Types, Semantics, and Programming Languages (IK3620)

Exercises for Module 3
Subtyping and Polymorphism

Version 1.0

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

May 22, 2018

Contents

Chapter 1

Introduction

Welcome to the third course module for the course *Types, Semantics, and Programming Languages (IK3620)*. In this module you will learn about different forms of polymorphism.

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 3 is the TAPL book by Pierce [1], as well as 3 papers. Please see the course website for links.

Chapter 2

Polymorphism Overview

2.1 Explanation of concepts

Read the following:

1. Sections 1.1, 1.2, and 1.3 of the paper by Cardelli and Wegner.
2. Sections 19.3 and 23.2 in TAPL.

Task

Briefly answer each of the following questions:

1. What is a monomorphic language and why is it restrictive?
2. What does static type checking mean and what are the pros and cons compared to dynamic checking?
3. Explain briefly the concept of parametric polymorphism by giving a small example in a language of your choice.
4. How does ad-hoc polymorphism and universal polymorphism differ? Try to give examples in a language of your choice that illustrate the difference.
5. According to Cardelli and Wegner, there are two main categories of ad-hoc polymorphism. Explain the difference between these two forms.
6. What is the difference between nominal and structural type systems? Give examples of languages that use either of them and discuss some pros and cons.

Chapter 3

Subtype Polymorphism

3.1 Arrow type

The subtyping rule for arrow types (TAPL page 184) is reversed (contravariant) for argument types.

Task

Give an example that clearly demonstrates why covariant would be wrong for argument types. Give a detailed explanation.

3.2 Algorithmic Subtyping

The typing rules presented in TAPL chapter 15 are not syntax directed and it is therefore non trivial to write a type checker directly based on these rules. A solution to this problem is to translate the rules into rules for algorithmic subtyping, as presented in TAPL chapter 16.

Task

Explain and define an illustrative example term that shows how algorithmic subtyping is used with record types. As part of your explanation, include the type derivation tree of the term.

3.3 Implementing Algorithmic Subtyping

Task (conditionally optional)

Note: You may choose to do this exercise, or the optional exercise 4.2, or both. However, you must do at least one of them.

Extend your language from module 2 with:

1. record types, as presented in TAPL chapter 11.8
2. algorithmic subtyping, as presented in TAPL chapter 16.

Note that your language must also handle at least two base types, e.g., integers and booleans. In your presentation, show at least 3 test programs that illustrate the use of subtyping. The examples should both be presented in the PDF, and as program files in the repository. Please explain exactly how a user should execute these test programs.

Chapter 4

Parametric Polymorphism

4.1 System F

Read chapter 23.

Task

1. Write out the type derivation tree for the term `doubleNat`, as stated on page 345 in TAPL. Note that you should expand the term `double` before writing out the term. For readability, you may use abbreviations for certain terms, as long as you clearly define these definitions and that you show the complete derivation tree.
2. Solve TAPL exercise 23.4.4. Advice: start by first performing 23.4.3. If you get stuck, there is a solution for 23.4.3 in the book. As stated in the book, the exercise is best done online using the `fullomega` checker.

4.2 Implementing Parametric Polymorphism

Task (conditionally optional)

Note: You may choose to do this exercise, or the optional exercise 3.3, or both. However, you must do at least one of them.

Extend your language from module 2 with either let-polymorphism (TAPL chapter 22) or System F (TAPL chapter 23).

Note that your language must also handle at least two base types, e.g., integers and booleans. In your presentation, show at least 3 test programs that illustrate the use of parametric polymorphism. The examples should both be presented in the PDF, and as program files in the repository. Please explain exactly how a user should execute these test programs.

Chapter 5

Adhoc Polymorphism and Gradual Typing

5.1 Type Classes

Task

Read the paper by Wadler and Blott (1989). Explain in your own words the limitations of ad-hoc polymorphism, and the key ideas of type classes, as presented by the authors. Your explanation should be approximately one page long.

5.2 Gradual Typing and Frames

Task

Read the sections 1 and 2 of the paper by Siek and Taha (2006). Select a term (an expression) of your choice that demonstrates the idea of gradual typing. For this expression, show the type derivation using the typing rules presented in the paper (Figure 2). Also, explain clearly why this term is a good example that illustrates gradual typing.

Task

Read the paper by Broman and Siek (2018), with the focus on section 3. Explain the following in detail.

1. What are the technical differences between the inference rules described in Figures 6, 8, and 9? In particular, explain how the relations in Figures 6 and 8 differ from Figure 9 in how they treat terms. You do not need to explain the specific rules in detail, but rather give an overview of the idea behind the different relations.
2. The dynamic semantics in Figure 10 use frames instead of explicit congruence rules. Explain in detail the meaning of the frame concept, how it works, and propose a benefit compared to writing out the congruence rules explicitly.

Bibliography

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