Applications of Type Theory

Bernd Mahr

Fachbereich Informatik
Technische Universität Berlin

Abstract. The paper discusses the setting of type theory and proposes a uniform framework for disciplines of declarations and types. The framework combines fundamental concepts found in the setting of type theory and supports both its denotational and its constructive view. In the denotational view ϵ -structures are introduced as semantic models and in the constructive view the general forms of judgements and rules are given. Finally applications are discussed which build on the ideas of this framework, namely models of untyped λ -calculus which truly solve the equation of reflexive domains, and $\epsilon\mathcal{T}$ -logic, a first-order theory of propositions with selfreference and impredicative quantification allowing for intensional models of truth.

1 Introduction

Types and typing not only apply in the foundations of mathematics (see Russell [Ru 08] and Martin-Löf [ML 84]) and the study of functions and propositions (see Church [Ch 40], Curry and Feys [CF 58], Girard [Gi 71] and Howard [Ho 69]), but, equally important, also in correctness of programs (see Reynolds [Re 74] and Milner [Mi 78]), formal specification and programming (see Martin-Löf [ML 80] and Nordström [No 81], [NPS 90]), terminological modelling (see [BHR 90] for an overview) and in the constructive foundation of functional programming (see Thompson [Th 91] for example).

The notion of type has no rigid definition. It is intentional by nature and has a fixed meaning only in the context of a given discipline, formalism or system. As such, types may be indices, sorts, categories, kinds, sets, domains, propositions, specifications and the like. Types always indicate some form of classification, and typing - the attachment of types to entities - provides type information that can be used for control in building expressions and formation of concepts, for verification of well-formedness and consistency, in specification and in the declaration of ontologies and truth.

Though types and typing have various forms and uses, there is a common underlying setting which we call the *setting of type theory*. In the following we discuss this setting and applications in it, and derive a unified *framework for declaration and type disciplines*.

2 Type Propositions

The origins of type theory date back to ancient philosophy and to the still vivid question whether all objects are of one kind or not, i.e. that "there are properties which can be meaningfully predicated of certain objects but not of others" (see [FBL 84] p 188 ff). Though there are many formalizations and formalisms based on concepts like kind, class, category or type (as geometry distinguishes points, lines and planes, and logic distinguishes orders of predication), untyped languages are still in favor for their orthogonality and self-similarity properties (like LISP and some object oriented languages for example).

Russell used typing in [Ru 08] to avoid impredicative set formation (i.e. the set to be defined appears in the range of a quantifier that is used to define that set). By attaching indices to variables which indicate a certain level in the hierarchy of sets and by restricting membership and quantification accordingly, he ruled out antinomies that were found in naive set theory. As a consequence the paradoxical, $S = \{x \mid x \notin x\}$ which satisfies $S \in S \iff S \notin S$ and therefore contradicts Cantors conception that it be determined whether or not an object is a member of a given set, can no longer be expressed.

Other approaches to solve the foundational problems of mathematics have later made use of this idea of hierarchical types. For its naturalness Church introduced in [Ch 40] simply typed λ -calculus with typed variables, abstractions and applications

$$\begin{array}{c} x^a \\ (E^a \longrightarrow b F^a)^b \\ (\lambda x^a E^b)^a \longrightarrow b \end{array}$$

for any type expressions of the form a , b and $a \rightarrow b$.

With similar motivations Herbrand had earlier introduced sorts into first order logic [He 30], which was later extended to subsorts, and was adopted in universal algebra by Higgins [Hi 63], leading to the notion of heterogeneous or many-sorted algebra, which focus the basic concept of algebraic specification (see [EM 85] for example).

Common in these cases of introducing types into a formalism is, that formation of expressions becomes restricted by type constraints. An expression in the typed formalism therefore contains type information, be it explicitly stated or implicit in the property of well-formedness. This type information in expressions creates a fragment of their meaning.

However, different in these cases is the effect the introduction of types has on the formalism's expressive power: Untyped first order logic is equivalent to many-sorted first order logic, which was already shown by Herbrand. Many-sorted algebra with equational specifications is more powerful than unsorted algebra. Russell's Type Theory is obviously restricting naive set theory,

and so is simply typed λ -calculus less powerful than untyped λ -calculus (it actually forms a decidable theory of total functions that have at most superexponential complexity).

