MTT-Semantics Is Model-Theoretic As Well As Proof-Theoretic[*]

Zhaohui Luo[†]
Royal Holloway, University of London
Email: *Zhaohui.Luo@hotmail.co.uk*

July 4, 2019

Abstract

In this paper, we argue that formal semantics based on modern type theories (MTT-semantics) is both model-theoretic and proof-theoretic, and hence has unique advantages as a semantic framework. Being model-theoretic, it provides a wide coverage of various linguistic features partly because the rich type structure in MTTs can be used effectively to represent various collections, playing a role as sets do in Montague's model-theoretic semantics. Being proof-theoretic, its foundational languages have a proof-theoretic meaning theory and provide a solid foundation for natural language reasoning using proof assistants.

After presenting the basic arguments, we shall then focus on further development of the first, and arguably less understood, aspect: MTT-semantics is model-theoretic. We shall develop a notion of signature to allow new forms of subtyping and definitional entries and show that such formal contextual tools support useful ways of representing incomplete possible worlds in semantic studies. We shall also develop a notion of belief collection and show, from a different angle, that MTTs provide powerful mechanisms for linguistic semantics.

[*]The viewpoint, as stated in the title, that the formal semantics in modern type theories (MTT-semantics for short) is both model-theoretic and proof-theoretic, was first put forward in the author's talk [51] at the 2014 conference on Logical Aspects in Computational Linguistics. Since then, there have been further developments of the idea and arguments, enriching the theme, on the one hand, and clarifying some misunderstandings, on the other. In this paper, we intend to give a more thorough and self-contained explication.

[†]Partially supported by the EU grant EUTypes (COST Action CA15123).

1 Introduction

MTT-semantics [52, 21] is formal semantics in the style of Montague's semantics [61], but in Modern Type Theories (MTTs)[1] rather than in simple type theory (and its set-theoretical models) [22, 61, 33]. MTTs have important features that traditional logical systems such as simple type theory do not have. Besides containing embedded logics based on the Curry-Howard principle of propositions as types [27, 38], most of MTTs have rich type structures including, to name a few, dependent types, inductive types and type universes (see later for more elaborations). In this paper, we shall argue that MTT-semantics has a unique important feature: it is both model-theoretic and proof-theoretic, and this has made MTTs a particularly nice framework for formal semantics. Being model-theoretic, MTT-semantics provides a wide coverage of various linguistic features and, being proof-theoretic, its foundational languages have proof-theoretic meaning theory based on inferential uses (appealing philosophically and theoretically) and it establishes a solid foundation for practical reasoning in natural languages on computers (appealing practically). Altogether, this strengthens the argument that MTT-semantics is a promising framework for formal semantics, both theoretically and practically.

MTTs are proof-theoretically specified. Usually, an MTT is presented as a natural deduction system where, in particular, every type constructor is specified by rules among which introduction rules are to declare how the type (formula) is inhabited (proved) and the elimination rule to specify what consequences can be derived under the assumption that the type is inhabited. These rules are harmonious and, as an important consequence, MTTs themselves have proof-theoretic semantics, as studied by logicians such as Gentzen [35], Prawitz [69, 68] and Martin-Löf [59, 60] and discussed by philosophers such as Dummett [31] and Brandom [11, 12], among others.[2] For example, Martin-Löf has studied and developed meaning theory and given a solid foundation for his type theory [59]. Therefore, MTT-semantics, the formal semantics in MTTs, is proof-theoretic in the sense that its foundational semantic languages have proof-theoretic meaning the-

[1]By MTTs, we refer to the family of formal systems such as Martin-Löf's intensional type theory (MLTT) [58, 62], the type theory in Coq (pCIC) [24] and the Unifying Theory of dependent Types (UTT) [43]. See §2 for more details.

[2]It is worth remarking that even if a logical system is specified by proof-theoretic rules, it can still fail to have a proper proof-theoretic semantics. The introduction and elimination rules must be in harmony as discussed by Dummett [31] and others. See §4.2 for further discussions.

ory based on inferential uses.

