Study Notes

Theory of Functional Programming

Emma Bach

May 22, 2025

Contents

Preface

These notes are not based on any particular lecture I attended. Instead, they were created during self-study of theoretical topics related to type theory, functional programming and the λ-calculus.

My primary sources are “Types and Programming Languages” by Benjamin C. Pierce [3] and the “Handbook of Logic in Computer Science”, edited by Samson Abramsky, Dov Gabbay and Thomas Maibaum, in particular volume 2, which deals with computational structures [2]. Additional sources I found helpful include the video “What is plus times plus?” by YouTube channel 2Swap, which is a wonderful introduction to the λ-calculus [1].

The first few chapters correspond to [3], which tends to use informal, intuitive definitions, while the latter are generally from [2], which is very formal and rigourous. These differences will be noticable in the notes too.

Chapter 1

Untyped Systems

1.1 Untyped Arithmetic Expressions

This chapter serves as an introduction of many concepts related to formal languages, such as abstract syntax, inductive definitions, evaluations, and the modeling of run-time errors.

In this first section, we will examine a simple language containing the following elements:

- the boolean constants `true` and `false`
- conditional expressions `if` t_1 `then` t_2 `else` t_3
- the numeric constant 0
- the arithmetic operators `succ` and `pred`
- a 1-ary function `iszero`

We summarize these contents by writing them as the following grammar:

t ::=
```
    true
    false
    if t then t else t
    0
    succ t
    pred t
    iszero t
```

The first line t ::= declares that we are defining a set of **terms**, and that we are using the letter t to refer to these terms. Each line that follows gives a possible **syntactic form** for terms. The letter t does not refer to any specific term - whenever t appears, we may substitute any term. A symbol like t is called a **metavariable**, the "meta" referring to the fact that it is not a variable within the language we are defining (also known as the **object language**), but rather in the notation that we are defining the language in (the **metalanguage**).

This "meta-" naming scheme comes from **metamathematics**, which is the field of logic that deals with the mathematical properties of formal systems of mathematical reasoning, such as programming languages. The collection of true statements we can make about such a system is known as the **metatheory** of the system.

A **program** in the given language is just a term belonging to it. If we additionally assign semantical meaning to the terms we defined, we can evaluate terms belonging to our system, by **reducing** them to another term belonging to the language. For example, the term `if false then 0 else 1;` evaluates to `1`, and the term `iszero(pred(succ(0)))` evaluates to `true`. Note that parentheses are added for readability, but often not included in the formal description of the language. Also note that, in our language, all natural numbers are formally represented as repeated applications of `succ` to `0`. However, for brevity, arabic numerals will be used to represent these terms. For example, whenever we use the term `2`, the equivalent term actually used in the language would be `succ(succ(0))`.

A more formal mathematical definition of our language would use induction to define the set $\mathcal{T}$ of all terms:

Definition 1.1.1. The set of terms is the smallest set $\mathcal{T}$ such that:

1. $\{\texttt{true}, \texttt{false}, \texttt{0}\} \subseteq \mathcal{T}$
2. if $t_1 \in \mathcal{T}$, then $\{\texttt{succ }t_1\texttt{, pred }t_1\texttt{, iszero }t_1\} \subseteq \mathcal{T}$
3. if $t_1, t_2, t_3 \in \mathcal{T}$, then $\texttt{if }t_1\texttt{ then }t_2\texttt{ else }t_3 \in \mathcal{T}$

Chapter 2

Formal Term Rewriting Systems

2.1 Abstract Reduction Systems

Definition 2.1.1.

1. An **abstract reduction system** is a structure $\mathcal{A} = \langle A, \{\rightarrow_\alpha\}_{\alpha \in I}\rangle$, consisting of a set of terms A and a set of binary relations $\rightarrow_\alpha \in A^2$, called the **reduction relations** or the **rewrite relations**. As shorthand, the relation $\rightarrow_\alpha$ is occasionally written simply as α. An abstract reduction system with only one relation $\rightarrow$ is also known as a **replacement system** or a **transformation system**.

2. If $a \rightarrow_\alpha b$, then b is known as a **one step (α-) reduct** of a, meaning that b can be created by applying the reduction $\rightarrow_\alpha$ to a once.

Note that we let $\rightarrow_\alpha$ be any arbitrary relation, without additional demands of transitivity, reflexivity, or anything else of the sort.