Fundamental in the setting of type theory is the notion of *type proposition*. A type proposition provides type information and is written uniformly as

$$a : A$$

with a and A being *expressions*. In the locality of a given type proposition we call the expression a an *entity* and the expression A its *type*.

A type proposition is a proposition and therefore may be found or declared to be true or false. A type proposition has one or more *readings*. These readings articulate certain *interpretations* which are substantiated either by the *rules of a type system* in which the particular type proposition is admissible, or by the meaning of the type proposition in some *model*. Both ways, which actually belong to different views on formalization and mathematics, give semantics to the formal expression of a type proposition.

For example:

In the type proposition

$$5 : \text{nat}$$

5 may denote an element of the set of natural numbers denoted by nat . This type proposition makes an assertion about 5 (rather than nat) and has an obvious modeltheoretic interpretation.

In the type proposition

$$f : A \rightarrow B$$

A and B may denote sets and $A \rightarrow B$ may denote the set of functions from the set denoted by A to the set denoted by B . Again there is an obvious modeltheoretic interpretation. Another reading could be to regard A and B as propositions and $A \rightarrow B$ as an implication in intuitionistic logic. Then f represents a proof for $A \rightarrow B$ which is a method to construct from every proof for A a proof for B . While this reading does not have an obvious appropriate modeltheoretic interpretation, it is typical for type systems based on the 'Curry-Howard-Isomorphism', which stands for the concept of 'propositions as types', that these systems support with their rules both readings: $A \rightarrow B$ denoting the function space and $A \rightarrow B$ expressing a proposition.

Similarly with the type proposition

$$(w, p) : (\exists x : A. P(x))$$

which can be regarded as the assertion that the type $\exists x:A.P(x)$ which is the sum of types $P(x)$, indexed over the entities of A is inhabited and therefore exists, and at the same time as the assertion that the proposition $\exists x:A.P(x)$ is provable and has the pair (w, p) consisting of a witness w for the quantifier and a proof p for $P(w)$ as a proof (see [Th 91] for example).

Finally, in the type proposition

$$a : \text{true}$$

the focus is on the attachment of the type `true` to the entity a . A possible reading is to say that a denotes a proposition which is a member of the set of all true propositions denoted by `true`. This reading can indeed be supported by an appropriate modeltheoretic interpretation (see [Str 92]).

3 The Constructive View

Constructive mathematics and intuitionism have a major influence in type theory. They constitute, as we call it, *the constructive view* in the setting of type theory. The meaning of type propositions under this constructive view is declared by the rules of a type system, which determine what type propositions exist and how they are to be handled.

Simply typed λ -calculus is a first and good example. Moreover, it is the starting point for a great number of systems which have been designed for various purposes: to obtain a powerful theory of total functions which is decidable, to formalize type inference in programming, to support the reading of 'propositions as types' in a constructive theory of propositions, or to provide a constructive foundation to mathematics, that is consistent. Among these systems is the system F by Girard [Gi 71] which extends simply typed λ -calculus to second order typed λ -calculus allowing quantifications over type variables. A similar system was independently invented by Reynolds to "permit the definition of polymorphic procedures" in typed programming languages (see [Re 74]). Other systems are intuitionistic type theory by Martin-Löf (see [ML 71] and [ML 84]) and the calculus of constructions by Coquand and Huet (see [Co 85] and [CH 88]).

Most of these systems are built under the paradigm of 'proposition as types' and therefore not only support the dual reading of type propositions, but can also be seen as constructive theories of propositions. Understanding propositions as types was first exhibited by Curry and Howard (see [CF 58] and [Ho 69]) and then developed further by many authors (see Barendregt [Ba 90] for a summary of results). The idea of propositions as types originates in the similarities between the building-rules for typed λ -abstraction and application in simply typed λ -calculus on one side, and the rules for introduction and elimination of $\rightarrow$ in natural deduction systems of propositional logic (see Gentzen [Ge 69]) on the other.