Furthermore, the fact that MTTs are proof systems with proof-theoretic semantics has another significant and practical consequence in NL reasoning based on MTT-semantics. In particular, this makes it possible for MTTs to be implemented in proof assistants such as Coq [24], Agda [3] and Lego/Plastic [56, 14] – computer-assisted reasoning systems that computer scientists have developed and successfully used for formalisation of mathematics and verification of computer programs. Therefore, MTT-semantics can be directly implemented in proof assistants that implement MTTs: for example, the MTT-semantics in type theory UTT has been implemented in Coq and Plastic and used for NL reasoning [17, 49].

That MTTs are proof-theoretically presented has led to a widely-held view that formal semantics based on MTTs is only proof-theoretic and, in particular, it is not model-theoretic.[3] This is mistaken! As will be argued, MTT-semantics is model-theoretic as well: this is the major theme to be explicated in the current paper. First of all, it is necessary for us to make clear that, by MTT-semantics being model-theoretic, we do not mean that an MTT can be given a set-theoretical semantics (e.g., in some categorical framework); instead, we mean that an MTT itself can be employed as a meaning-carrying language to give the model-theoretic semantics to Natural Language. That is, in MTT-semantics, an MTT can play the role of meaning-carrying language – in the traditional model-theoretic semantics, this is the role played by set theory. In other words, we argue that MTTs can well serve as a foundational semantic language to give meanings in themselves in a model-theoretic semantics.

We shall develop the theme that MTT-semantics is model-theoretic in two fronts: the first is to note that, like sets in Montague's semantics, types in MTTs are employed to represent collections[4] of objects (for example, semantic interpretations of common nouns), and the rich type structure in MTTs provides powerful means of representation for formal semantics. In other words, intuitively, types in MTTs are rich enough to play the role of representing collections, just as sets in Montague's semantics.

For instance, compared with the types e, t and arrow types $A \to B$ in simple type theory used in the Montagovian setting, MTTs contain many more type constructors for dependent types (e.g., Π-types and Σ-types), in-

[3]An exception is Ranta [73] (p.346) where he pointed out that it is a misunderstanding to think that formal semantics based on Martin-Löf's type theory (an MTT) is not model-theoretic.

[4]In this paper, the word *collection* is used to refer to informal entities, rather than their formal representations such as *sets* or *types*.

ductive types (e.g., types of numbers/lists/vectors/trees and disjoint union types), logical types for propositions (under the propositions-as-types principle), and type universes (types that contain types as objects). As we know, types are different from sets. For example, in MTTs type-checking is decidable; in other words, it is decidable whether $a : A$ – this is the basis for the principle of propositions as types and also for efficient implementations of proof assistants such as Coq based on MTTs. This is in contrast to the fact that the membership relation $s \in S$ (or whether something satisfies a predicate) is undecidable, since it is just a formula in FOL and its truth is undecidable. As we are going to demonstrate, in MTT-semantics, these types play an important role in representing different kinds of collections such as those given by CNs modified by various adjectives. As these types do not exist in simple type theory, one cannot represent various collections as types there by means only the simple arrow types. In this paper, we shall summarise how types are used to play various different representation roles in MTT-semantics to show the power of MTTs' rich type structure. This is the first facet to explicate that MTT-semantics is model-theoretic.

The second facet is that the contextual structures in MTTs provides useful means to support model-theoretic descriptions of incomplete possible worlds[5]. We shall describe some new developments in this respect that support model-theoretic descriptions. In particular, we introduce a notion of *signature* into MTTs: rather than the usual judgement forms $\Gamma \vdash J$ in type theory, the new judgements with signatures are of the form

(1) $\Gamma \vdash_{\Sigma} J$

where Σ is a signature which, like a context, is a sequence of entries, including the traditional membership entries of the form $c : A$ that declares that the constant c is of type A. Declaring a constant (or making other declarations), an entry in a signature cannot be abstracted by means of quantification or λ-abstraction. This is in contrast to entries in a context in type theory (Γ in the above judgement form (1)) that describe variables which can be abstracted.

