Introduction to dependent type theory

Ali Caglayan

November 30, 2018

Contents

1 Type theory **1**
1.1 Syntax 1

1 Type theory

We will follow the structure of syntax outlined in Harper [2]. There are several reasons for this. Firstly, for example in Barendegt [1] we have notions of substituion left to the reader under the assumption that they can be fixed. Generally Bbarendegt's style is like this and even when there is much formalism, it is done in a way that we find perculiar. We will follow Harper as an upto date, reputable and modern source.

1.1 Syntax

We begin by outlining what exactly syntax is, and how to work with it. This will be important later on if we want to prove things about our "type theories" as we will essentially have good data structures to work with.

We will begin with the notion of an *abstract syntax tree*. Which can be what is informally known as syntax, thus formal statements about the syntax are refering to its manifestation as an abstract syntax tree.

Definition 1.1.1. Let $\mathcal{S}$ be a finite set, which we will call **sorts**. An element of $\mathcal{S}$ is called a **sort**.

A sort could be a term, a type, a kind or even an expression. It should be thought of an abstract notion of the kind of synctactic element we have. Examples will follow making this clear.

Definition 1.1.2. An **arity** is an element $((s_1, \ldots, s_n), s)$ of the set of **arities** $\mathcal{Q} := \mathcal{S}^\star \times \mathcal{S}$ where $\mathcal{S}^\star$ is the Kleene-star operation on the set $\mathcal{S}$ (a.k.a the free monoid on $\mathcal{S}$ or set of finite tuples of elements of $\mathcal{S}$). An arity is typically written as $(s_1, \ldots, s_n)s$.

Definition 1.1.3. Let $\mathcal{O} := \{\mathcal{O}_\alpha\}_{\alpha \in \mathcal{Q}}$ be an $\mathcal{Q}$-indexed (arity-indexed) family of (disjoint) sets of **operators** for each arity. An element $o \in \mathcal{O}_\alpha$ is called an **operator** of arity α. If o is an operator of arity $(s_1, \ldots, s_n)s$ then we say o has **sort** s and that o has n **arguments** of sorts $s_1, \ldots, s_n$ respectively.

References

[1] Barendregt, H., 2013. *Lambda calculus with types*, Perspectives in logic. Cambridge: Cambridge University Press.

[2] Harper, R., 2016. *Practical foundations for programming languages.* 2nd ed. Cambridge University Press.