The setting of type theory under the constructive view is based on rules and calculi which control the building of expressions and type propositions (*construction*), which determine what type propositions can be stated (*declaration*), and what type propositions can be inferred from others (*inference*). Rules, in this context, are sets of *judgements* (see [ML 84]) in which one judgement is distinguished (*conclusion*) from the others (*premisses*). The judgements of a rule usually have a certain *form* that is characteristic for the system in which this rule occurs.

Fundamental questions in the setting of type theory under the constructive view are consistency, existence of principal types, and normalization properties, which concern the expressive power of a system. Still under discussion is the question, whether or not *impredicativity* in type propositions should be admissible. While Russell's type theory and simply typed λ -calculus excluded impredicativity, second order typed λ -calculus, Martin-Löf's first system, and the calculus of constructions admitted impredicativity by quantification over propositions. In combination with a universal type 'type' (which is inhabited by itself, i.e. $\text{type}:\text{type}$ is true) Martin-Löf's first system is inconsistent. In intuitionistic type theory impredicativity is therefore excluded. On the other hand, it seems meaningful to allow for impredicativity in the propositional fragment of a type system, since it is not by itself a source for inconsistency and occurs naturally if not banned (see also Luo [Luo 90]).

4 The Denotational View

Tarski's perception of semantics is to provide meaning to syntactic objects by giving models in which these objects can be interpreted. Since type propositions are expressed by syntactic objects, a modeltheoretic interpretation requires appropriate mathematical structures which allow to capture their underlying intention. While this is obvious in the case of the simply typed λ -calculus, for example, in which every expression can be interpreted as a total function of a kind expressed in the expression's type, it is less trivial for other type systems and may even lead back to the foundational problems of mathematics in which circularity was discarded for reasons of consistency. Furthermore, the question of what we want to accept as a model of a type system has not clearly been answered yet. It is discussed in [Mey 82] for the untyped λ -calculus, but has been revised again for type calculi (see the discussion of this topic in the various contributions of [Hue 90]).

Fundamental in the setting of type theory under the denotational view is *the interpretation of a type proposition as a certain relationship* that holds between the interpretation of its entity on one side and of its type expression on the other i.e. the equation

$$\llbracket a:A \rrbracket = \llbracket a \rrbracket \llbracket : \rrbracket \llbracket A \rrbracket$$

The actual nature of $\llbracket a \rrbracket$, $\llbracket A \rrbracket$ and $\llbracket : \rrbracket$ is determined by the model in which these items are defined. $\llbracket : \rrbracket$ is in any case a binary relationship that is usually uniformly understood as some kind of "*membership*". The relations between the expression a and the object $\llbracket a \rrbracket$ and between A and

$\llbracket A \rrbracket$ depends, at least in most of the familiar uses on the structure of a and A . This dependency can appropriately be expressed by "*denotational constraints*" which formulate the conditions to hold for $\llbracket a \rrbracket$ and $\llbracket A \rrbracket$ as interpretations of a and A respectively.

For example, the expression $f(a)$ which is obtained by application of the function symbol f to the argument expression a , has the denotational constraint $C(f(a))$ which requires that

" $\llbracket f(a) \rrbracket$ is the result of applying the function $\llbracket f \rrbracket$ to the argument $\llbracket a \rrbracket$ "

In many formalisms with modeltheoretic semantics, like algebra or logic for example, these denotational constraints are implicit in the architecture of the formalism (signature, term structure, algebraic structure, evaluation homomorphism).

5 A Framework for Declaration and Type Disciplines under the Denotational View

The components we have identified as fundamental in the setting of type theory can now be combined into a uniform framework. This framework provides under the denotational view general concepts for model theoretic semantics of type propositions and declarations, and fixes under the constructive view the forms of rules which control construction, declaration and inference of type propositions.

The focus of this framework is on the notion of type proposition and declaration, where declarations are sets of type propositions. Following the basic observation that modeltheoretic semantics has to provide an interpretation of type propositions, the framework avoids to use signatures as classifiers for structures and instead models the attachment of a type proposition; namely, if $a:A$ is a type proposition, $\llbracket a:A \rrbracket$ represents a pair

$$(\llbracket a \rrbracket, \llbracket A \rrbracket)$$

of objects from the domain in which expressions are evaluated, where the evaluation of expressions obeys the denotational constraints associated with the expressions. Signature information, however, is not discarded, but is coded in type propositions and denotational constraints like any other informations that appears in a declaration.

