2 An Axiomatization of Terms

Terms are a complex data structure. The main complications are:

1. The choice of argument variables does not matter. For instance, $\lambda x.x = \lambda y.y$. To put it plain, we don't get what we see.
2. An application st can only be formed if the types of s and t match.

There are two complementary methods for giving a precise definition of terms:

1. The constructive approach: Give a construction of terms in set theory.[1]
2. The axiomatic approch: Give an axiomatization of terms in set theory.[2]

The real numbers are a good example for a mathematical data structure defined with the axiomatic approach. A well-known construction of the real numbers uses so called Dedekind cuts and was devised by Richard Dedekind around 1860. Another construction of the real numbers employs equivalence classes of rational Cauchy sequences.

To validate an axiomatization, at least one model (i.e., a construction satisying the axioms) has to be devised. The added value that comes with an axiomatization is the fact that it doesn't commit us to a particular model. Rather, an axiomatization gives us the freedom to work with different models, as long as they satisfy the axioms.

An axiomatization saves as basis for proofs. Proofs based on an axiomatization are automatically valid for all models of the axiomatization.

We will now develop an axiomatization of terms that we will use as the base for all proofs and algorithms to come. The axiomatization captures the syntactic aspects of terms. The semantic interpretation of terms will be defined on top of the axiomatization.

2.1 Types

We start with an axiomatization of types.

2.1.1 Axiomatization

We assume two sets

$$Sor \subseteq Ty$$

[1] Constructions are related to implementations in programming.
[2] Axiomatizations are related to abstract data types (ADTs) in programming.

whose elements we call **sorts** and **types**, respectively. Types that are not sorts are called **functional types**. Furthermore, we assume two functions:

$$F \in Ty \to Ty \to Ty \qquad \text{function type}$$
$$|\cdot| \in Ty \to \mathbb{N} \qquad \text{size}$$

and arrange the following notations:

$$C, D \in Sor \qquad \text{sorts}$$
$$S, T \in Ty \qquad \text{types}$$
$$S \to T \quad \rightsquigarrow \quad FST$$

The first and the second notational line introduce so-called **meta-variables**. The first notational line states that the meta-variables C, D will always denote sorts, if not said otherwise. Similiarly, the second notational line states that the meta-variables S, T will always denote types. Finaly, the last line says that we may write $S \to T$ in place of FST.

We assume that types satisfy the following axioms (i.e., basic properties):

Par For every type T, exactly one of the following conditions is satisfied:
(1) $T \in Sor$ (2) $\exists S, S' \colon\ T = S \to S'$

CS $|C| = 1$

CF $|S \to T| = 1 + |S| + |T|$

IF $S \to T = S' \to T' \iff S = S' \wedge T = T'$

Inf $Sor \cong \mathbb{N}$

Axiom Par says that a type is either basic or functional, and that there is no type that is both basic and functional. The axiom CF says that every functional types can be obtained in finitely many steps from sorts. Together, Par and CF make it possible to prove claims about types by induction on $|T|$. Axioms IF says that two functional types are equal if both their argument types and their result types are equal. The final axiom Inf says that we may think of sorts as natural numbers. Formally, $X \cong Y$ means that there is a bijection between the set X and the set Y.

We insist on infinitly many sorts since we see sorts as names that may or may not be given a fixed meaning by an interpretation. Interpretations capture the semantic aspects of terms and will be defined later.

Every axiom holds for all possible values of the unquantified meta-variables occurring in it (so-called **universal interpretation**). For instance, the axiom CS holds for all sorts C, and the axiom CF holds for all types S and T. Written explicitly, axioms CS and CF look as follows:

CS $\forall C \in Sor\colon\ |C| = 1$

CF $\forall S, T \in Ty\colon\ |S \to T| = 1 + |S| + |T|$

2.1.2 Standard Model

We obtain a model of the axiomatization of types by defining the sets Sor and Ty as follows:

$$\begin{aligned} Sor &:= \{1\} \times \mathbb{N} \\ Ty &:= Sor \cup (\{2\} \times (Ty \times Ty)) \end{aligned}$$

The equation for Ty is to be interpreted recursively, that is, Ty contains only those pairs that can be obtained in finitely many steps from Sor. The definition of F and $|\cdot|$ is now straightforward:

$$\begin{aligned} FST &:= (2, (S, T)) \\ |(1, n)| &:= 1 \\ |(2, (S, T))| &:= 1 + |S| + |T| \end{aligned}$$

Note that the definition of $|\cdot|$ is recursive since the definition of Ty is recursive.

2.2 Terms

We continue with an axiomatization of terms.

2.2.1 Axiomatization

The axiomatization of terms assumes three sets

$$Var \subseteq Ind \subseteq Ter$$

whose elements are called **variables**, **individual names** and **terms**, respectively. Individual names that are not variables are called **constants**. We assume

$$Ty \cap Ter = \emptyset$$

and arrange the following notations:

$x, y, z \in Var$			variables
$u, v \in Ind$			individual names
$s, t \in Ter$			terms
Con	$\rightsquigarrow$	$Ind - Var$	constants
$a, b, c \in Con$			constants
Nam	$\rightsquigarrow$	$Sor \cup Ind$	names

We assume the following functions

$A \in Ter \to Ter \to Ter$	application
$L \in Var \to Ter \to Ter$	abstraction
$\tau \in Ter \to Ty$	type
$\lvert\cdot\rvert \in Ter \to \mathbb{N}$	size
$\mathcal{N} \in (Ty \cup Ter) \to \mathcal{P}(Nam)$	names
$\mathbf{S} \in Sub \to Ter \to Ter$	substitution

and arrange the following notations:

st	$\rightsquigarrow$	Ast	
$\lambda x.t$	$\rightsquigarrow$	Lxt	
$t{:}T$	$\rightsquigarrow$	$\tau t = T$	
Sub	$\rightsquigarrow$	$\{ f \in Ind \to Ter \mid \forall u\colon\ \tau(fu) = \tau u \}$	**substitutions**
$\theta \in Sub$			
$s[x := t]$	$\rightsquigarrow$	$\mathbf{S}\ (\lambda y \in Var.\ \text{if } x = y \text{ then } t \text{ else } y)\ s$	

The functions A and L provide for the construction of applications and abstractions. Note that A is a partial function. This reflects the fact that applicative terms can only be formed if a type constraint is satisfied. The function τ yields the type of a term. The existence of τ formalizes the assumption that every term has exactly one type and that individual names come with a built-in type. The size function provides for the axiomatization of the fact that every term can be constructed in finitely many steps from individual names. The function $\mathcal{N}$ yields all names that occur in a term. The function $\mathbf{S}$ provides for substitution.

Here are the axioms for terms:

DA1 $Dom\ A = \{ t \mid \tau t \text{ functional} \}$

DA2 $Dom\ (At) = \{ s \mid \exists T\colon\ \tau t = \tau s \to T \}$

Par For every term t, exactly one of the following conditions is satisfied:
(1) $t \in Ind$ (2) $\exists s, s'\colon\ t = ss'$ (3) $\exists x, s\colon\ t = \lambda x.s$

IA $st = s't' \iff s = s' \wedge t = t'$

IL $\lambda x.t = \lambda x'.t' \iff x' \notin \mathcal{N}(\lambda x.t) \wedge t' = t[x := x']$

TA $\tau t = \tau s \to \tau(ts)$

TL $\tau(\lambda x.t) = \tau x \to \tau t$

CN $|u| = 1$

CA $|ts| = 1 + |t| + |s|$

CL $|\lambda x.t| = 1 + |t|$

NS $\mathcal{N}C = \{C\}$

NF $\mathcal{N}(S \to T) = \mathcal{N}S \cup \mathcal{N}T$

NN $\mathcal{N}u = \{u\}$

NA $\mathcal{N}(ts) = \mathcal{N}t \cup \mathcal{N}s$

NL $\mathcal{N}(\lambda x.t) = \mathcal{N}(\tau x) \cup (\mathcal{N}t - \{x\})$

SN $\mathbf{S}\theta u = \theta u$

SA $\mathbf{S}\theta(ts) = (\mathbf{S}\theta t)(\mathbf{S}\theta s)$

SL $(\forall u \in \mathcal{N}(\lambda x.t)\colon\ x \notin \mathcal{N}(\theta u)) \implies \mathbf{S}\theta(\lambda x.t) = \lambda x.\mathbf{S}(\theta[x := x])t$

Inf $\forall T\colon\ \{\, x \mid \tau x = T \,\} \cong \mathbb{N} \ \wedge\ \{\, c \mid \tau c = T \,\} \cong \mathbb{N}$

Most of the axioms are obvious given an intuitive understanding of terms. We will say more about the axioms IL, SL and Inf.

IL states under which conditions two descriptions $\lambda x.t$ and $\lambda x'.t'$ yield the same term. Read from right to left, IL makes precise which variables can be used as argument variable in the description of an abstraction, and how the choice of the argument variable affects the body of the description.

Inf states that there are infinitly many variables and constants for every type.

SL states how substitution applies to abstractions. As we will see, the precondition of SL can always be satisfied by choosing a suitable argument variable.

If an axiom contains an application of the partial function A or $\mathbf{S}$, the axiom holds only for those values of the meta-variables that make the applications of A and S well-defined. For instance, the axiom TA holds only for those terms s and t such that t is functional and $s \in \mathit{Dom}\,(At)$. Written explicitly, axioms

TA $\tau t = \tau s \to \tau(ts)$

TL $\tau(\lambda x.t) = \tau x \to \tau t$

look as follows:

TA $\forall s, t \in \mathit{Ter}\colon\ t \in \mathit{Dom}\,A \ \wedge\ s \in \mathit{Dom}\,(At) \implies \tau t = F(\tau s)(\tau(Ats))$

TL $\forall x \in \mathit{Var}\ \forall t \in \mathit{Ter}\colon\ \tau(Lxt) = F(\tau x)(\tau t)$

Note that the meta-variable x used in the compact formulation of TL is not bound by the preceeding λ. Make sure you understand why this is the case.

2.2.2 Standard Model

The standard model for terms extends the standard model for types. The definition of variables, constants and individual names is easy:

$$\begin{aligned} Con &:= \{3\} \times (\mathbb{N} \times Ty) \\ Var &:= \{4\} \times (\mathbb{N} \times Ty) \\ Ind &:= Con \cup Var \end{aligned}$$

The definition of the set Ter requires more ingenuity. We will first define recursively a larger set whose elements we call **quasi-terms**:

$$QT := Ind \cup (\{5\} \times (QT \times QT)) \cup (\{6\} \times (Ty \times QT)) \cup (\{7\} \times \mathbb{N})$$

Applications are modeled as pairs starting with 5, abstractions as pairs starting with 6, and argument references (de Bruijn indices) as pairs starting with 7.

Let X be a set. We use X^* denote the set of all tuples whose components are in X. The elements of X^* may be thought of as strings or vectors. We use the following notations for $v \in X^*$:

- $v.n$ denotes the $(n+1)$-th component of v. For instance, $\langle 3,7,4\rangle.1 = 7$.
- $x :: v$ denotes the tuple obtained from v by adding x as leftmost component. For instance, $9 :: \langle 3,7,4\rangle = \langle 9,3,7,4\rangle$.

To define the set $Ter \subseteq QT$ of terms, we will first define an **admissibiliy relation** $R \subseteq Ty^* \times QT \times Ty$. Given R, we define the set of terms as follows:

$$Ter := \{\, t \in QT \mid \exists T\colon\ (\langle\rangle, t, T) \in R \,\}$$

Intuitively, this definition may be read as follows: a quasi-term t is a term if it doesn't contain dangling de Bruijn indices and if it is well-typed.

For the definition of R we introduce the following notations:

$$\begin{aligned} st &\rightsquigarrow (5,(s,t)) \\ \lambda T.t &\rightsquigarrow (6,(T,t)) \end{aligned}$$

We define R recursively by the following inference rules, where we assume $\Gamma \in Ty^*$, $n \in \mathbb{N}$, $S,T \in Ty$, and $s,t \in Ter$.

$$\frac{c = (3,(n,T))}{(\Gamma,c,T) \in R} \qquad \frac{x = (4,(n,T))}{(\Gamma,x,T) \in R} \qquad \frac{i = (7,n) \quad \Gamma.n = T}{(\Gamma,i,T) \in R}$$

$$\frac{(\Gamma,\ t,\ S \to T) \in R \quad (\Gamma,\ s,\ S) \in R}{(\Gamma,\ ts,\ T) \in R} \qquad \frac{(S :: \Gamma,\ t,\ T) \in R}{(\Gamma,\ \lambda S.t,\ S \to T) \in R}$$

It remains to define the functions A, L, τ, $|\cdot|$, $\mathcal{N}$, and $\mathbf{S}$. The definition of A, τ, $|\cdot|$, and $\mathcal{N}$ is straightforward, while L and $\mathbf{S}$ require more ingenuity. It also remains to verify that the axioms are satisfied, which in some cases is tedious.

2.3 Proof of Basic Properties

Based on the axioms, we can now prove properties that are satisfied by every model of axiomatization of terms.

In the following, we assume that some model is given. Since we don't make any special assumptions about this model, the following propositions will hold for all models.

Proposition 2.1 $\mathcal{N}T$ is a finite set.

Proof Exercise. ∎

Proposition 2.2 $\mathcal{N}t$ is a finite set.

Proof By induction on $|t|$. Case analysis according to Par.

Case $t = u$. Then $\mathcal{N}t = \{u\}$ by NN.

Case $t = ss'$. Then $\mathcal{N}t = \mathcal{N}s \cup \mathcal{N}s'$ by NA. Hence $\mathcal{N}t$ finite by induction hypothesis.

Case $t = \lambda x.s$. Then $\mathcal{N}t = \mathcal{N}(\tau x) \cup (\mathcal{N}s - \{x\})$ by NL. Hence $\mathcal{N}t$ finite by Proposition 2.1 and induction hypothesis. ∎

Proposition 2.3 (Type Preservation) $\tau(\mathbf{S}\theta t) = \tau t$

Proof By induction on $|t|$. Case analysis according to Par.

Case $t = u$. Then $\tau(\mathbf{S}\theta t) = \tau(\theta u) = \tau u = \tau t$ by SN.

Case $t = ss'$. Then

$\mathbf{S}\theta t = (\mathbf{S}\theta s)(\mathbf{S}\theta s')$	SA
$\tau(\mathbf{S}\theta s) = \tau(\mathbf{S}\theta s') \to \tau(\mathbf{S}\theta t)$	TA
$\tau s = \tau s' \to \tau(\mathbf{S}\theta t)$	induction hypothesis

Since $\tau s = \tau s' \to \tau t$ by TA, we have $\tau(\mathbf{S}\theta t) = \tau t$ by IF.

Case $t = \lambda x.s$. By Proposition 2.2, Inf and IL we can assume $x \notin \mathcal{N}(\theta u)$ for all $u \in \mathcal{N}t$. Hence

$\tau(\mathbf{S}\theta t)$	$= \tau(\lambda x.\mathbf{S}(\theta[x := x])s)$	SL
	$= \tau x \to \tau(\mathbf{S}(\theta[x := x])s)$	TL
	$= \tau x \to \tau s$	induction hypothesis
	$= \tau t$	TL

∎

Proposition 2.4 (Coincidence) $(\forall u \in \mathcal{N}t\colon\ \theta u = \theta' u) \implies \mathbf{S}\theta t = \mathbf{S}\theta' t$

Proof Exercise. ∎

Proposition 2.5 $\mathbf{S}(\lambda u{\in}Ind.u)t = t$

Proof Exercise. ∎

2.4 Renaming

Proposition 2.6 (Renaming) $y \notin \mathcal{N}t \implies (t[x{:=}y])[y{:=}s] = t[x{:=}s]$

Proposition 2.7 $v \in \mathcal{N}(\mathbf{S}\theta t) \iff (\exists u \in \mathcal{N}t\colon\ v \in \mathcal{N}(\mathbf{S}\theta u))$

Proposition 2.8 $t' = t[x{:=}x'] \implies (t = t'[x'{:=}x] \iff x' \notin \mathcal{N}(\lambda x.t))$

Proposition 2.9 $\lambda x.t = \lambda x'.t' \iff t = t'[x'{:=}x] \wedge t' = t[x{:=}x']$

2.5 Beta and Phi

We fix a function

$$\beta \in Ter \to Ter \to Ter$$

such that

$$\beta(\lambda x.t)s = t[x{:=}s]$$

for every variable x and all terms s, t such that $\tau x = \tau s$. The existence of β follows from IL and Proposition 2.6.

Exercise 2.10 Explain why the existence of β is not obvious.

Proposition 2.11 (Beta) For every abstraction $t : S \to T$ and every variable $x : S$, the following equivalence holds:

$$t = \lambda x.s \iff x \notin \mathcal{N}t \wedge s = \beta t x$$

Proof Let $t : S \to T$ be an abstraction and $x{:}S$ be a variable. Since t is an abstraction, there exist x' and s' such that $t = \lambda x'.s'$. Since $\beta t x = s'[x'{:=}x]$ it remains to show that

$$\lambda x'.s' = \lambda x.s \iff x \notin \mathcal{N}t \wedge s = s'[x'{:=}x]$$

for ever term s. This property is an instance of Axiom IL. ∎

There are exactly as many ways to describe an abstraction $t : S \to T$ with λ as there are variables $x : S$ such that $x \notin \mathcal{N}t$. Often it is convenient to have a fixed λ-description for every abstraction. To this purpose we fix a function

$$\varphi \in Ter \to Var$$

such that $\varphi t \notin \mathcal{N}t$ and $\varphi t : S$ for every abstraction $t : S \to T$. The existence of such a function follows from Proposition 2.2 and Inf.

Proposition 2.12 (Unique Decomposition) Let t be an abstraction and $x = \varphi t$. Then there exists one and only one term s such that $t = \lambda x.s$.