The use of the notion of signature in type theory, as far as the author knows, first appeared in Edinburgh Logical Framework, where signatures with membership entries are used to describe a logical system [37].[6] We shall

[5]In this paper, the phrases such as *possible worlds* or *situations* are used informally without referring to any formal meanings of these words that may have been employed in the literature.

[6]Historically, signatures have been used in describing algebraic structures and, for

introduce signatures that do not just have the usual membership entries, but also contain the following two new forms of entries:

- $A \leq_\kappa B$, a subtyping entry that declares that A be a subtype of B via coercion κ.[7]
- $c \sim a : A$, a manifest entry that declares that c behaves exactly as the object a of type A.[8]

These forms of entries strengthen the power of signatures in representing (incomplete) possible worlds, even in cases where, for example, situations are infinite or involve more sophisticated phenomena. It can be shown that extending MTTs with signatures, which may contain entries of the above new forms as well as the membership entries, preserves their nice meta-theoretic properties such as strong normalisation and hence logical consistency. In this paper, however, we shall focus on representational issues by presenting examples to explain how signatures can be used to represent possible worlds, and only sketch the meta-theoretic results – their technical studies and detailed proofs are dealt with elsewhere (see, for example, [42, 40]).

We shall also propose a formal treatment of beliefs in MTTs that is capable of dealing with hyperintensional problems. Ranta [72] introduced a notion of belief context in type theory which can be adopted very usefully in various ways in formal semantics. However, it does not solve hyperintensional problems such as that of logical omniscience [26] since, under Ranta's notion of belief context, whether an agent believes in something is closed under derivability and hence, for example, if one believes in a proposition P, one believes in every proposition that is logically equivalent to P. Our treatment is more restrictive and reflects the idea that, given a collection of beliefs of an agent, a statement is believed by the agent if, and only if, it is intensionally equal to one of the existing beliefs in the agent's belief collection. This also shows, from a different angle, that the mechanisms in MTTs are rich and powerful in dealing with various kinds of phenomena in formal semantics.

example, more recently they were employed in describing many-sorted structures in the study of algebraic specifications [13]. However, it should be noted that the notion of signature in type theory is rather different from that in algebras, although they may be related informally.

[7]Here, a subtyping entry $A \leq_\kappa B$ is based on coercive subtyping, where $\kappa : (A)B$ is a function from type A to type B, as studied in [44, 57]. In this paper, we shall study its localisation (i.e., declaring them in signatures).

[8]Informally, this is to introduce entries that allow 'definitional' declarations, whose introduction into contexts by unit types and coercions was studied in [46] and, in this paper, we shall study their introduction into signatures.

It is worth pointing out that foundational semantic languages that are both model-theoretic and proof-theoretic were not available before the development of MTT-semantics or, at least, people have not recognised that there is such as possibility (in particular, set theory is not such a language since it is not proof-theoretic.) That MTT-semantics is both model-theoretic and proof-theoretic sheds a new light on the divide between model-theoretic semantics and proof-theoretic semantics and allows us to come up with a new perspective in studying formal semantics.

The main contributions of the current paper are two-fold. First, our main theme is to argue, for the first time as far as the author is aware, that MTT-semantics can be seen as both model-theoretic and proof-theoretic, in the sense that we have sketched and shall elaborate in the paper. As said, the model-theoretic characteristics of MTT-semantics is reflected in two facets: rich type structure for representations and contextual mechanisms for representing various linguistic phenomena, both unique for MTTs. However, as will be explained, the existing contextual mechanisms in MTTs have limitations and potential problems for such representations. Therefore, this is our second main contribution: we introduce signatures, which are more suitable than contexts for representations, and add two new forms of entries to signatures that allow us to overcome the limitations of the existing contextual mechanism that does not allow subtyping or manifest entries and to add new representational power to MTTs.