Example 2.1.2. As an example, we could consider a term rewriting system that reduces sums of natural numbers. To write this system as an abstract reduction system, we can define A and $\rightarrow$ as follows:

- Let A be the smallest set of terms such that:
  - Every natural number $n \in \mathbb{N}$ is a term.
  - If t_1 and t_2 are terms, then $t_1 + t_2$ is a term.
- We define the following reduction relations $\rightarrow_0$ and $\rightarrow_1$:
  - If n and m are natural numbers and $k = n + m$, then $n + m \rightarrow_0 k$
  - If $t_1 \rightarrow_0 k$, then $t_1 + t_2 \rightarrow_1 k + t_2$

We could now apply these reduction rules to reduce a sum:

$$10 + 5 + 2 + 7 \rightarrow_1 15 + 2 + 7 \rightarrow_1 17 + 7 \rightarrow_0 24$$

The term $15 + 2 + 7$ is a one step 1-reduct of the term $10 + 5 + 2 + 7$, while the term $17 + 7$ is a one-step 1-reduct of $15 + 2 + 7$ and therefore a two step 1-reduct of $10 + 5 + 2 + 7$.

Definition 2.1.3. Given a reduction relation $\rightarrow_\alpha$, we are often interested in investigating new relations created by extending $\rightarrow_\alpha$. Some relations that are of interest to us include:

- The **transitive reflexive closure** $\twoheadrightarrow_\alpha$, defined as the smallest transitive relexive relation containing $\rightarrow_\alpha$. To give a more explicit definition, we have $a \twoheadrightarrow_\alpha b$ iff:
  1. $a \rightarrow_\alpha b$, or
  2. $a = b$, or
  3. $\exists n : \exists c_1, \ldots, c_n : a \rightarrow_\alpha c_1 \rightarrow_\alpha \ldots \rightarrow_\alpha c_n \rightarrow_\alpha b$

  In most of the literature, this closure would be denoted as $\rightarrow_\alpha^*$. However, the book uses $\twoheadrightarrow_\alpha$ because it makes transitive diagrams involving the relation more legible.

- The **convertibility relation** $=_\alpha$ generated by $\rightarrow_\alpha$, defined as the smallest equivalence relation containing $\rightarrow_\alpha$. Once again, to give a more explicit definition:

  We have $a =_\alpha b$ iff. $\exists u_1, \ldots, u_n \in A$ such that $a = u_1$, $b = u_n$, and for each u_i, we have either $u_i \rightarrow_\alpha u_{i+1}$ or $u_{i+1} \rightarrow_\alpha u_i$.

  Note that even though $\twoheadrightarrow_\alpha$ is the smallest transitive reflexive relation that contains $\rightarrow_\alpha$ and $=_\alpha$ is the smallest equivalence (transitive, reflexive, and symmetric) relation that contains $\rightarrow_\alpha$, $=_\alpha$ is **not** equivalent to the symmetric closure of $\twoheadrightarrow_\alpha$, as can be seen in Example 2.1.4.

Example 2.1.4. Let $\rightarrow_\alpha = \{(a, c), (b, c), (c, d)\}$. Then:

1. $\twoheadrightarrow_\alpha = \{(a, a), (b, b), (c, c), (d, d), (a, c), (a, d), (b, c), (b, d), (c, d)\}$
2. The symmetric closure of $\twoheadrightarrow_\alpha$ would be $\twoheadrightarrow^s_\alpha = \{(a, a), (b, b), (c, c), (d, d), (a, c), (c, a), (a, d), (d, a), (b, c), (c, b), (b, d), (d, b), (c, d), (d, c)\}$. Note that this relation is no longer transitive, since it includes (a, c) and (c, b), but not (a, b).
3. $=_\alpha$ would include every single pair of these elements.

In short, if $a \rightarrow_\alpha c \leftarrow_\alpha b$, then we have $a =_\alpha b$, but not necessarily $a \twoheadrightarrow^s_\alpha b$.

Bibliography

[1] 2swap. What is plus times plus? https://www.youtube.com/watch?v=RcVA8Nj6HEo. Accessed: Spring 2025.

[2] S. Ambramsky, Dov M. Gabbay, and T.S.E.Maibaum, editors. *Handbook of Logic in Computer Science, Volume 2: Background: Computational Structures.* Oxford Science Publications, 1992.

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