Introduction to Lambda Calculus

Maciek Makowski

May 24, 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[9][10], through tools that analyse programs written in popular languages for suspicious patterns[11], 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 appear intimidating. 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.

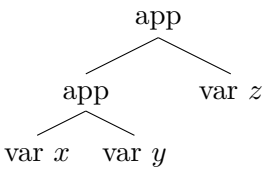
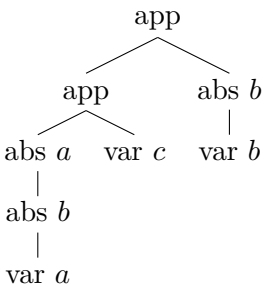
2 Syntax

Given a set of variables X, terms of lambda calculus are generated by the following grammar:

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

[1] Infamous historical examples include Mars Climate Orbiter's inconsistent usage of units of measurement[5] and Therac-25 radiation therapy overdoses[6]. Recent faults such as security-related Apple goto fail[7] and OpenSSL Heartbleed bug[8], while not life-threatening, had wide-ranging implications for the security of e-commerce and privacy of internet users.

where $x \in X$. Notational convention is that application binds to the left, the full-stop can be treated as an opening parenthesis that extends until the end of the sub-term and redundant parentheses are omitted. Examples of lambda terms as they are typically written, together with ASTs they represent:

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

Within a term, occurrences of variables that are not bound by enclosing abstraction – i.e. where the variable name does not appear in any abs node on the path to the root of the AST – are called *free*. In the examples below free occurrences are underlined:

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

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

[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 for example de Bruijn representation of $\lambda a.\lambda b.a\ b\ c$ would be $\lambda.\lambda.1\ 0\ 2$ under naming context $\{c \mapsto 2\}$. The naming context is required to map free variables to indices.

3 Rewriting Rules

Lambda terms are not of much use without some operations that can be performed on them. Two[3] operations we will use going forward are presented below.

3.1 Renaming of variables

It is often convenient to identify terms of the same structure that differ only by the names of bound variables. This intuition is captured by an operation called *α-conversion* that consistently renames variables.

Example: $(\lambda x.x\,y)\ (\lambda x.x) \longleftrightarrow_\alpha (\lambda a.a\,y)\ (\lambda b.b)$ Variable renaming is known as .

3.2 Removal of abstraction under application

$(\lambda x.x\,y)\ (\lambda x.x) \longrightarrow_\beta y$

β-reduction

This is the key operation of lambda calculus, as it represents a *computation*. It allows us to view a lambda term as a program that can be evaluated to a final value (i.e. a term that cannot be further reduced).

4 Programming

We claimed that a lambda term can be treated as a computer program. To substantiate this, let us see how familiar elements of programming languages can be represented in lambda calculus.

4.1 Conditionals

TODO

4.2 Numbers

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

[3]The third frequently applied operation is introduction/removal of abstraction (η-conversion): $\lambda x.M\,x \longleftrightarrow_\eta M$ for any term M. We will not require it in this presentation of lambda calculus.

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

TODO[4]

4.3 Loops

TODO

5 Model of Computation

TODO

6 Church-Rosser Theorem

TODO

7 Curry-Howard Correspondence

TODO

8 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/`

[4] Seeing how Peano arithmetic can be defined in lambda calculus, and drawing parallels with Zermelo-Fraenkel set-theoretical model of natural numbers and the build-out of other mathematical constructions on this basis, one could ask: can untyped lambda calculus be treated as a foundational theory in which all known mathematical concepts can be stated? Alonso Church had that in mind when he conceived lambda calculus, but it was soon proven by his studens, Kleene and Rosser[12], that in fact the calculus as we present it here is inconsistent, i.e. any proposition can be proven in it.

[2] Morten Heine B. Sørensen, Paweł Urzyczyn, *Lectures on the Curry-Howard Isomorphism*, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.7385

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

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

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

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

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

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

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

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

[11] 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

[12] Wikipedia, *Kleene-Rosser paradox*, https://en.wikipedia.org/wiki/Kleene-Rosser_paradox