More formally we define the following:

An ϵ -*structure* $M = (M, \epsilon)$ consists of a nonempty set M of objects and a binary relation on M .

By ϵ -*logic* we refer to classical first order logic with equality and a single binary predicate ϵ .

Atomic ϵ -formulas are of the form $x = y$ or $x \epsilon y$ for variables x and y (note, this logic is

which can be interpreted as declaring a self-applicable identity function id . Assume, that the discipline D implies the following denotational constraints:

$C(A \longrightarrow A)$	$\equiv$	" $A \longrightarrow A$ is the set of all functions from A to A "
$C(\text{true})$	$\equiv$	"the only member of true is T "
$C(\text{id}(x))$	$\equiv$	" $\text{id}(x)$ is the unique object y such that $(x, y) \in \text{id}$ "
$C(x = y)$	$\equiv$	" x and y are equal"

and for all other expressions in Δ , namely x , id , and A , the denotational constraints are void, i.e. of the form $x = x$ for some variable x (note, that these constraints can be expressed in $\in$ -logic). Then the following $\in$ -structure $(M, \in)$, represented by the graph

with $\llbracket \cdot \rrbracket$ assigning

$\llbracket \text{id} \rrbracket$	$= 1$
$\llbracket A \rrbracket$	$= a$
$\llbracket A \longrightarrow A \rrbracket$	$= a$
$\llbracket x \rrbracket$	$= 1$
$\llbracket \text{id}(x) = x \rrbracket$	$= T$
$\llbracket \text{TRUE} \rrbracket$	$= \text{true}$

is a model of Δ , in which 1 is indeed the selfapplicable identity, seen as a set of pairs, with pairs coded after Kuratowski (see [MSU 90]).

$\in$ -structures form a theory of non-wellfounded sets similar to the one given by Aczel in [Acz 88]. A major difference, however, is the fact that objects in an $\in$ -structure, which are regarded as sets, are not identified with their extension, but instead have an extension only associated. Accordingly, the axiom of extension is not valid, if not explicitly required, and the possibility of circles is not ruled out. The exact relationship between $\in$ -sets and Aczel-sets has been studied by Sträter [Str 92a] where it is shown that, roughly speaking, every Aczel set can be obtained as a quotient of an $\in$ -set, factorized by the strongest extension conform equivalence relation (which is closely related to bisimulation).

The set theoretic interpretation of objects in $\in$ -structures not only leads to a new set theory in which sets are viewed as collections of "statements of memberships", but also allows for semantic models with circularity properties, not constructible under conventional set theory like ZF. Such models have not only been constructed for investigating modeltheoretic semantics in natural language processing, but also for untyped λ -calculus.

In [Po 92] the idea of selfapplicable functions, which is already in the above example, is used to define $\in$ -structures which contain objects, say D , that solve the domain equation

$$D = (D \longrightarrow D)$$

for some set of functions from D to D correctly, while regarding functions in a conventional way as sets of ordered pairs. Based on this conception, an environment model of the λ -calculus is defined which, under an appropriate notion of homomorphism, is initial in the class of all generated models.

6 A Framework for Declaration and Type Disciplines under the Constructive View

What we accept as an admissible declaration in a discipline $D = (\text{Expr}, C, \text{Decl})$ is only listed in the class Decl . But in most of the known declaration and type disciplines declarations satisfy internal completeness and consistency conditions and are constructable from inductive definitions (signature, terms, axioms, specifications). Following the setting of type theory under the constructive view, we use rules in this framework for the design of declarations. Such *rules have the following general form*.

$$\frac{}{J} \quad \text{or} \quad \frac{J_1 \quad \cdot \quad \cdot \quad \cdot \quad J_n}{J}$$

with $J, J_1, \dots, J_n$ being judgements. Proofs and derivations are conventionally defined and give rise to the notion of *consequence*

$$R \vdash J$$

where R is a set of rules and J is a judgement that can be derived using the rules of R . Judgements are uniformly assumed to *have one of the following forms*:

$c \text{ ident}$	(construction)
$\Delta \text{ decl}$	(declarations)
$\Delta \Vdash a:A$	(construction)
$\Delta \vdash a:A$	(inference)

