 8]

Related readings

- 'The Triumph of Types: *Principia Mathematica's* Impact on Computer Science' [Con2010].
- 'Against a Universal Definition of 'Type'' [Pet2015].

What is a Type?

- BHK (Brouwer, Heyting, Kolmogorov) interpretation: A type is a set, a proposition, a problem and a specification.*

Type A	Term $a : A$	Interpretation
A is a set	a is an element of the set A	$A \neq \emptyset$
A is a proposition	a is a proof (construction) of the proposition A	A is true
A is a problem	a is a method of solving the problem A	A is solvable
A is a specification	a is a program than meets the specification A	A is satisfiable

*See, e.g. [Mar1984].

Applications of Typed Lambda Calculus

- To carry useful information for programs optimisation.

Applications of Typed Lambda Calculus

- To carry useful information for programs optimisation.
- To reduce the semantic gap between programs and their properties.

Applications of Typed Lambda Calculus

- To carry useful information for programs optimisation.
- To reduce the semantic gap between programs and their properties.
- The propositions-as-types-principle: Computational interpretation of logical constants.

Applications of Typed Lambda Calculus

- To carry useful information for programs optimisation.
- To reduce the semantic gap between programs and their properties.
- The propositions-as-types-principle: Computational interpretation of logical constants.
- Unified programing logics (programs, specification and satisfaction relation).

References

[Bar1992] Henk Barendregt. Lambda Calculi with Types. In: Handbook of Logic in Computer Science. Volume 2. Ed. by S. Abramsky, Dov M. Gabbay and T. S. E. Maibaum. Clarendon Press, 1992, pp. 117–309 (cit. on p. 2).

[Con2010] Robert L. Constable. The Triumph of Types: *Principia Mathematica's* Impact on Computer Science. Presented at the *Principia Mathematica* anniversary symposium. 2010 (cit. on pp. 12–14).

[HS2008] J. Roger Hindley and Jonathan P. Seldin. Lambda-Calculus and Combinators. An Introduction. Cambridge University Press, 2008 (cit. on p. 2).

[Hoa1972] C. A. R. Hoare. Notes on Data Structuring. In: Structured Programming. Ed. by O.-J. Dahl, E. W. Disjkstra and C. A. R. Hoare. Academic Press, 1972, pp. 83–174 (cit. on pp. 8, 9).

[KS2008] Oleg Kiselyov and Chung-chieh Shan. Interpreting Types as Abstract Values. Formosan Summer School on Logic, Language and Computacion (FLOLAC 2008). 2008 (cit. on pp. 12–14).

[Mar1984] Per Martin-Löf. Intuitionistic Type Theory. Bibliopolis, 1984 (cit. on p. 15).

References

[Pet2015] Tomas Petricek. Against a Universal Definition of 'Type'. In: Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2015). ACM, 2015, pp. 254–266. DOI: 10.1145/2814228.2814249 (cit. on pp. 12–14).

[Pie2002] Benjamin C. Pierce. Types and Programming Languages. MIT Press, 2002 (cit. on pp. 12–14).

[Rus1938] Bertrand Russell. The Principles of Mathematics. 2nd ed. W. W. Norton & Company, Inc, 1938 (1903) (cit. on pp. 5–7).