Chapter 1
Program Testing and the Meaning Explanations of Intuitionistic Type Theory

Peter Dybjer
Department of Computer Science and Engineering
Chalmers University of Technology
Rännvägen 6, 412 96 Göteborg, Sweden
peterd@chalmers.se

Abstract

The relationship between program testing and Martin-Löf's meaning explanations for intuitionistic type theory is investigated. The judgements of intuitionistic type theory are viewed as conjectures which can be tested in order to be corroborated or refuted. This point of view provides a new perspective on the meaning of hypothetical judgements, since tests for such judgements need methods for generating inputs. Among other things, we need to generate function input. The continuity principle is invoked and the impredicativity of types of functionals is rejected. Furthermore, we provide testing semantics only for decidable identity types.

At the end we turn to impredicative type theories, and discuss possible testing semantics for such theories. In particular we propose that testing for impredicative type theory should be based on the evaluation of open expressions. This is in contrast to our testing semantics for Martin-Löf's predicative intuitionistic type theory which is based on the evaluation of closed expressions.

1.1 Introduction

Consider the following often cited remark by Knuth [22]:

> Beware of bugs in the above code; I have only proved it correct, not tried it.

How come? If you have proved your program correct, you should be certain that it works! However, several things can go wrong:

- The formal specification may fail to capture the intended behaviour of the program.
- The formal representation of the program in the logical system may fail to capture what actually happens when you run the program.

- The proof can be wrong. This easily happens with manual proofs, but even mechanically assisted proofs can be wrong. The logical principles may not be implemented correctly.

What does this have to do with the foundations of mathematics? There is yet another possibility:

- The logical principles employed may themselves be wrong! Maybe the logical system is inconsistent.

Can you "test" a logical law? Does the following make sense?

> Beware of bugs in the above proof; I have only followed inference rules, not run it.

However, you cannot in general test a proof in the same way as you can test a program. For example, how would you "run" a proof in Zermelo-Fraenkel set theory?

On the other hand, in Martin-Löf's *intuitionistic type theory* you can run proofs. In this theory the basic unit is that of a *judgement*. There are four forms: $A\ type$, $A = A', a \in A, a = a' \in A$. Each of these can be hypothetical, that is, depend on a context $x_1 \in A_1, \ldots, x_n \in A_n$. I shall argue that the following makes sense:

> When you've made your judgement evident to yourself, then you'd better run it, to make sure it's valid!

I shall explain what I mean by this, and let me immediately say that I do not mean running the type-checking algorithm used by proof assistants based on intuitionistic type theory! Instead I will base the discussion on the computation of closed expressions to canonical form, which underlies Martin-Löf's meaning explanations from the paper *Constructive Mathematics and Computer Programming* [26]. We shall see how the testing point of view provides a way to reformulate the meaning explanations using vocabulary from programming rather than from philosophy and logic. In other words, we look at the meaning explanations from the point of view of the computer programmer (or better the computer user) rather than from the point of view of the constructive mathematician.

Originally, the testing point of view that we explore in this paper was not meant to provide alternative meaning explanations of intuitionistic type theory, only an alternative *presentation* of these meaning explanations. Nevertheless, it seems that my interpretation of both hypothetical judgements and of type equality differs from Martin-Löf's [26, 27]. A consequence is that I only provide meaning to identity types $I(A, a, b)$ if the equality on A is a decidable, but not otherwise, for example, when A is a function type $N \to N$.

Another aim has been to pave the way for meaning explanations for other systems than Martin-Löf type theory. If the essence of the meaning explanations is that they explain how to test judgements, then we can ask ourselves whether we can write testing manuals for other logical systems than intuitionistic type theory. As an example, we shall discuss how to test the judgements of Coquand and Huet's Calculus of Constructions [10], an impredicative intuitionistic type theory.

In the future I hope to provide testing interpretations of systems including coinductive types and partial types and functions. It would also be interesting to rephrase

insights about the computational content of classical logic [9] as Martin-Löf style meaning explanations. A research program with this aim is Hayashi's *Limit Computable Mathematics*: "To test formalization of proofs by experiments (animation) via Gold's limiting recursive functions" [17]. Already in the 1980s Hayashi pioneered the idea of *proof animation*, whereby he utilized the Curry-Howard isomorphism to test formal proofs in his system PX [18] in much the same way as one tests computer programs [19]. Limit Computable Mathematics is the extension of this idea to classical logic. Hayashi's ideas have been an important source of inspiration for the present work.

Plan of the paper

In Section 2 we review the meaning explanations for intuitionistic type theory without committing ourselves to either a *pre-mathematical* or a *meta-mathematical* interpretation. In Section 3 we recall a meta-mathematical realizability interpretation following Allen [4]. The reader who is familiar with the meaning explanations can skip Sections 2 and 3 and go straight to Section 4 which is the principal part of the paper. There we propose a pre-mathematical testing interpretation involving evaluation of terms and generation of inputs. In Section 5 we briefly discuss the possibility of a pre-mathematical testing interpretation for the Calculus of Constructions.

1.2 Meaning explanations

History

Martin-Löf's meaning explanations for intuitionistic type theory were first presented in 1979 in the paper *Constructive mathematics and computer programming* [26]. These ideas were elaborated on in the book *Intuitionistic Type Theory* [27]. In both works meaning explanations are used to justify an extensional polymorphic version of intuitionistic type theory. Another useful reference for the philosophical basis of meaning explanations is *On the meaning of the logical constants and the justification of the logical laws* [28] from 1983, although this is concerned with meaning explanations for ordinary intuitionistic predicate logic (without formal proof objecs) rather than for intuitionistic type theory.

The meaning explanations are also refered to as *direct semantics*, *intuitive semantics*, *informal semantics*, *standard semantics*, or the *syntactico-semantical* approach to meaning theory. Semantics is here understood *pre-mathematically* rather than *meta-mathematically*, as is clear from the following quotation from the first paragraph of *Intuitionistic Type Theory*.

> Mathematical logic and the relation between logic and mathematics have been interpreted in at least three different ways:
>
> 1. mathematical logic as symbolic logic, or logic using mathematical symbolism;
> 2. mathematical logic as foundations (or philosophy) of mathematics;
> 3. mathematical logic as logic studied by mathematical methods, as a branch of mathematics.
>
> We shall here mainly be interested in mathematical logic in the second sense. What we shall do is also mathematical logic in the first sense, but certainly not in the third.

In the present paper we shall also do mathematical logic in the third sense, when we interpret the meaning explanations meta-mathematically in Section 3. Hilbert's original use of the word meta-mathematics assumed that only finitistic methods were used on the meta-level. However, in Section 3 we assume that general mathematical (classical set-theoretic) methods are available, although we could argue informally that we only use parts of set theory which are constructively valid. In Section 4, we shall interpret the meaning explanations in the second sense, that is, pre-mathematically, which means that we restrict ourselves to everyday concepts relating to programming: running a program, observing its result, generating input, etc.

The meaning explanations were a profound contribution to the semantics of intuitionistic type theory (and to intuitionism in general) when they were first presented in 1979. Prior versions of intuitionistic type theory did not come with such meaning explanations, but were justified by normalization proofs. Consistency then follows from the Church-Rosser property. The first version, *A theory of types* from 1971 [24], which had an axiom that there is a type of all types, was actually inconsistent, although it was proved to have the normalization property. The problem was that the meta-theory of the normalization proof was itself inconsistent. This problem was rectified in *An intuitionistic theory of types* from 1972 [29], where the type of all types was replaced by a type of *small* types (a universe), and a correct normalization proof was provided. The first published version of type theory *An intuitionistic theory of types: predicative part* from 1973 [25] had an infinite sequence of universes and contained a proof of normalization by an intuitionistic, but meta-mathematical, model construction.