This choice of forms is motivated from the observation that rules in the setting of type theory under the constructive view generally fulfill one of the tasks: construction, declaration or inference. Furthermore, we can restrict ourselves to type propositions as the only entity which creates judgements (note, that the judgements $c \text{ ident}$ and $\Delta \text{ decl}$ are actually type propositions, just written differently). The particular choice of rules and judgements appropriate for a particular discipline does very much depend on the underlying intention. But generally, if a *judgement of declaration*, $\Delta \text{ decl}$, is made, Δ is assumed to be a declaration, i.e. a set of type propositions. The *judgement of construction*, $\Delta \Vdash a:A$, expresses that there is enough information in Δ to construct $a:A$. The *judgement of inference*, $\Delta \vdash a:A$, expresses that the type proposition $a:A$ is a consequence of Δ , thereby assuming that every model of Δ is also a model of $a:A$. We can now say that a discipline $D = (\text{Expr}, C, \text{Decl})$ is *constructive*, if a set of rules R exists such that for every $\Delta \in \text{Decl}$ we have $R \triangleright \Delta \text{ decl}$.

The following set of rules, to give a simple example, allows to derive algebraic signatures

$\frac{}{c \text{ ident}}$	(introducing an identifier)
$\frac{}{\emptyset \text{ decl}}$	(introducing the empty declaration)
$\frac{\Delta \Vdash c:E}{\Delta \cup \{c:E\} \text{ decl}}$	(building of declarations)
$\frac{\Delta \Vdash c:E \quad \Delta \cup \{f:F\} \text{ decl}}{\Delta \cup \{f:F\} \Vdash c:E}$	(monotonicity of construction)
$\frac{\Delta \text{ decl} \quad s \text{ ident}}{\Delta \Vdash s : \text{sort}}$	(constructing a sort)
$\frac{c \text{ ident} \quad \Delta \Vdash s : \text{sort}}{\Delta \Vdash c : s}$	(constructing a constant)

$$\vdash \text{ident}$$

$$\Delta \Vdash s1 : \text{sort}$$

$$\Delta \Vdash s_{n+1} : \text{sort}$$

$$\Delta \Vdash f : s1 \dots sn \longrightarrow s_{n+1} \quad (\text{constructing an } n\text{-ary operation symbol})$$

In [MSU 90] further rules are given which allow for the construction of algebraic specifications and the inference of equational logic.

Constructive disciplines are usually more liberal than those defined in conventional ways, which is due to the fact that in this framework there are no means to produce information that restricts the use of identifiers.

A so called *standard discipline* has been designed in [Um 91] which is presently under revision. It includes logic, set theoretic operators and functions. Based on this discipline, an implementation of this framework, called *calculus of declarations*, has been given in [BD 91].

A major investigation has been made in a discipline of first order propositions in Sträters thesis [Str 92], which leads to a theory of truth with total truth predicates. This theory, called $\in \mathcal{T}$ -logic, allows for self reference, like the liar paradox

$$(x = (x : \text{false})) : \text{true}$$

and impredicative quantification over propositions. Both, self reference and impredicativity appear naturally if intensional equality (proposition x says the same as proposition y) and quantification are admitted. A complete calculus for this logic is given and, based on the ideas of $\in$ -structures, noncyclic extensional and intensional models are constructed which satisfy the Tarski biconditionals (expressing that a proposition about the model that is true is also true in the model, and vice versa) thereby proving the conjecture, ascribed to Tarski, false that such models with total truth predicates can not be built.

References

- [Acz 88] Aczel, P.: Non-Wellfounded Sets, CSLI Lecture Notes 14, 1988
- [Ba 90] Barendregt, H.: Introduction to Generalized Type Systems, Tech. Report 90-8, Univ. of Nijmegen, 1990.
- [BD 91] Ballmann, S., Dunker, G.: Entwurf und Implementierung für den Kalkül getypter Deklarationen, students thesis, Fachbereich 20, TU Berlin 1991
- [BHR 90] Bläsius, K.H., Hedtstück, U., Rollinger, C.R. (eds): Sorts and Types in Artificial Intelligence, LNAI 418, Springer, 1990

