Types, Semantics, and Programming Languages (IK3620)

Exercises for Module 2
Typed lambda calculus with extensions

Version 1.0

David Broman
KTH Royal Institute of Technology
dbro@kth.se

August 29, 2016

Contents

Chapter 1

Introduction

Welcome to the second course module for the course *Types, Semantics, and Programming Languages (IK3620)*. In this module you will learn about type systems, the simply typed lambda calculus, and various extensions that are typically needed when creating a real programming language.

1.1 Submit your solutions

Please see the course website about deadlines and details of how to submit the solutions. `http://www.kth.se/social/group/ik3620/`

1.2 Resources

The main resource for module 2 is the TAPL book by Pierce [1]. Please read chapters 8 to 14 carefully.

Chapter 2

Simply Typed Lambda Calculus and Type Safety

2.1 Typed Arithmetic Expressions

Task

Consider the syntax and semantics for the typed arithmetic expression language, defined in TAPL on the pages 34, 41, and 93. For each of the terms below, answer the following questions:

1. What is the typing derivation of the term? Show the tree.
2. Is the term well typed?
3. What is the type of the term?
4. What is the final normalized value of the term when no more evaluation steps can be taken?
5. Is the normalized term stuck? Why, why not?
6. If the term is not well typed, discuss if this is due to that the type system is conservative. Also, for not well typed terms, propose a type rule that would make the term typable. Discuss if this is a good idea or not.

Terms:

1. `iszero (if true then pred 0 else succ 0)`
2. `iszero (pred true)`
3. `succ (succ (if iszero 0 then 0 else succ 0))`
4. `if iszero (pred 0) then true else succ 0`

2.2 Simply Typed Lambda Calculus

Task

Perform TAPL exercise 9.2.2 on page 103.

Task

Extend your big-step operational evaluator that has de Bruijn index (Module 1, Exercise 7.2) with the simply typed lambda calculus. The type checker must reject terms that are not well typed.

2.3 Type Safety

Task

Perform TAPL exercise 9.3.9 on page 107. You must write out the proof in detail, explaining every step, every lemma you use, all assumptions you use etc.

Hint: Go through the proofs in TAPL Chapter 8 and the proof of progress in Chapter 9 in detail before you try to solve the task above. You learn much more by writing out all the steps yourself than just reading the steps in the book.

2.4 Curry-Howard Correspondence

Task

Explain the Curry-Howard Correspondence in your own words. The description do not have to be long (max half a page), but should explain the key ideas for someone that is not familiar with the topic.

Chapter 3

Extensions for Creating Programming Languages

In the rest of the course, you should design your own programming language. You may extended it in any way you want, but the following tasks state what the language must *at least* include, but it can include more.

Note also that you should comment your source code according to common sound software engineering practice.

3.1 Simple Extensions

Task

You may add any extensions or variants (with different syntax) of the simple extensions that are described in TAPL Chapter 11. However, you must at least include the following extensions.

- Let bindings
- Tuples
- General recursion

3.2 Lists, References, and Exceptions

3.2.1 Task

Write down a pedagogical summary that explains the key ideas of the syntax and semantics for Lists, References, and Exceptions. The summary should be approximately one page for each of the three language features.

3.2.2 Task

Select and implement at least one of the language features: Lists, References, or Exceptions.

3.3 User Manual and Examples

Task

For your new language, you should provide a short user manual (a separate document) that explains the key functionalities of the language. The manual should include program code examples that demonstrate the different language features that you have implemented. Please add these examples both in the manual and in separate files that are directly executable. Do not forget to describe how the programs should be executed.

Bibliography

[1] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.