Terms and computation rules

We shall now give an informal account of Martin-Löf's meaning explanations for the fragment of his intuitionistic type theory where the only types are natural numbers N, identity types $I(A, a, b)$, cartesian products of families of types $\Pi(A, B)$, and a universe of small types U. These type formers suffice to illustrate the main points. The discussion of other type formers is analogous.

We shall stay rather close to the accounts given in *Constructive Mathematics and Computer Programming* [26] and *Intuitionistic Type Theory* [27], although there will be three (relatively minor) differences:

- We will present type theory based on a theory of expressions (as proposed in the preface of *Intuitionistic Type Theory* [27], see also [31]). This choice is inessential; similar meaning explanations can be provided for both earlier and later versions of intuitionistic type theory, and can also be extended to deal with other types if details are modified suitably.
- Another inessential difference is that we will present both computation rules and meaning explanations using inference rule notation, although such notation is not employed in [26, 27]. Later on we will discuss in detail how to read these inference rules both meta-mathematically (Section 3) and pre-mathematically (Section 4).
- Martin-Löf [26, 27] actually considers an extensional type equality: two canonical types are equal iff they have equal objects and equal object equality. However, in our testing semantics it will be easier to justify an intensional type equality, where two canonical types are equal only if they begin with the same type constructor. Both extensional and intensional type equality were previously discussed by Allen [5, 4].

The *theory of expressions* is nothing but the simply typed lambda calculus with one base type ι. The types are called "arities" to distinguish them from the types of type theory which will be introduced later. Abstraction in the theory of expressions is written $(x)a$, and application is written $f(a)$. Furthermore, we add constants for the type formers, and for the canonical and non-canonical term constructors. The arities of the constants in our fragment of intuitionistic type theory are

$$\begin{aligned}
N &: \iota \\
0 &: \iota \\
s &: \iota \to \iota \\
R &: \iota \to \iota \to (\iota \to \iota \to \iota) \to \iota \\
I &: \iota \to \iota \to \iota \to \iota \\
r &: \iota \\
J &: \iota \to \iota \to \iota \\
\Pi &: \iota \to (\iota \to \iota) \to \iota \\
\lambda &: (\iota \to \iota) \to \iota \\
Ap &: \iota \to \iota \to \iota \\
U &: \iota
\end{aligned}$$

We abbreviate $f(a_1, \ldots, .a_n) = f(a_1) \cdots (a_n)$. Moreover, $(\lambda x)a = \lambda((x)a)$ and $(\Pi x \in A)B = \Pi(A, (x)B)$.

Universes are formulated à la Russell [27] and there is a common syntactic category of types and terms.

Judgements are interpreted in terms of the relation $a \Rightarrow v$ between *closed* terms of base arity, meaning "a has canonical form v". Canonical forms are *lazy*:

$$v ::= \Pi(a, a) \mid \lambda(a) \mid 0 \mid s(a) \mid I(a, a, a) \mid r \mid U \mid \cdots$$

where a ranges over arbitrary, not necessarily canonical, terms. The canonical form relation is given by the following computation rules:

$$\frac{c \Rightarrow 0 \quad d \Rightarrow v}{R(c,d,e) \Rightarrow v} \qquad \frac{c \Rightarrow s(a) \quad e(a, R(a,d,e)) \Rightarrow v}{R(c,d,e) \Rightarrow v}$$

$$\frac{c \Rightarrow r \quad d \Rightarrow v}{J(c,d) \Rightarrow v}$$

$$\frac{c \Rightarrow \lambda(b) \quad b(a) \Rightarrow v}{Ap(c,a) \Rightarrow v}$$

in addition to the rule

$$v \Rightarrow v$$

stating that a canonical term has itself as value.

The meaning of judgement forms

The general principle is that if A is a type, then it has a canonical type as value:

$$\frac{A \Rightarrow C(a_1, \ldots, a_m) \quad \cdots}{A \; type}$$

where C is an m-place type constructor and $\cdots$ stand for additional requirements on $a_1, \ldots . a_m$, which are part of the definition of what it means to be a type.

The types A and A' are equal if their canonical forms begin with the same type constructor:

$$\frac{A \Rightarrow C(a_1, \ldots, a_m) \quad A' \Rightarrow C(a'_1, \ldots, a'_m) \quad \cdots}{A = A'}$$

where $\cdots$ stand for additional requirements on $a_1, a'_1, \ldots, a_m, a'_m,$. (Note again that this an intensional notion of type equality which differs from the extensional type equality in Martin-Löf's [26] meaning explanations.)

Furthermore, a is an element of A provided

$$\frac{A \Rightarrow C(a_1, \ldots, a_m) \quad a \Rightarrow c(b_1, \ldots, b_n) \quad \cdots}{a \in A}$$

where c is an n-place term constructor for C, and where $\cdots$ stand for additional requirements on $a_1, \ldots, a_m, b_1, \ldots, b_n$.

The elements a and a' are equal elements of A provided

$$\frac{A \Rightarrow C(a_1, \ldots, a_m) \quad a \Rightarrow c(b_1, \ldots, b_n) \quad a' \Rightarrow c(b'_1, \ldots, b'_n) \quad \cdots}{a = a' \in A}$$

and where $\cdots$ stand for additional requirements on $a_1, \ldots, a_m, b_1, b'_1 \ldots, b_n, b'_n$.

General schema for type formers

Martin-Löf does not elaborate on what is allowed as $\cdots$ in the schematic rules above. The schema for inductive-recursive definitions [12, 13, 14] provides a general form which covers most type formers existing in the literature. Beyond that, Setzer has proposed several proof-theoretically stronger types: a Mahlo universe [37] and in unpublished work an autonomous Mahlo universe and a Π_3-reflecting universe. We hope that the present article helps explaining what is involved in claiming that these large types are constructively valid, and why their justification is *predicative*, although we do not discuss their testing semantics explicitly.

In this article, however, we restrict ourselves to a few crucial instances of canonical types: $N, I(A, a, b), \Pi(A, B)$, and U.

Natural numbers

$$\frac{A \Rightarrow N}{A\ type}$$

$$\frac{A \Rightarrow N \quad A' \Rightarrow N}{A = A'}$$

$$\frac{A \Rightarrow N \quad a \Rightarrow 0}{a \in A} \qquad \frac{A \Rightarrow N \quad a \Rightarrow s(b) \quad b \in N}{a \in A}$$

$$\frac{A \Rightarrow N \quad a \Rightarrow 0 \quad a' \Rightarrow 0}{a = a' \in A} \qquad \frac{A \Rightarrow N \quad a \Rightarrow s(b) \quad a' \Rightarrow s(b') \quad b = b' \in N}{a = a' \in A}$$

Identity types

$$\frac{A \Rightarrow I(B, b, b') \quad b \in B \quad b' \in B}{A\ type}$$

$$\frac{A \Rightarrow I(B, b, c) \quad A \Rightarrow I(B', b', c') \quad B = B' \quad b = b' \in B \quad c = c' \in B}{A = A'}$$

$$\frac{A \Rightarrow I(B, b, b') \quad a \Rightarrow r \quad b = b' \in B}{a \in A}$$

$$\frac{A \Rightarrow I(B,b,b') \quad a \Rightarrow r \quad a' \Rightarrow r \quad b = b' \in B}{a = a' \in A}$$

Remarks on the meaning of identity types

Note that the judgement $A = A'$ presupposes the judgements $A, A'\ type$, since the former judgement can be valid only if the latter are. Similarly, $a \in A$ presupposes $A\ type$, and $a = a' \in A$ presupposes $a, a' \in A$, and $A\ type$. We do not write out presupposed judgements in the rules.