The structure of the paper is as follows. §2 gives an introduction to MTTs, summarising their historical development and introducing the embedded logic and various type constructors, and §3 briefly introduces MTT-semantics. In §4, we present the arguments that MTT-semantics is both model-theoretic and proof-theoretic, showing in §4.1 that the rich type structure of MTTs provides powerful means of representation in formal semantics, on the one hand, and in §4.2 that proof-theoretic semantics for MTTs establishes the basis for inferential semantics and computerised practical reasoning, on the other. §5 studies the extension with signatures that may contain subtyping and manifest entries and their uses in semantics, both in its basic form in §5.1 and in its advanced form in §5.2. In §6, an intensional notion of belief collection is introduced to show, from another angle, that MTTs have very rich type structures, and in this case, powerful logical types for representations in semantic studies. Some concluding remarks on related and future work are given in Conclusion.

2 Modern Type Theories

The development of type theory started from Russell's work on Ramified Theory of Types [79, 75], aiming at solving the problem of paradoxes as found in Cantor's naive set theory for the foundation of mathematics. It was later developed into Simple Type Theory by Ramsay [71] and Church [22], which has been used by Montague in Intensional Logic (IL) as an intermediate language for model-theoretic semantics of natural language [61, 33].

The studies of simple type theory have been driven in search of foundational languages for classical mathematics. In the 1970s, various researchers studied foundational languages for constructive rather than classical mathematics. Besides other systems, Martin-Löf's type theory [58, 59] has been widely studied and applied to the foundations of mathematics, computer science and linguistic semantics. It contains powerful typing mechanisms such as dependent typing, inductive typing and type universes. Its study, together with that of simple type theory, has led to the development of a family of (intensional) type theories called Modern Type Theories (MTTs), including predicative type theories such as Martin-Löf's intensional type theory (MLTT) (see its description in Part III of [62]) and impredicative type theories such as the calculus of constructions (CC) [25] and the Unifying Theory of dependent Types (UTT) (see Chapter 9 od [43]).

In computer science, MTTs have been implemented in proof assistants and used in applications to formalisation of mathematics and verification of programs; for example, (some version of) MLTT has been implemented in Agda [3], the system pCIC (an extension of CC) in Coq [24] and UTT in Lego/Plastic [56, 14].

It is worth remarking that, although formalising constructive mathematics was the main motivation of the early development of Martin-Löf's type theory, it is not the case that modern type theories can only be employed constructively.[9] Put in another way, powerful typing is not monopolised by constructive mathematics or constructive reasoning; instead, it can be used in much wider applications such as linguistic semantics (for example, MTT-semantics considered in this paper).

In this section, we shall introduce MTTs, some aspects of which are detailed because they are used in latter parts of the paper. It will focus on necessary background information and provide notational conventions as

[9]In this respect, one may be interested in the work on Logic-enriched Type Theories (LTTs) [1, 34] and a logical framework of LTTs [45, 2] for reasoning in various different logical foundations, including classical as well as intuitionistic systems.

well.

2.1 Judgements

In an MTT, the notion of judgement is the most basic.[10] A typical form of judgement is:

(2) $\Gamma \vdash a : A$

which states that 'object a is of type A in context Γ'. Types in MTTs can be used to represent collections of objects in a model-theoretic sense, although they are syntactic entities, often inductively defined in MTTs. In other words, intuitively, the above judgement states that, in the environment represented by context Γ, the object a stands for an object in the collection represented by A. It may be worth noting that, unlike ordinary logical systems, a judgement of the form (2) in an MTT is not a logical proposition, which is a type according to the propositions-as-types principle.

Contexts in MTTs are formal mechanisms that, in logical terms, stand for assumptions of environments, and can be used, in semantic studies, to represent incomplete possible worlds in which statements are made.[11] Currently, in type theory, the notion of context only allows membership entries: the context Γ in (2) is a finite sequence (3) of variable-type pairs in which one assumes that x_i be of type A_i ($i = 1, ..., n$).