- [CF 58] Curry, H.B., Feys, R.: *Combinatory Logic*, North Holland, 1958
- [Ch 40] Church, A.: A formulation of the simple Theory of Types, *J. Symb. Log.* 5, 1940, 56-68
- [CH 88] Coquand, Th., Huet, G.: The Calculus of Constructions, *Information and Computation* 76, 2/3, 1988, 95-120
- [Co 85] Coquand, Th.: *Une théorie des constructions*, Thèse, Université Paris VII, 1985
- [EM 85] Ehrig, H., Mahr, B.: *Fundamentals of Algebraic Specification 1*, Springer-Verlag, 1985
- [FBL 84] Fraenkel, A.A., Bar-Hillel, Y., Levy, A.: *Foundations of Set Theory*, North-Holland, 1984
- [Ge 69] Gentzen, G.: The collected Papers of Gerhard Gentzen. ed E. Szabo, North Holland, 1969
- [Gi 71] Girard, J.Y.: Une extension de l'interprétation de Gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et dans le théorie des types, *Proc. of the Zud Scandinavian Logic Symposium* (ed. J.E. Fenstad), North Holland, 1971, 63-92
- [He 30] Herbrand, J.: Investigations in Proof Theory (1930) in: *Logical Writings*, ed. W.D. Goldfarb, Reidel, 1971
- [Hi 63] Higgins, P.J.: Algebra with a scheme of operators, *Mathematische Nachrichten*, Z7, 1963, 115 - 132
- [Ho 69] Howard, W.A.: The formulae-as-Types notion of construction, in: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, ed. J.P. Seldin and J.R. Hindley, Academic Press, 1980, 479-490
- [Luo 90] Luo, Z.: *An Extended Calculus of Constructions*, Dept. of Comp. Science, Univ. of Edinburgh, 1990
- [Mey 82] Meyer, A.R.: What is a model of Lambda Calculus? in: *Information and Control* 52, 1982, 87-122
- [Mi 78] Milner, R.: A Theory of Type Polymorphism in Programming, *J.C.S.S.* 17, 1978, 384-375
- [ML 71] Martin-Löf, P.: A Theory of Types, Report 71-3, Dept. of Math., Univ. of Stockholm, 1971
- [ML 80] Martin-Löf, P.: Constructive Mathematics and Computer Programming, *Logic, Methodology and Philosophy of Science* 6, 1980, 153-175
- [ML 84] Martin-Löf, P.: *Intuitionistic Type Theory*, Bibliopolis, 1984
- [MSU 90] Mahr, B., Sträter, W., Umbach, C.: *Fundamentals of a Theory of Types and Declarations*, KIT-Report 82, TU Berlin, Fachbereich 20, 1990
- [No 81] Nordström, B.: Programming in Constructive Set Theory: Some Examples. *Proc. ACM Conf. of Funct. Prog. Lang. and Comp. Arch.*, ACM, 1981, 141-154
- [NPS 90] Nordström, B., Petersson, K., Smith, J.M.: *Programming in Martin-Löfs` Type Theory*, Clarendon Press, 1990
- [Po 92] Pooyan, L.: ϵ -structures as Semantic Models of the λ -calculus, Diploma Thesis, Fachbereich Informatik, TU Berlin, 1992

- [Qu 63] Quine, W.O.: Mengenlehre und ihre Logik, Vieweg, 1973, First published in 1963
- [Re 74] Reynolds, J.C.: Towards a Theory of Type Structure, LNCS 19, Springer-Verlag, 1974, 408-425
- [Ru 08] Russell, B.: Mathematical Logic as based on the theory of types, Am. J. Math. 30, 1908, 222 - 262
- [Str 92] Sträter, W.: $\in_T$ Eine Logik erster Stufe mit Selbstreferenz und totalem Wahrheitsprädikat, KIT-Report 98, TU Berlin, Fachbereich 20, 1992
- [Str 92a] Sträter, W.: Internal notes on non wellfounded sets, Fachbereich 20, TU Berlin, 1992
- [Th 91] Thompson, S.: Type Theory and Functional Programming, Addison-Wesley, 1991
- [Um 91] Umbach, C.: A standard discipline in the calculus of declarations, Internal notes, Fachbereich 20, TU Berlin, 1991