Note that $r \in I(B,b,b')$ iff $b = b' \in B$. Hence these meaning explanations justify the rule

$$\frac{\Gamma \vdash c \in I(B,b,b')}{\Gamma \vdash b = b' \in B}$$

which is present in extensional type theory, but not in intensional type theory [29, 25, 31].

Note that we can form the type $I(B,b,b')$ for any type B in intuitionistic type theory [25, 26, 27], and that this construction will be validated by the meta-mathematical model construction (realizability) in the next section. However, this rule will not be validated by our pre-mathematical testing semantics for general B, only for B with decidable equality. (This does not mean that we cannot define a type expressing the extensional equality of two functions, only that this type is not primitive.)

Function types

$$\frac{A \Rightarrow \Pi(B,C) \quad y \in B \vdash C(y)\ type}{A\ type}$$

$$\frac{A \Rightarrow \Pi(B,C) \quad A' \Rightarrow \Pi(B',C') \quad B = B' \quad y \in B \vdash C(y) = C'(y)}{A = A'}$$

$$\frac{A \Rightarrow \Pi(B,C) \quad a \Rightarrow \lambda(c) \quad y \in B \vdash c(y) \in C(y)}{a \in A}$$

$$\frac{A \Rightarrow \Pi(B,C) \quad a \Rightarrow \lambda(c) \quad a' \Rightarrow \lambda(c') \quad y \in B \vdash c(y) = c'(y) \in C(y)}{a = a' \in A}$$

Note that some premises are hypothetical judgements. For example, $y \in B \vdash C(y)\ type$ means that $C(y)$ is a type under the assumption that $y \in B$. At this point we shall not discuss the meaning of hypothetical judgements further. The reader is referred to Section 3 for a meta-mathematical interpretation and to Section 4 for a pre-mathematical interpretation. Note that we have again omitted presupposed judgements, and that if $x \in A$ is an assumption in a valid hypothetical judgement, then $A\ type$ is also valid and can be presupposed.

The universe (à la Russell) of small types

$$\frac{A \Rightarrow U}{A\ type}$$

$$\frac{A \Rightarrow U \quad A' \Rightarrow U}{A = A'}$$

$$\frac{A \Rightarrow U \quad a \Rightarrow N}{a \in A} \qquad \frac{A \Rightarrow U \quad a \Rightarrow I(b,c,d) \quad b \in U \quad c,d \in b}{a \in A}$$

$$\frac{A \Rightarrow U \quad a \Rightarrow \Pi(b,c) \quad b \in U \quad y \in b \vdash c(y) \in U}{a \in A}$$

$$\frac{A \Rightarrow U \quad a \Rightarrow N \quad a' \Rightarrow N}{a = a' \in A}$$

$$\frac{A \Rightarrow U \quad a \Rightarrow I(b,c,d) \quad a' \Rightarrow I(b',c',d') \quad b = b' \in U \quad c = c' \in b \quad d = d' \in b}{a = a' \in A}$$

$$\frac{A \Rightarrow U \quad a \Rightarrow \Pi(b,c) \quad a' \Rightarrow \Pi(b',c') \quad b = b' \in U \quad y \in b \vdash c(y) = c'(y) \in U}{a = a' \in A}$$

Other inductive and inductive-recursive types

Analogous rules can be given for the remaining type formers of intuitionistic type theory [26, 27]. Even more, analogous rules can be given for inductive types and inductive-recursive types [12, 13]. It would be interesting to discuss the rules for inductive and inductive-recursive families [12, 14] from the testing point of view, but this is outside the scope of the present paper.

1.3 Meta-mathematical reading - realizability

We shall now interpret the meaning explanations meta-mathematically, using set theory as meta-language and essentially following Allen [5, 4]. It is sometimes said that the meaning explanations are nothing but a realizability interpretation (in the sense of Kleene), but this is fundamentally misleading. Realizability provides a meta-mathematical and not a pre-mathematical interpretaion! Nevertheless, it helps us to be precise and to understand the details involved in the meaning explanations.

Realizability interpretations of intuitionistic type theory go back to Aczel [2, 3], Beeson [6], and Smith [38]. Note that Aczel [3], Beeson [6], and Allen [5, 4] are "semantic" interpretations where set theory is the meta-language, whereas Aczel [2] and Smith [38] provide syntactic translations into versions of intuitionistic predicate logic. Allen's version is closest to Martin-Löf's meaning explanations; it is a rather direct mathematical interpretation of the relevant passages in [26, 27].

In the meta-mathematical interpretation, the set of terms and the arity relation are defined inductively (as usual for the simply typed lambda calculus). Furthermore, the canonical terms is an inductively defined subset of the set of terms and the canonical form relation is an inductively defined relation between terms and canonical terms of arity ι.

The question is then how to give a meta-mathematical interpretation of the rules providing the meaning of the judgements? A first try would be to let them inductively generate four relations on terms – one for each form of judgement. However, this would not yield a positive inductive definition, since the $\in$-relation appears negatively in the rules for Π and U. To turn these rules into a meaningful set-theoretic definition is the main problem which was solved in different ways by Aczel, Beeson, and Allen.

Allen uses the fact that it suffices to interpret the judgement $a = a' \in A$; the interpretation of the other judgement can then also be derived, like in other work on partial equivalence relation (per) models. He defines a relation $\mathcal{E}l \subseteq \Lambda \times \mathcal{P}(\Lambda^2)$, such that $a = a' \in A$ is valid in the model iff there is an $\mathcal{R}$ such that $(A, \mathcal{R}) \in \mathcal{E}l$ and $(a, a') \in \mathcal{R}$. In other words, $\mathcal{E}l$ is the graph of the function which maps a type A to its notion of element equality, a partial equivalence relation on Λ. With this method the rules can be understood as the rules of a positive inductive definition of $\mathcal{E}l$.

Allen follows Martin-Löf [26] and interprets type equality extensionally: two types are equal iff they have the same canonical elements and the same equal canonical elements. However, he points out that if we want to interpret a theory with intensional type equality then we should instead inductively generate $\mathcal{E}l \subseteq \Lambda^2 \times \mathcal{P}(\Lambda^2)$, but does not present the details. However, since our testing interpretation in the next section naturally gives rise to intensional type equality we shall sketch its realizability counterpart here.

The $\mathcal{E}l$-relation interprets an auxiliary heterogenous equality judgement:

- $A \in a = a' \in A'$ is valid in the model iff there is an $\mathcal{R}$ such that $((A, A'), \mathcal{R}) \in \mathcal{E}l$ and $(a, a') \in \mathcal{R}$.

$\mathcal{E}l$ is the graph of a function which maps equal types to the per they denote. The domain of this function is a per $\sim$ which will interpret type equality:

- $A = A'$ is valid in the model iff here is an $\mathcal{R}$ such that $((A, A'), \mathcal{R}) \in \mathcal{E}l$.

The other judgement forms are interpreted as follows:

- $A\ type$ is interpreted as $A = A$;
- $a = a' \in A$ is interpreted as $A \in a = a' \in A$;
- $a \in A$ is interpreted as $a = a \in A$;

The interpretation of a hypothetical judgement is

- $x_1 \in A_1, \ldots, x_n \in A_n \vdash \mathcal{J}$ is interpreted as whenever $a_1 \in A_1, \ldots, a_n \in A_n$ then $\mathcal{J}[a_1/x_1, \ldots, a_n/x_n]$.

We will use Aczel's rule sets [1] to present the inductive definition of $\mathcal{E}l$. In this setting an inductive definition is given by a set of rules, where a *rule* on a set V is a pair $\frac{X}{x}$, such that $X \subseteq V$ and $x \in V$. A set $A \subseteq V$ is closed under this rule iff $X \subseteq A$ implies $x \in A$. A is said to be inductively generated by a set of rules Φ iff it is the least set closed under all rules in Φ. (Aczel's notion of inductive definition given by rule sets is equivalent to the notion given by monotone operators.)

To interpret N we inductively generate the per $\mathcal{N}$ of equal natural numbers. The rule set is

$$\{\frac{\emptyset}{(a,a')} \mid a \Rightarrow 0, a' \Rightarrow 0\} \cup \{\frac{\{(b,b')\}}{(a,a')} \mid a \Rightarrow s(b), a' \Rightarrow s(b')\}$$

To interpret Π-types, we define the cartesian product of a doubly indexed famiy of pers. Let be a per, and $\mathcal{R}(y,y')$ be a per for y, y' such that yy'. Then

$$\prod(,\mathcal{R}) = \{(a,a')|a \Rightarrow \lambda(c), a' \Rightarrow \lambda(c'), (\forall yy')c(y)\mathcal{R}(y,y')c'(y')\}$$

To interpret I-types we let be a per, and define the per of identity proofs:

$$\mathcal{I}(,b,c) = \{(a,a')|a \Rightarrow r, a' \Rightarrow r, bc\}$$

To interpret the type of small types U we first inductively generate the relation $\mathcal{T} \subseteq \Lambda^2 \times \mathcal{P}(\Lambda^2)$, the graph of the function mapping two equal *small* types to the per they denote.

$$\{\frac{\emptyset}{((A,A'),\mathcal{N})}|A \Rightarrow N, A' \Rightarrow N\}$$

$$\cup$$

$$\{\frac{\{((B,B'),)\} \cup \{((C(y),C'(y'),\mathcal{R}(y,y'))|yy'\}}{((A,A'),\prod(,\mathcal{R}))}|A \Rightarrow \Pi(B,C), A' \Rightarrow \Pi(B',C')\}$$

$$\cup$$

$$\{\frac{\{((B,B'),)\}}{((A,A'),\mathcal{I}(,b,c))}|A \Rightarrow I(B,b,c), A' \Rightarrow I(B',b',c'), bb', cc'\}$$

We then interpret U as the per $\mathcal{U}$ of equal small types, that is, the domain of $\mathcal{T}$:

$$\mathcal{U} = \{(A,A')|\exists\mathcal{R}.(A,A')\mathcal{T}\mathcal{R}\}$$

Finally we inductively generate $\mathcal{E}l \subseteq \Lambda^2 \times \mathcal{P}(\Lambda^2)$, the graph of the function mapping two equal possibly *large* types to the per they denote. The rules for $\mathcal{E}l$ is the union of the rule set for $\mathcal{T}$ and the rule set

$$\{\frac{\emptyset}{((A,A'),\mathcal{U})}|A \Rightarrow U, A' \Rightarrow U\}$$

As explained above we can interpret all the judgement forms of intuitionistic type theory using $\mathcal{E}l$, and also show that all the inference rules of this theory are valid.

However, we refer to Allen for the details and limit ourselves to validate the rule of N-elimination:

$$\frac{C(x)\ type \qquad a \in N \qquad d \in C(0) \qquad x \in N, y \in C(x) \vdash e(x,y) \in C(s(x))}{R(a,d,e) \in C(a)}$$

We have here omitted the context Γ which is common to the premises and the conclusion.

To prove the validity of this rule we do induction on $\mathcal{N}$, the interpretation of N. We know that if $a \in N$ is valid, then either $a \Rightarrow 0$ or $a \Rightarrow s(b)$ where $b \in N$ is valid. If $a \Rightarrow 0$ then $R(a,d,e) \Rightarrow v$ iff $d \Rightarrow v$. Thus $d \in C(0)$ implies $R(a,d,e) \in C(a)$, since $\in$ is invariant under evaluation: if $a \Rightarrow v$ and $a' \Rightarrow v'$ then $a \in C(a')$ iff $v \in C(v')$ (a lemma to be proved). The validation of the case where $a \Rightarrow s(b)$ is similar.

1.4 Pre-mathematical reading - a testing manual for intuitionistic type theory

The meta-mathematical reading of the meaning explanations does not have foundational significance. To prove that mathematical induction (N-elimination) is valid in the model we rely on (something more complex than) the principle of mathematical induction in the meta-language. Similarly, to justify the correctness of proof-theoretically strong extensions of Martin-Löf type theory, such as Setzer's Mahlo universe [37], autonomous Mahlo universe, and Π_3-reflecting universe, we need analogous notions in the meta-language: Mahlo cardinals, autonomous Mahlo cardinals, and Π_3-reflecting cardinals.

Instead we shall understand Martin-Löf's meaning explanations as a manual for testing the validity of judgements. For example, to test the rule of N-elimination, we test the primitive recursion combinator $R(c,d,e)$ for $c = 0, c = s(0), c = s(s(0)), \ldots$, and for arbitrary base case d and arbitrary step case e which satisfy the assumptions of the rule. Note that, such tests are necessarily incomplete since there are infinitely many possible inputs. As in inductive inference in philosophy of science, a judgement is a conjecture which can be corroborated by a successful test or refuted by an unsuccessful test, see Popper [33]. A justification of the rule of induction is a justification of our belief that all tests of the primitive recursion combinator will succeed. Such justifications are of course fallible, although the risk that the rule of N-elimination is incorrect might seem slim. However, if we remember that it is only meaningful to test an *implementation* of the primitive recursion combinator, we realize that there is a real danger of falsification.

Similarly, to give meaning explanations for Setzer's Mahlo universe, autonomous Mahlo universe, and Π_3-reflecting universe involves explaining how judgements containing them are to be tested. To justify the rules for these universes we must explain why there is reason to believe that all such tests will succeed. Here the risk

of falsification is greater, although these universes are defined in such a way that it should be possible to "see" that they pass all tests. This is achieved by explaining how they are "built up from below" or how some measure decreases during computation over them. Again, such explanations are made by humans and fallible. We cannot say that these rules are constructively valid in an absolute sense, only that we believe them to be constructively valid in the sense that we believe that they will pass all tests. And when we say that they are "predicatively valid" we mean that we can provide a justification which suggests a "well-founded" structure. Again, such a justification is fallible.

The program testing point of view emphasizes that meaning explanations are about what *really* happens! It is a process in space-time. An expression is something static (in space). Computation is something dynamic (in time). The formal proposition $a \Rightarrow v$ is a static mathematical representation of the *real* fact that a will turn into v a little later after some computation is done. This relates to Martin-Löf's term "syntactico-semantical". Semantics is what happens during execution. The meaning is the extension which is gradually unfolded as time goes by.

Coming back to the remark in the beginning of the introduction: the above discussion of validity and testing is rather obvious in the context of software testing. It is something hardly worth saying. But when we present the same idea in the context of intuitionistic type theory it nevertheless seems surprising.

Is mathematics an empirical science?

The relevance of a similar perspective has recently been argued by Miquel in his essay *The experimental effectiveness of mathematical proof* [30][p 38]:

> We can thus argue (against Popper) that mathematics fulfills the demarcation criterion that makes mathematics an empirical science. The only specificity of mathematics is that the universal empirical hypothesis underlying mathematics is (almost) never stated explicitly.

Pre-mathematical understanding of terms and computation rules

We now read the grammar and typing rules as concrete prescriptions for generating well-formed terms of type theory. Contrast this to the meta-mathematical interpretation in terms of the mathematical concept of an inductive definition. Similarly, the computation rules should be read as concrete prescriptions for how to evaluate a term to canonical form. Here this is a prescription for manual evaluation of a term, and hopefully this prescription is sufficiently precise to reproduce the same value each time. Alternatively, we can replace this prescription by an implementation, a real machine which can perform the evaluation.

Testing categorical judgements

We shall now read the rules of the meaning explanations, which in the meta-mathematical interpretation were made to inductively generate the realizability interpretation, as a *testing manual*, a manual for *falsification of conjectures*, or *bug-finding*. A tester only needs to push a button "execute program" and inspect the results. He or she is only a "user" who does not need to know logic or programming.

We first show how to test the *categorical* judgements, that is, judgements without hypotheses.

How to test A type?

The relevant rules are

$$\frac{A \Rightarrow N}{A\ type} \qquad \frac{A \Rightarrow I(B,b,c) \quad b,c \in B}{A\ type}$$

$$\frac{A \Rightarrow \Pi(B,C) \quad y \in B \vdash C(y)\ type}{A\ type} \qquad \frac{A \Rightarrow U}{A\ type}$$

As before, we do not write out presupposed assumptions. For example, the assumption that $B\ type$ in the second and third rules is omitted. This is reflected in the testing manual: we do not need to test presupposed assumptions, since their validity is a consequence of a succesful test of another assumption.

To test $A\ type$ we always begin by evaluating A to canonical form, where the outermost form is a constructor.

- If it has canonical form N, then the test is successful.
- If it has canonical form $I(B,b,c)$, then test $b \in B$ and $c \in B$.
- If it has canonical form $\Pi(B,C)$, then test $B\ type$ and $y \in B \vdash C(y)\ type$. The latter is a hypothetical judgement the testing of which will be explained below.
- If it has canonical form U, then the test is successful.
- If it has a canonical form which does not begin with a type constructor then the test fails.

If A has no canonical form, then the judgement is not valid either. In this case we will wait forever – at no stage will we observe a canonical form. Nevertheless, if we observe the intermediate stages of the computation of A we may for example detect an infinite loop, and thus conclude that it will never terminate. However, this requires higher level reasoning than the simple observations of canonical forms.

How to test $A = A'$?

The relevant rules are

$$\frac{A \Rightarrow N \quad A' \Rightarrow N}{A = A'}$$

$$\frac{A \Rightarrow I(B,b,c) \quad A \Rightarrow I(B',b',c') \quad B = B' \quad b = b' \in B \quad c = c' \in B}{A = A'}$$

$$\frac{A \Rightarrow \Pi(B,C) \quad A' \Rightarrow \Pi(B',C') \quad B = B' \quad y \in B \vdash C(y) = C'(y)}{A = A'}$$

$$\frac{A \Rightarrow U \quad A' \Rightarrow U}{A = A'}$$

To test $A = A'$ we always begin by evaluating A and A' to canonical form, where the outermost form is a constructor.

- If both have canonical form N, then the test is successful.
- If $A \Rightarrow I(B,b,c)$ and $A' \Rightarrow I(B',b',c')$, then first test $B = B'$ and if successfull test $b = b' \in B$ and then $c = c' \in B$.
- If $A \Rightarrow \Pi(B,C)$ and $A' \Rightarrow \Pi(B',C')$, then test $B = B'$ and $y \in B \vdash C(y) = C'(y)$. The latter is a hypothetical judgement the testing of which will be explained below.
- If both have canonical form U, then the test is successful.
- If A and A' have canonical forms with different constructors or if one of them does not begin with a type constructor, then the test fails.

If neither A nor A' has canonical form, then the test fails, see the discussion of non-termination above.

How to test $a \in A$?

The relevant rules are

$$\frac{A \Rightarrow N \quad a \Rightarrow 0}{a \in A} \qquad \frac{A \Rightarrow N \quad a \Rightarrow s(b) \quad b \in N}{a \in A}$$

$$\frac{A \Rightarrow I(B,b,b') \quad a \Rightarrow r \quad b = b' \in B}{a \in A}$$

$$\frac{A \Rightarrow U \quad a \Rightarrow N}{a \in A} \qquad \frac{A \Rightarrow U \quad a \Rightarrow I(b,c,d) \quad b \in U \quad c,d \in b}{a \in A}$$

$$\frac{A \Rightarrow U \quad a \Rightarrow \Pi(b,c) \quad b \in U \quad y \in b \vdash c(y) \in U}{a \in A}$$

The testing manual states that we begin by running both A and a! We need both canonical forms in order to know which rule applies. The further premises of that rule tell us what further tests we need to perform:

- If $A \Rightarrow N$ and $a \Rightarrow 0$, then the test is successful.
- If $A \Rightarrow N$ and $a \Rightarrow s(b)$, then test whether $b \in N$.
- If $A \Rightarrow I(B, b, b')$ and $a \Rightarrow r$, then test whether $b = b' \in B$
- If $A \Rightarrow U$ and $a \Rightarrow N$, then the test is successful.
- If $A \Rightarrow U$ and $a \Rightarrow I(b, c, d)$, then test whether $b \in U$ and $c, d \in b$.
- If $A \Rightarrow U$ and $a \Rightarrow \Pi(b, c)$, then test whether $b \in U$ and $y \in b \vdash c(y) \in U$.
- If A and a have non-matching canonical forms, that is, combinations of canonical forms for which there is no rule, then the test fails.

If neither A nor a has canonical form, then the test fails, see the discussion of non-termination above.

How to test $a = a' \in A$?

We leave it to the reader to write the testing manual for this judgement form.

Testing hypothetical judgements and function types

Let us now turn to hypothetical judgements:

$$\Gamma \vdash \mathcal{J}$$

For example, how do we read the rule for Π-types

$$\frac{A \Rightarrow \Pi(B, C) \qquad a \Rightarrow \lambda(c) \qquad x \in B \vdash c(x) \in C(x)}{a \in A}$$

as a rule in our testing manual? What action should we take to test

$$x \in B \vdash c(x) \in C(x)?$$

In *Constructive Mathematics and Computer Programming* [26] and *Intuitionistic Type Theory* [27] the meaning of this hypothetical judgement is that $c(b) \in C(b)$ provided $b \in B$ (and also that $c(b) = c(b') \in C(b)$ provided $b = b' \in B$). However, this is not a satisfactory answer when we look at it from the testing point of view, because we must ask ourselves how we obtained $b \in B$. What if it came from a malicious hacker?

Instead we had better manufacture our own tests! To this end the rules for the judgement form $a \in A$ will be given a second reading: how to generate input to hypothetical tests! This is a point which to my knowledge has not previously been discussed in the context of Martin-Löf's meaning explanations. On the other hand, input generation is an important aspect of software testing, as in the testing

tools *QuickCheck* [8] and *SmallCheck* [36] for the functional programming language Haskell.

Input generation

The relevant rules are those which define the meaning of the judgement $a \in A$. We display those rules again but in order to suggest that they should now be read as rules for instantiating variables, we replace the letters $a, b, \ldots$, by $x, y, \ldots$. For this reason we also reorder the premises of some of the rules.

$$\frac{A \Rightarrow N \quad x \Rightarrow 0}{x \in A} \qquad \frac{A \Rightarrow N \quad x \Rightarrow s(y) \quad y \in N}{x \in A}$$

$$\frac{A \Rightarrow I(B,b,b') \quad b = b' \in B \quad x \Rightarrow r}{x \in A}$$

$$\frac{A \Rightarrow \Pi(B,C) \quad x \Rightarrow \lambda(z) \quad y \in B \vdash z(y) \in C(y)}{x \in A}$$

$$\frac{A \Rightarrow U \quad x \Rightarrow N}{x \in A} \qquad \frac{A \Rightarrow U \quad x \Rightarrow I(y,z,z') \quad y \in U \quad z, z' \in y}{x \in A}$$

$$\frac{A \Rightarrow U \quad x \Rightarrow \Pi(y,z) \quad y \in U \quad t \in b \vdash z(t) \in U}{x \in A}$$

If we want to generate $x \in A$, we begin by computing the canonical form of A:

- If it has canonical form N, then either generate $x = 0$ or generate $x = s(y)$ and then generate $y \in N$.
- If it has canonical form $I(B,b,b')$, then we need to test whether $b = b' \in B$. If so generate $x = r$. If not, there is no element to generate. Note that this requires us to *decide* in finite time whether $b = b' \in B$. (This test presupposes the test of $B\ type$, if this fails, then the whole judgement, in which $x \in A$ is a hypothesis, fails.)
- If it has canonical form $\Pi(B,C)$, then generate $x = \lambda(z)$ where z is a function such that $y \in B \vdash z(y) \in C(y)$. However, we do not generate function terms z but input-output pairs, as we shall explain below.
- If it has canonical form U, then either generate $x = N$, or generate $x = I(y,z,z')$ and then generate $y \in U, z, z' \in y$, or generate $x = \Pi(y,z)$, and then generate $y \in U$ and $t \in y \vdash z(t) \in U$. As above, we do not generate function terms z, but input-output pairs.

A question arises: should we fully instantiated a variable x to a closed expression before computing the expression which it is part of, or should we have a lazy instantiation procedure, where we compute with open expressions, and only instantiate x when it blocks further computation, for example, when we get an expression $R(x,d,e)$ which cannot be computed further unless we know the canonical form

of x. As we shall see it will be important to evaluate open expressions, when we generate functional input.

Functional input generation

To simplify the discussion, let us consider the generation of numerical functions only. How do we read the rule

$$\frac{x \Rightarrow \lambda(z) \quad y \in N \vdash z(y) \in N}{x \in N \to N}$$

as a rule for generating x? It states that we generate $x = \lambda(z)$ and then generate z so that $z(y) \in N$ for $y \in N$. Now, it would be wrong to try to read $y \in N \vdash z(y) \in N$ as syntactic derivability in some formal system for Martin-Löf type theory. Instead we want testing to be "local", that is, not depending on the formal system as a whole. We want the "semantic" notion, not a syntactic one!

This observation is relevant to the question of the impredicativity of types of functionals. A functional $g \in (N \to N) \to N$, is an expression such that $g(f) \in N$ for all $f \in N \to N$, including those f which contain g. This may seem circular if we take a syntactic point of view (generating function terms in a given type system), but not the semantic view (generating input-output pairs) in the sense to follow. It will follow that the justification of types of functionals is predicative in the testing interpretation.

When generating input-output pairs, it is useful to recall the lessons of domain theory (the continuity principle) and game semantics. What we need to do is to generate as many pairs (m, n) with $m, n \in N$ as needed! Consider for example testing

$$x \in N \to N \vdash b(x) \in N$$

- We begin testing $b(x) \in N$ without knowing x. This means we try to compute the canonical form of $b(x)$.
- At some stage the computation may get stuck because it does not know x. For example, $R(Ap(x, 0), d, e)$ needs to know the canonical form of $Ap(x, 0)$.
- So we generate an input-output pair $(0, y)$ for the function x, where the output $y \in N$ will be generated as described above. Now the computation can go on, until we either arrive at the canonical form or get stuck again.
- Etc.

We will not here provide a complete description of the generation of function input and leave it to future work on game semantics for dependent types.

Are function identity types meaningful?

As we saw above we do not provide testing semantics for function identity types. We are led to restrict the formation rule for the types $I(B, b, b')$ to cases where the judgement $b = b' \in B$ is decidable (under the assumption that $b, b' \in B$). This is the case if $B = N$ or if B is another algebraic data types, but it is also the case $B = I(N, c, c')$, for example. Compare Hedberg's coherence theorem [20], which is valid only for decidable identity types.

Although we do not justify types such as $I(N \to N, f, g)$ as a primitive identity type, we can of course still define it as $(\Pi x \in N)I(N, Ap(f, x), Ap(g, x)$, expressing the extensional equality of $f, g \in N \to N$. This type can be tested by following the manual for testing Π-types and I-types over the type N.

1.5 Impredicative type theory

We have explained how the meaning explanations for intuitionistic type theory are about testing judgements by running programs interactively, that is, by generating input, and observing results. We repeat the process for several inputs. In order to test higher order functions we need to consider repeated interactive testing in the style of game semantics.

Can we provide similar meaning explanations for other theories? That is, can we describe the meaning of the judgements of the theory by running programs interactively, generating input, observing results, and thereby corroborating or falsifying results? Is this what constructive validity is all about?

In this section we shall ask ourselves what such meaning explanations for impredicative type theory would be like. We would like to write testing manuals for impredicative type theories such as

- System F of Girard [15]?
- The Calculus of Constructions of Coquand and Huet [10]?
- The Calculus of Inductive Constructions of Coquand and Paulin [32], the theory of the Coq-system [7]?

These systems have to my knowledge only been analyzed meta-mathematically. For example, Girard [15] showed that System F is strongly normalizing. In this proof he relied on impredicative aspects of his meta-language: classical set theory.

We shall now ask ourselves what a pre-mathematical testing semantics would be like for these systems? Note that they have real users, especially the last one (the Coq-users). What do these users expect when they "run" their programs?

Let us here consider Coquand and Huet's pure Calculus of Constructions [10]. This theory has an impredicative universe U (usually called *Prop*, the universe of impredicative propositions). It is impredicative since we can form the cartesian product of a family of small types indexed by an arbitrary (not necessary small) type:

$$\frac{A\ type \quad x \in A \vdash B \in U}{(\Pi x \in A)B \in U}$$

In particular we can put $A = U$ and define a new element of U by quantification over all elements in U including the one we are defining.

Contrast this to the type of small types in Martin-Löf type theory. This is predicative: we can only form a family of small types indexed by an arbitrary small type:

$$\frac{A \in U \quad x \in A \vdash B \in U}{(\Pi x \in A)B \in U}$$

Unlike in Martin-Löf type theory,we can thus define the type of natural numbers as the type of Church numerals in the Calculus of Construction as follows:

$$N = (\Pi X \in U)X \to (X \to X) \to X \in U$$

We can also define the type of identity proofs over an arbitrary type (Leibniz equality):

$$I(A, a, b) = (\Pi X \in A \to U)X(a) \to X(b) \in U$$

Since they can be encoded, the pure Calculus of Constructions does not have primitive types N and $I(A, a, b)$. This coding can be done for a very general class of inductive types. However, some aspects of this encoding are unsatisfactory. As a consequence Coquand and Paulin decided that primitive inductive types and families should be added. The resulting theory is the Calculus of Inductive Constructions, the core of the Coq system [7].