(3) $x_1 : A_1,\ x_2 : A_2,\ ...,\ x_n : A_n,$

Note that those types in a context may depend on 'previous' variables: $x_1, ..., x_{i-1}$ may occur free in A_i. For instance, if Man is the type of men and $\mathit{handsome} : Man \to \mathit{Prop}$ a predicate over Man, then we can consider a context that contains the entries in (4) as a part, which assumes that x is an (arbitrary) handsome man (i.e., x is an arbitrary man and y is an arbitrary proof that x is handsome):

(4) $x : Man,\ y : \mathit{handsome}(x)$

[10]In §5, judgement forms will be extended with signatures.

[11]In linguistics, the term *context* is used to refer to those objects or entities which surround a communicative event, providing resources for its appropriate interpretation. In MTT-semantics, one may consider contexts in MTTs as formal representatives of some of those linguistic contexts. However, it is an open issue as to what extent the formal notion of context in MTTs can approximate the informal linguistic notion and this is beyond our discussion in the current paper. Using contexts to represent possible situations in real world was studied by Ranta in [72], among others. We shall extend the notion in this paper with new entries which are particularly useful for formal semantics.

We usually call an entry of the form $x : A$ a membership entry. In the second entry of (4), the type is the logical proposition $handsome(x)$, where x occurs free; in other words, $handsome(x)$ is a type depending on x.

Remark In this paper, unless explicitly stated otherwise, we usually give examples (for instance, (4)) in the impredicative type theory UTT [43], where there is a type $Prop$ of all logical propositions (see §2.2.3). $\square$

2.2 Types

A type in a type theory informally stands for a collection of entities. In simple type theory, examples of types are the base types e of entities and t of truth values and function types such as $e \to t$. In modern type theories, there are many different types which can be formed by various type constructors other than $\to$. In this subsection, we shall give informal but precise descriptions of these types – their formal rules will mostly be given in the appendixes except Σ-types of dependent pairs in §2.2.1, which we use as an example for illustrations.

Like those in simple type theory, types in MTTs are manageable, as compared with sets in set theory. For example, in MTTs, type-checking is decidable: for any t and T, it is mechanically determinable whether t is an object of type T (i.e., whether $t : T$). This is different from set theory where, for example, it is in general undecidable whether s is a member of set S (i.e., whether $s \in S$). Decidability of type-checking is fundamental; for example, it is the basis for the propositions-as-types principle.[12] Having a rich type structure with such properties is one of the reasons that MTTs offer a nice semantic framework with advantages compared with the traditional one in simple type theory.

2.2.1 Σ-types of Dependent Pairs

Modern type theories contain *dependent types* and a Σ-type is a typical example of a dependent type. If A is a type and B is a family of types that depends on objects of type A, then $\Sigma(A, B)$, or sometimes written as $\Sigma x{:}A.B(x)$, is a type, consisting of pairs (a, b) such that a is of type A and b is of type $B(a)$. Σ-types are associated with the projection operators π_1 and π_2 so that, for every (a, b) of type $\Sigma(A, B)$, $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$.

[12]This is not difficult to see: given a proposition P and a term p, it should be decidable whether p is a proof of P, i.e., whether $p : P$. Note that this is different from deciding whether an arbitrary P is true, which is undecidable.

Here, the dependency of B over objects of A is the crucial aspect that makes $\Sigma(A, B)$ a dependent type. Let's use an example to explain: assuming that Man is the type of men and $handsome : Man \to Prop$ is the predicate expressing that a man is handsome. Then, $\Sigma(Man, handsome)$ is the type of handsome men or, more precisely, the type that represents the collection of pairs (m, p) of a man m and a proof p that m is handsome. Note that the predicate *handsome* is a family of propositions (and, by propositions as types, a family of types): $handsome(x)$ is a proposition/type for each man $x : Man$. In other words, $handsome(x)$ depends on x and $\Sigma(Man, handsome)$ is a dependent type. In particular, for (m, p) of type $\Sigma(Man, handsome)$, p's type $handsome(m)$ depends on m.

