Introduction to Lambda Calculus

Maciek Makowski

May 14, 2014

1 Motivation

Software is pervasive in the modern world and has influence over many aspects of our lives. In some cases, such as avionics or medical equipment control, human life depends on the correctness of software. Yet, high profile cases of bugs[1] do not inspire confidence in the state of software engineering. The "software crisis" is a phenomenon recognised by practitioners of the field. A number of ways to address the reliability issue has been proposed, from reliance on programmer's discipline[8][9], through tools that analyse programs written in popular languages for suspicious patterns[10], to languages that restrict valid programs to ones whose properties can be formally proven. The latter approach relies on a body of theoretical knowledge that can be intimidating at first sight. It turns out, however, that much of the required insight is built on systematic extensions of a very simple formal system – the lambda calculus. Familiarity with the fundamentals of lambda calculus is a prerequisite for proficiency with modern software engineering tools. Fortunately, thanks to the simplicity of the calculus, it is easily achievable.

2 Syntax

Terms of lambda calculus represent anonymous functions over some predefined set of variables X:

$$\begin{array}{lll} \langle term \rangle ::= & x & \text{(variable)} \\ \quad\quad\quad \mid & \lambda x.\langle term \rangle & \text{(abstraction)} \\ \quad\quad\quad \mid & \langle term \rangle\ \langle term \rangle & \text{(application)} \end{array}$$

[1]Infamous historical examples include Mars Climate Orbiter's inconsistent usage of units of measurement[4] and Therac-25 radiation therapy overdoses[5]. Recent faults such as security-related Apple goto fail[6] and OpenSSL Heartbleed bug[7], while not life-threatening, had wide-ranging implications for the security of e-commerce and privacy of internet users.

where $x \in X$. Examples:

- v_1
- $x\ y$
- $(\lambda a.\lambda b.a)\ c\ (\lambda a.b)$

Within a term, occurrences of variables that are not bound by enclosing abstraction are called *free.* In examples below the underlined variable occurrences are free:

- $\lambda x.\underline{y}$
- $\lambda a.\underline{b}\ a\ (\lambda b.b)$

Note that the same variable name might have both bound and free occurrences within a term[2]. Terms with no free occurrences are known as *closed terms*, or *combinators.*

3 Rewriting Rules

The grammar tells us how to generate arbitrary lambda-terms. In order for this representation of mathematical functions to be faithful, we need to be able to unify different representations of the same value. Lambda calculus allows us to do that syntactically, by applying three kinds transformations:

- renaming of bound variables (α-*conversion*); e.g. $(\lambda x.x\,y)\ (\lambda x.x) \longleftrightarrow_\alpha (\lambda a.a\,y)\ (\lambda b.b)$
- removal of abstraction under application (β-*reduction*); e.g. $(\lambda x.x\,y)\ (\lambda x.x) \longrightarrow_\beta y$
- introduction/removal of redundant abstraction (η-*conversion*); e.g. $\lambda x.y\,x \longleftrightarrow_\eta y$

TODO: note on subtleties in substitution

TODO: expand on each rule

[2]Variable capture is a potential source of subtle bugs in a practical implementation. For that reason a convenient way to represent lambda terms when implementing evaluation is De Bruijn encoding. It replaces variable names with numerical index of the lambda that binds given variable occurrence TODO: example

4 Foundational Theory

What exactly is the number 2? Foundational theories of mathematics attempt to provide a concrete answer in terms of some primitive objects. The best known example is that based in set theory. Using Peano's arithmetic where $2 = S(S(0))$ (S is the successor function) natural numbers can be modelled as follows:

$$\begin{aligned} 0 &= \emptyset \\ 1 &= \{\emptyset\} \\ 2 &= \{\{\emptyset\}, \emptyset\} \end{aligned}$$

$$\boxed{\text{In general, } S(n) = n \cup \{n\}.}$$

It turns out that all known mathematical concepts can be stated in terms of set theory. Lambda calculus was conceived by Church as an alternative foundational theory in which mathematics can be embedded[2]. How do we model natural numbers in it?

$$\begin{aligned} 0 &= \lambda s.\lambda z.z \\ 1 &= \lambda s.\lambda z.s\, z \\ 2 &= \lambda s.\lambda z.s\,(s\, z) \end{aligned}$$

$$\boxed{\text{In general, } S(n) = \lambda s.\lambda z\, \underbrace{s\,(\ldots s\,(s}_{n}\ z)\ldots).}$$

5 Model of Computation

We have seen that simple arithmetic operations can be expressed in TODO

6 Programming Language

TODO

7 Church-Rosser Theorem

TODO

8 Curry-Howard Correspondence

TODO

9 Further Reading

A direct inspiration for this talk was the presentation of lambda calculus in [1]. The book is very well written and builds a sophisticated type system in easy to follow steps, starting from untyped lambda calculus.

For a succinct but rigorous introduction to lambda calculus see [3].

TODO

References

[1] Benjamin C. Pierce, *Types and Programming Languages*, `http://www.cis.upenn.edu/~bcpierce/tapl/`

[2] Klaus Grue, *Lambda calculus as a foundation of mathematics*, `http://www.diku.dk/~grue/papers/church/church.html`

[3] Henk Berendregt, Erik Barendsen, *Introduction to Lambda Calculus*, `http://www.cse.chalmers.se/research/group/logic/TypesSS05/Extra/geuvers.pdf`

[4] NASA, *Mars Climate Orbiter Team Finds Likely Cause of Loss*, `http://www.jpl.nasa.gov/news/releases/99/mcoloss1.html`

[5] Nancy Leveson, Clark S. Turner, *An Investigation of the Therac-25 Accidents* `http://courses.cs.vt.edu/cs3604/lib/Therac_25/Therac_1.html`

[6] *CVE-2014-1266*, `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1266`

[7] *CVE-2014-0160*, `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160`

[8] Robert C. Martin, *Clean Code*

[9] various authors, *CERT C Coding Standard*, `https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=524435`

[10] Nick Rutar, Christian B. Almazan, Jeffrey S. Foster, *A Comparison of Bug Finding Tools for Java*, `http://www.cs.umd.edu/~jfoster/papers/issre04.pdf`