A testing manual for the Calculus of Constructions based on the evaluation of closed expressions

As we saw above, there is great similarity between Martin-Löf type theory and the Calculus of Constructions, except that the latter

- only has types U and $(\Pi x \in B)C$ and no primitive data types $N, I(A, a, b), \ldots$;
- has U which is closed under impredicative Π.

Let us first try to make a minimal modification of the testing manual for Martin-Löf type theory to accommodate the types of the impredicative universe U. The difference appears in the test for elements of U. The only rule is:

$$\frac{A \Rightarrow U \quad a \Rightarrow (\Pi x \in B)C \quad B\ type \quad x \in B \vdash C \in U}{a \in A}$$

Hence, there is no base case! It is not clear how the testing procedure would ever stop.

Testing based on normalization of open expressions,

Martin-Löf type theory is a functional programming language. A user will run a program in much the same way as a user of an ordinary lazy functional language such as Haskell, that is, the user will evaluate *closed* expressions to canonical form. (When we base Martin-Löf type theory on the theory expression this corresponds to evaluating closed expressions of arity ι to canonical form.)

However, a user of the Calculus of Constructions will instead evaluate programs to full normal form, for example when computing with Church numerals. To obtain this full normal form we need to evaluate expressions under λ, that is, we need to evaluate *open* expressions. We will therefore consider a testing manual for the Calculus of Constructions based on the evaluation of open expressions.

Evaluation of open expressions in Martin-Löf type theory

Incidentally, Martin-Löf has recently considered introducing meaning explanations based on the evaluation of open expressions for intensional intuitionistic type theory. The abstract of his talk *Evaluation of open expressions* (given at the *Symposium on Programming, Types, and Languages* in Gothenburg in March 2009) states his aim as follows:

> The informal, or intuitive, semantics of type theory makes it evident that closed expressions of ground type evaluate to head normal form, whereas metamathematics, ..., is currently needed to show that expressions which are open or of higher type can be reduced to normal form. The question to be discussed is: Would it be possible to modify the informal semantics in such a way that it becomes evident that all expressions, also those that are open or of higher type, can be reduced to full normal form?

The aim is to provide meaning explanations for intensional type theory which do not validate the rule of equality reflection, and which match the type-checking algorithm for intensional type theory.

Testing manual for the Calculus of Constructions

The terms are

$$a ::= U \mid (\Pi x \in a)a \mid (\lambda x)a \mid a(a) \mid x$$

Note that this is the syntax of the ordinary untyped lambda calculus extended with U and Π. We do not employ the theory of expressions here. These terms will be evaluated to open weak head normal forms:

$$v ::= U \mid (\Pi x \in a)a \mid (\lambda x)a \mid x(a, \ldots, a)$$

where we use the abbreviation $x(a_1, \ldots, a_n) = x(a_1) \cdots (a_n)$. Note that a in the above productions of values range over arbitrary terms.

There is only one computation rule

$$\frac{c \Rightarrow (\lambda x)b \quad b[a/x] \Rightarrow v}{c(a) \Rightarrow v}$$

in addition to the rule that a canonical term has itself as value

$$v \Rightarrow v$$

To test the hypothetical judgement $\Gamma \vdash A\ type$ we evaluate the open expression A. We need to consider three possibilities:

$$\frac{A \Rightarrow (\Pi y \in B)C \quad \Gamma \vdash B\ type \quad \Gamma, y \in B \vdash C\ type}{\Gamma \vdash A\ type} \qquad \frac{A \Rightarrow U}{\Gamma \vdash A\ type}$$

$$\frac{A \Rightarrow x(b_1, \ldots, b_n) \qquad \Gamma \vdash b_1 \in B_1 \quad \cdots \quad \Gamma \vdash b_n \in B_n}{\Gamma \vdash A\ type}$$

In the last rule the type of x in Γ is $D \Rightarrow (\Pi y_1 : B_1) \cdots (\Pi y_n : B_n)U$.

To test the hypothetical judgement $\Gamma \vdash a \in A$ we evaluate the open expressions a and A:

$$\frac{A \Rightarrow U \qquad a \Rightarrow (\Pi y \in B)C \qquad \Gamma \vdash B\ type \qquad \Gamma, y \in B \vdash C \in U}{\Gamma \vdash a \in A}$$

$$\frac{A \Rightarrow (\Pi y \in B)C \quad a \Rightarrow (\lambda y)c \quad \Gamma, y \in B \vdash c \in C}{\Gamma \vdash a \in A}$$

$$\frac{a \Rightarrow x(b_1, \ldots, b_n) \quad \Gamma \vdash b_1 \in B_1 \quad \cdots \quad \Gamma \vdash b_n \in B_n \quad \Gamma \vdash C[b_1/y_1, \ldots, b_n/y_n] = A}{\Gamma \vdash a \in A}$$

In the last rule the type of x in Γ is $D \Rightarrow (\Pi y_1 : B_1) \cdots (\Pi y_n : B_n)C$.

We leave it to the reader to write the testing manual for the hypothetical judgements $\Gamma \vdash A = A'$ and $\Gamma \vdash a = a' \in A$.

Note that there is no need for input generation. The meaning of hypothetical judgements is defined directly in terms of evaluation to open canonical forms. Note that this testing manual is nothing but the type-checking algorithm for the Calculus of Constructions! The correctness of the Calculus of Constructions thus simply amounts to the termination with appropriate canonical forms of the type-checking algorithm. There is circumstantial evidence that this algorithm is correct since it has been corroborated many times. However, there is not yet a ”well-founded” explanation of why this is the case. Nevertheless, many, but not all, researchers express certainty that it will indeed always terminate correctly.

1.6 Conclusion

To conclude, I will discuss some important distinctions.

Pre-mathematical versus meta-mathematical semantics

I have emphasized the difference between on the one hand (formal) meta-mathematical semantics, understood as the translation into another mathematical language (usually set theory) the meaningfulness of which is taken for granted, and on the other hand, pre-mathematical semantics, where the meaning is explained in terms of real world interactions with and observations of expressions and their computations. This difference is crucial, although I would not like to claim that it is unproblematic. Formality may not be a requirement of meta-mathematics in the usual sense, see for example the discussion by Kleene [21].

I believe pre-mathematical and meta-mathematical aspects are complementary. Meta-mathematical semantics can provide further insight into the pre-mathematical situation. But meta-mathematical semantics may also introduce extraneous issues which are not relevant for the pre-mathematical understanding. For example, it is not clear that the meta-mathematical work of turning the meaning explanations into a positive inductive definition in set theory has much foundational significance.

Game semantics versus realizability semantics

The meta-mathematical semantics in Section 3 does not match the pre-mathematical semantics in Section 4. This is because the realizability semantics is not sufficiently fine-grained to adequately capture the details of the testing manual in Section 4. It would be interesting to provide a meta-mathematical treatment of the testing manual which captures more such details, perhaps in the style of game semantics.

Input generation versus output computation

Martin-Löf's meaning explanations for intuitionistic type theory are based on the evaluation of expressions to canonical form, that is, they are based on the computation of output. One of the key ideas of this paper is that a dual discussion about methods for input generation is needed for meaning explanations of hypothetical judgement.

Judgement versus proposition

It is important to note that we test *judgements* and not *propositions* in type theory.

Molecular versus holistic view of meaning

Another important facet of the present notion of meaning is that it is *molecular* in the sense that the meaning of a judgement only depends on the meaning of its constituent parts. This is to be contrasted to the *holistic* view, where meaning depends on the whole formal system which the judgement is part of. Consistency is an example of a holistic property.

Validated by testing versus made evident by thinking

Prior discussions of meaning explanations have focussed on their capacity to make the rules of inference *evident*. However, to be evident is a subjective notion: something is evident to *somebody*. Here we emphasize that judgements can also be validated by testing, and that this is an objective notion. The result of the test does not depend on who executed the test. For further discussion of *epistemological* vs *ontological* aspects of truth and proof in intuitionistic logic the reader is referred to Prawitz' article in this volume [34].

Primary school computation of closed expressions versus secondary school computation of open expressions

Martin-Löf style meaning explanations are based on the computation of *closed* expressions. This is after all the primary notion of computation; normalization of open expressions is a secondary notion of computation, although it may be relevant for meaning explanations of impredicative type theory.

Meaning as testing and meaning as use

The notion of a user occupies a central stage in the testing view of meaning explanations. It seems to fit well into Wittgenstein's "meaning-as-use" paradigm which has been further developed by Dummett [11] and Prawitz [35] in connection with the philosophical basis of intuitionism.

Some historical notes

This is not the place to trace the historical origin of the idea that testing is fundamental for the meaning of logic. I would just like to mention that Gödel's Dialectica interpretation [16] is a proof as programs interpretation where the correctness is justified in terms of tests. However, note that the Dialectica interpretation differs from Kleene realizability. The game interpretation of logic goes back to Lorenzen [23]. It seems that Lorenzen's dialogical semantics for intuitionistic predicate logic would be closely related to a potential game interpretation of intuitionistic type theory.

Acknowlegement

I would like to express my deep gratitude to Per Martin-Löf for his profound ideas and for many discussions and much help over the years. The present paper owes a lot to these discussionson, for example, on the nature of the meaning explanations, on the distinction between the pre-mathematical and the meta-mathematical, and on the meaning of induction.

I would also like to thank Erik Palmgren and an anonymous referee for useful feedback on a preliminary version of this paper. The paper is based on a talk given several times. I am grateful for interesting comments by many people who attended these talks, for example, Andreas Abel, Peter Aczel, Pierre Clairambault, Thierry Coquand, Peter Hancock, Bengt Nordström, Andrew Pitts, Gordon Plotkin, Michael Rathjen, Dag Prawitz, Peter Schroeder-Heister, Helmut Schwichtenberg, Anton Setzer, and Sören Stenlund.

References

1. P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland, 1977.
2. P. Aczel. The strength of Martin-Löf's type theory with one universe. In S. Miettinen and J. Väänanen, editors, *Proceedings of the Symposium on Mathematical Logic (Oulu 1974)*, pages 1–32, 1977. Report No 2 of Dept. Philosophy, University of Helsinki.
3. P. Aczel. *Frege Structures and the Notions of Proposition, Truth, and Set*, pages 31–59. North-Holland, 1980.
4. S. F. Allen. A non-type-theoretic definition of Martin-Löf's types. In *Proceedings of Second IEEE Symposium on Logic in Computer Science*, pages 215–224, 1987.
5. S. F. Allen. *A Non-Type-Theoretic Semantics for Type-Theoretic Language*. PhD thesis, Cornell University, 1987.
6. M. Beeson. Recursive models for constructive set theories. *Annals of Mathematical Logic*, 23, 1982.
7. Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

8. K. Claessen and J. Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*, volume 35.9 of *ACM SIGPLAN Notices*, pages 268–279. ACM Press, September 2000.
9. T. Coquand. A semantics of evidence for classical arithmetic. *Journal of Symbolic Logic*, 60(1):325–337, 1995.
10. T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.
11. M. Dummett. The philosophical basis of intuitionistic logic. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, pages 5 – 40. North Holland, 1973.
12. P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2):525–549, June 2000.
13. P. Dybjer and A. Setzer. A finite axiomatization of inductive-recursive definitions. In J.-Y. Girard, editor, *Typed Lambda Calculi and Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 129–146. Springer, April 1999.
14. P. Dybjer and A. Setzer. Indexed induction-recursion. *Journal of Logic and Algebraic Programming*, 2006.
15. J.-Y. Girard. Une extension de l'interpretation de Gödel à l'analyse, et son application à l'èlimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, editor, *Proceedings 2nd Scandinavian Logic Symposium*, pages 63–92. North Holland, 1971.
16. K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finites Standpunktes. *Dialectica*, pages 280–287, 1958.
17. S. Hayashi. PA/LCM home page. http://www.shayashi.jp/PALCM/.
18. S. Hayashi and H. Nakano. *PX: a Computational Logic*. MIT Press, 1989.
19. S. Hayashi, R. Sumitomo, and K. Shii. Towards the animation of proofs - testing proofs by examples. *Theoretical Computer Science*, 272(1-2):177–195, 2002.
20. M. Hedberg. A coherence theorem for Martin-Löf's type theory. *Journal of Functional Programming*, 8(4):413–436, 1998.
21. S. C. Kleene. *Introduction to Meta-Mathematics*. North-Holland, 1952.
22. D. Knuth. Notes on the van Emde Boas construction of priority deques: An instructive use of recursion, March 1977. Memo sent to Peter van Emde Boas, see http://www-cs-faculty.stanford.edu/ uno/faq.html.
23. P. Lorenzen and K. Lorenz. *Dialogische Logik*. Wissenschaftliche Buchgesellschaft, Darmstadt, 1978.
24. P. Martin-Löf. A theory of types. Preprint, Stockholm University, 1971.
25. P. Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, pages 73–118. North Holland, 1975.
26. P. Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science, VI, 1979*, pages 153–175. North-Holland, 1982.
27. P. Martin-Löf. *Intuitionistic Type Theory: Notes by Giovanni Sambin of a Series of Lectures Given in Padua, June 1980*. Bibliopolis, 1984.
28. P. Martin-Löf. On the meaning of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1), 1996.
29. P. Martin-Löf. An intuitionistic theory of types. In G. Sambin and J. Smith, editors, *Twenty-Five Years of Constructive Type Theory*. Oxford University Press, 1998. Reprinted version of an unpublished report from 1972.
30. A. Miquel. The reasonable effectiveness of mathematical proof. In M. Quatrini and S. Tronon, editors, *Anachronismes logiques*, Logique, Langage, Sciences, Philosophie. Publications de la Sorbonne, 2010. To appear.
31. B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's Type Theory - an Introduction*. Oxford University Press, 1989.
32. C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *Proceedings Typed λ-Calculus and Applications*, pages 328–245. Springer-Verlag, LNCS, March 1993.

33. K. Popper. *Conjectures and Refutations: The Growth of Scientific Knowledge*. Routledge, 1963.
34. D. Prawitz. Truth and proof in intuitionism. This volume.
35. D. Prawitz. Meaning and proofs: on the conflict between classical and intuitionistic logic. *Theoria*, 43:11–40, 1977.
36. C. Runciman, M. Naylor, and F. Lindblad. SmallCheck and Lazy SmallCheck - automatic exhaustive testing for small values. In *Proceedings of the first ACM SIGPLAN symposium on Haskell*, pages 37–48, 2008.
37. A. Setzer. Extending Martin-Löf type theory by one Mahlo universe. *Archive for Mathematical Logic*, 39(3):155–181, 2000.
38. J. Smith. An interpretation of Martin-Löf's type theory in a type-free theory of propositions. *Journal of Symbolic Logic*, 49(3):730–753, 1984.