The ordinary product type $A \times B$ of non-dependent pairs is a special case of the dependent Σ-type. Put in another more precise way, when $B(x)$ is a constant type (i.e., always the same type no matter what x is), the Σ-type $\Sigma(A, B)$ degenerates into a product type of non-dependent pairs that we are familiar with. For instance, if $Woman$ is the type of women, then the product type $Man \times Woman$ is that of ordinary pairs (m, w) such that $m : Man$ and $w : Woman$. This ordinary product type $Man \times Woman$ is the same as the Σ-type $\Sigma(Man, W)$, where the constant family of types $W(x)$ is always equal to $Woman$.

Σ-types are inductively defined. Formally, every inductive type constructor is specified by means of inference rules in Natural Deduction, called formation rule, introduction rules, elimination rules and computation rules. For Σ-types, these rule are given in Figure 1.

- The formation rule (Σ) tells us how a Σ-type can be legally formed.
- The introduction rule $(Pair)$ tells us that a Σ-type consists pairs as objects.[13] Note that, as explained above, the second component b of the pair is of type $B(a)$, which depends on the first component a – this is type dependency. To explain in terms of the example of the above, in an object pair (m, p) of type $\Sigma x{:}Man.handsome(x))$, p must be a proof of m being handsome (not of something else).
- The elimination rules $(Proj_1)$ and $(Proj_2)$ specify the projection operators which get their meanings by means of the computation/definition rules $(Conv_1)$ and $(Conv_2)$ that stipulate that, when applied to a pair

[13]Here, we have simplified the formality a bit for understandability: formally, the pair (a, b) needs to contain typing information so that it can be deduced during type inference. We omit the details here.

$$(\Sigma) \qquad \frac{\Gamma \vdash A\ type \quad \Gamma,\ x{:}A \vdash B\ type}{\Gamma \vdash \Sigma x{:}A.B\ type}$$

$$(Pair) \qquad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : [a/x]B \quad \Gamma, x{:}A \vdash B\ type}{\Gamma \vdash (a,b) : \Sigma x{:}A.B}$$

$$(Proj_1) \qquad \frac{\Gamma \vdash p : \Sigma x{:}A.B}{\Gamma \vdash \pi_1(p) : A}$$

$$(Proj_2) \qquad \frac{\Gamma \vdash p : \Sigma x{:}A.B}{\Gamma \vdash \pi_2(p) : [\pi_1(p)/x]B}$$

$$(Conv_1) \qquad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : [a/x]B}{\Gamma \vdash \pi_1(a,b) = a : A}$$

$$(Conv_2) \qquad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : [a/x]B}{\Gamma \vdash \pi_2(a,b) = b : [a/x]B}$$

Figure 1: The rules for Σ-types.

of a Σ-type, the first and second projections are equal to the first and second components, respectively.

A Σ-type $\Sigma(A, P)$ can be used to represent subset constructions when its second parameter P is a predicate. This makes use of the idea of propositions as types – for any $x : A$, $P(x)$ is a proposition and hence a type. Therefore, $\Sigma(A, P)$ is a type, according to the above formation rule. Also, in general, Σ-types can be regarded as types of tuples since its formation can be nested. A nested Σ-type $\Sigma x_1{:}A_1\ \Sigma x_2{:}A_2\ ...\ \Sigma x_{n-1}{:}A_{n-1}.\ A_n$ can be written by the following notation and, intuitively, its objects are n-tuples of terms of the respective types.

$$\left\{ \begin{array}{lll} x_1 & : & A_1 \\ x_2 & : & A_2 \\ ... & & \\ x_n & : & A_n \end{array} \right\}$$

2.2.2 More Type Constructors

Here, we briefly introduce several other type constructors and examples of their uses in MTT-semantics. Like Σ-types above, they are all inductive types and their formulations follow a similar pattern as that for Σ-types described above in §2.2.1. We will give the formal rules for each type constructor in Appendix A.

