Introduction to dependent type theory

Ali Caglayan

November 30, 2018

Contents

1 Type theory **1**
1.1 Introduction 1
1.2 Syntax 1

1 Type theory

1.1 Introduction

We will follow the structure of syntax outlined in Harper [3]. There are several reasons for this. Firstly, for example in Barendregt [1] we have notions of substitution left to the reader under the assumption that they can be fixed. Generally Barendregt's style is like this and even when there is much formalism, it is done in a way that we find peculiar. We will follow Harper as an up to date, reputable and modern source.

In other type theory books such as [2, 4] not much attention is given to the syntax. For example notions like substitution are defined directly on terms of a lambda calculus, which we disagree with wholeheartedly. We think it is important to have a well-defined and well-understood notion of syntax before defining any type theories for a few reasons, firstly the structures we use are well understood and studied by other computer scientists in areas such as compilers [REF], and secondly, it will give us good notions of variables, bound variables and substituion. Also, our chosen structure will allow us to derive the semantics of our type theories in a much easier way, whereas if we hadn't paid attention to syntax we would have trouble keeping things well defined.

1.2 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 syntax as we will essentially have good data structures to work with.

Definition 1.2.1 (Sorts)**.** 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 syntactic element we have. Examples will follow making this clear. Let us fix a set $\mathcal{S}$.

Definition 1.2.2 (Arities)**.** An **arity** is an element $((s_1, \dots, s_n), s)$ 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 $(s_1, \dots, s_n)s$.

Similarly to before, we fix a set $\mathcal{Q}$ of arities.

Definition 1.2.3 (Operators)**.** 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 $(s_1, \dots, s_n)s$ then we say o has **sort** s and that o has n **arguments** of sorts $s_1, \dots, s_n$ respectively.

Fix an $\mathcal{Q}$-indexed family $\mathcal{O}$ of sets of operators of each arity in $\mathcal{Q}$.

Definition 1.2.4 (Variables)**.** Let $\mathcal{X} := \{\mathcal{X}_s\}_{s \in \mathcal{S}}$ be an $\mathcal{S}$-indexed (sort-indexed) family of disjoint (finite?) sets $\mathcal{X}_s$ of **variables** of sort s. An element $x \in \mathcal{X}_s$ is called a **variable** x of **sort** s.

Definition 1.2.5 (Fresh variables)**.** We say that x is **fresh** for $\mathcal{X}$ if $x \notin \mathcal{X}_s$ for any sort $s \in \mathcal{S}$. Given an x and a sort $s \in \mathcal{S}$ we can form the family $\mathcal{X}, x$ of variables by adding x to $\mathcal{X}_s$.

Remark 1.2.6. The notation $\mathcal{X}, x$ is ambiguous because the sort s associated to x is not written. But this can be remedied by being clear from the context what the sort of x should be.

Definition 1.2.7 (Abstract syntax tree)**.** The family $\mathcal{A}[\mathcal{X}] = \{\mathcal{A}[\mathcal{X}]_s\}_{s \in \mathcal{S}}$ of **abstract syntax trees** (or asts), of **sort** s, is the smallest family satisfying the following properties:

1. A variable x of sort s is an ast of sort s: if $x \in X_s$, then $x \in \mathcal{A}[\mathcal{X}]_s$.
2. Operators combine asts: If o is an operator of arity $(s_1, \dots, s_n)s$, and if $a_1 \in \mathcal{A}[\mathcal{X}]_{s_1}, \dots, a_n \in \mathcal{A}[\mathcal{X}]_{s_n}$, then the ast $o(a_1; \dots; a_n) \in \mathcal{A}[\mathcal{X}]_s$.

[Draw a picture here]

Remark 1.2.8. The idea of a smallest family satisfying certain properties is that of structural induction. So another way to say this would be a family of sets inductively generated by the following constructors.

Remark 1.2.9. An ast can be thought of as a tree whose leaf nodes are variables and brach nodes are operators.

Remark 1.2.10. When we prove properties $\mathcal{P}(a)$ of an ast a we can do so by structural induction on the cases above.

Lemma 1.2.11. We have $\mathcal{A}[\mathcal{X}] \subseteq \mathcal{A}[\mathcal{Y}]$ if and only if $\mathcal{X} \subseteq \mathcal{Y}$.

References

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

[2] Crole, R.L., 1993. *Categories for types*. Cambridge: Cambridge University Press.

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

[4] Lambek, J., 1986. *Introduction to higher order categorical logic*, Cambridge studies in advanced mathematics ; 7. Cambridge: Cambridge University Press.