An untyped λ-calculus, *UL*

Principles of Programming Languages

Mark Armstrong

Fall 2020

1 Preamble

1.1 TODO Notable references

- Benjamin Pierce, "Types and Programming Languages"
  - Chapter 5, The Untyped Lambda-Calculus

1.2 TODO Table of contents

2 Introduction

In this section we construct our first simple programming language, an untyped λ-calculus (lambda calculus).

More specifically, we construct a λ-calculus without (static) type checking (enforcement), but including the natural numbers and booleans.

2.1 What is the λ-calculus?

The λ-calculus is, put simply, a notation for forming and applying functions.

- Because the function (procedure, method, subroutine) abstraction gives us a means of representing control flow, if we have a means of representing data, the λ-calculus is a Turing-complete model of computation.

2.2 History

The (basic) λ-calculus as we know it was famously invented by Alonzo Church in the 1920s.

- This was one culmination of a great deal of work by mathematicians investigating the foundations of mathematics.

As mentioned, the λ-calculus is a Turing-complete model of computation.

- Other models proposed around the same time include
  - the Turing machine itself (due to Alan Turing), and
  - the general recursive functions (due to Stephen Cole Kleene.)
- Hence the "Church" in the "Church-Turing thesis".

The λ-calculus has since seen widespread use in the study and design of programming languages.

- It is useful both as a simple programming language, and
- as a mathematical object about which statements can be proved.

2.3 Descendents of the λ-calculus

:TODO:

3 The basics

In our discussion of abstractions, we mentioned the abstraction of the function/method/procedure/subroutine.

- The functional abstraction provides a means to represent control flow.

In its pure version, every term in the λ-calculus is a function.

- In order for such a system to be at all useful, it must of course support higher-order functions; functions may be applied to functions.
- Values such as booleans and natural numbers are *encoded* (represented) by functions.

3.1 The terms

The pure untyped λ-calculus has just three sort of terms;

- variables such as x, y, z,
- λ-*abstractions*, of the form $\lambda x \cdot t$,
  - where x is a variable and t is a λ-term, and
- applications of the form tu
  - where t and u are λ-terms.

The meaning of each term is, informally:

- A λ-abstraction $\lambda x \cdot t$ represents a function of one argument, which, when applied to a term u, substitutes all occurrences of x with u.
- An application applies the term u to the function (term) t.
- A variable on its own (a free variable) has no further meaning.
  - Variables are intended to be *bound*.

3.2 Variable binding

:TODO:

4 The formal syntax and semantics of *UL*

4.1 A grammar for *UL*

$\langle$`term`$\rangle$ ::= `var` | λ `var` • $\langle$`term`$\rangle$ | $\langle$`term`$\rangle$ $\langle$`term`$\rangle$

In the case that we are restricted to ASCII characters, we will write abstraction as

`"lambda" var .` $\langle$`term`$\rangle$

5 α-conversion, β-reduction and η-conversion

:TODO:

6 Topics of theoretical interest

6.1 The pure λ-calculus

:TODO:

6.2 Nameless representation of terms

:TODO: