A.1 The first presentation

Canonical name	A1TheFirstPresentation
Date of creation	2013-11-09 4:41:54
Last modified on	2013-11-09 4:41:54
Owner	PMBookProject (1000683)
Last modified by	PMBookProject (1000683)
Numerical id	4
Author	PMBookProject (1000683)
Entry type	Feature
Classification	msc 03B15

The objects and types of our type theory may be written as terms using the following syntax, which is an extension of λ-calculus with *variables* $x, x', \ldots$, *primitive constants* $c, c', \ldots$, *defined constants* $f, f', \ldots$, and term forming operations

$$t \coloneqq x \mid \lambda x.\, t \mid t(t') \mid c \mid f$$

The notation used here means that a term t is either a variable x, or it has the form $\lambda x.\, t$ where x is a variable and t is a term, or it has the form $t(t')$ where t and t' are terms, or it is a primitive constant c, or it is a defined constant f. The syntactic markers 'λ', '(', ')', and '.' are punctuation for guiding the human eye.

We use $t(t_1, \ldots, t_n)$ as an abbreviation for the repeated application $t(t_1)(t_2)\ldots(t_n)$. We may also use *infix* notation, writing $t_1 \star t_2$ for $\star(t_1, t_2)$ when $\star$ is a primitive or defined constant.

Each defined constant has zero, one or more **defining equations**. There are two kinds of defined constant. An *explicit* defined constant f has a single defining equation

$$f(x_1, \ldots, x_n) \equiv t,$$

where t does not involve f. For example, we might introduce the explicit defined constant $\circ$ with defining equation

$$\circ(x, y)(z) \equiv x(y(z)),$$

and use infix notation $x \circ y$ for $\circ(x, y)$. This of course is just composition of functions.

The second kind of defined constant is used to specify a (parameterized) mapping $f(x_1, \ldots, x_n, x)$, where x ranges over a type whose elements are generated by zero or more primitive constants. For each such primitive constant c there is a defining equation of the form

$$f(x_1, \ldots, x_n, c(y_1, \ldots, y_m)) \equiv t,$$

where f may occur in t, but only in such a way that it is clear that the equations determine a totally defined function. The paradigm examples of such defined functions are the functions defined by primitive recursion on the natural numbers. We may call this kind of definition of a function a *total recursive definition*. In computer science and logic this kind of definition of a function on a recursive data type has been called a **definition by structural recursion**.

Convertibility $t \downarrow t'$ between terms t and t' is the equivalence relation generated by the defining equations for constants, the computation rule

$$(\lambda x.\, t)(u) :\equiv t[u/x],$$

and the rules which make it a *congruence* with respect to application and λ-abstraction:

- if $t \downarrow t'$ and $s \downarrow s'$ then $t(s) \downarrow t'(s')$, and
- if $t \downarrow t'$ then $(\lambda x.\, t) \downarrow (\lambda x.\, t')$.

The equality judgment $t \equiv u : A$ is then derived by the following single rule:

- if $t : A$, $u : A$, and $t \downarrow u$, then $t \equiv u : A$.

Judgmental equality is an equivalence relation.