Π-types of Dependent Functions. Another typical dependent type constructor is that of Π-types of dependent functions. When A is a type and B is a family of types that depends on objects of A, then $\Pi(A, B)$, sometimes written as $\Pi x{:}A.B(x)$, is the type of λ-functions f such that, for any $a : A$, $f(a)$, the application of f to a, is of type $B(a)$. Note that the type $B(a)$ of the application $f(a)$ is dependent on the input a – that makes the Π-type a dependent type. A Π-types degenerates to the ordinary function type $A \to B$ in the non-dependent case – when its second parameter is the constant type B. For instance, the type of predicates over Man is the function type $Man \to Prop$, where $Prop$ is the type of all logical propositions (see §2.2.3). The formal rules for Π-types can be found in Appendix A.1.

Disjoint Union Types. When A and B are types, so is the disjoint union type $A + B$, whose objects are either $inl(a)$ for $a : A$ and $inr(b)$ for $b : B$. Disjoint union types have been proposed to give interpretations of common nouns when modified by privative adjectives [48, 16, 19]. For instance, for the types G_R of (real) guns and G_F of fake guns, one can use the following disjoint union type $G = G_R + G_F$ to represent the collection of all guns (see §4.1 for details). The formal rules for disjoint union types can be found in Appendix A.2.

Type Universes. A universe is a type of types. Theoretically, the introduction of a universe provides extra power that allows one to formalise some concepts in type theory which cannot be done without universes. Practically, universes are very useful since they make it possible to treat various collections of types as internal totalities and, combined with Π-types, they help to represent concepts polymorphically or generically – let's call it *Π-polymorphism.* In MTT-semantics, examples of universes include CN, the universe of (the interpretations of) common nouns [52], and *LType*, the universe of 'linguistic types', introduced in studying coordination [15]. (See §4.1 for examples uses of universes and Π-polymorphism.)

2.2.3 *Prop* and Logical Propositions

Modern type theories have embedded logics based on the Curry-Howard principle of propositions as types [27, 38], with explicit proof terms of propositions, which are types in the type theory. Let's consider a simple example:

the logical proposition A in (5) can be proved by the term p in (6); in other words, (7) is a correct judgement stating that p is a proof term of A (formally, one says that the judgement (7) can be derived):

(5) $A = \forall P : Nat \to Prop\ \forall x : Nat.\ P(x) \supset P(x)$

(6) $p = \lambda P : Nat \to Prop\ \lambda x : Nat\ \lambda y : P(x).\ y$

(7) $p : A$

A logical formula, represented as a type, is true if, and only if, there exists a proof of the corresponding type: in the above example, A is true since there is p which is a proof of A.

Prop, as found in impredicative type theories like UTT, is an example of a universe – the type of all logical propositions. When the second parameter of a Π-type is a predicate $P : A \to Prop$, the Π-type $\Pi(A, P)$, sometimes written as $\forall x{:}A.P(x)$, stands for the proposition of universal quantification. For example, the sentence (8) can be represented as (9), where *handsome* and *ugly* are predicates over Man. Using the notation ∀ for Π, (9) can be written as (10) as well.

(8) If a man is handsome, he is not ugly.

(9) $\Pi x{:}Man.\ handsome(x) \to \neg ugly(x)$

(10) $\forall x{:}Man.\ handsome(x) \to \neg ugly(x)$

The formal rules of the logical universe *Prop* can be found in Appendix A.3. Note that every proposition is a type (see (*Prop*2) in AppendixA.3) and that is why we have used predicates to form, for example, Σ-types: in order to form $\Sigma x{:}Man.handsome(x)$, we would need $handsome(x)$ be a type (see rule (Σ) in Figure 1) – but it is a proposition and, by (*Prop*2) in AppendixA.3, it is a type.

Furthermore, when both parameters for Π are logical propositions, the non-dependent $P \to Q$ is the proposition that P implies Q. In general, in an impredicative type theory, one can define all the other logical operators such as implication, conjunction, disjunction, negation and existential quantifier by means of ∀. This was studied and noticed by several researchers when studying second-order logics (see, for example, [67]) and the definitions can be found in, for example, Chapter 5 of [43]. For instance, the conjunction operator can be defined as (11).

(11) $P \wedge Q =_{\mathrm{df}} \Pi X{:}Prop.\ (P \to Q \to X) \to X$

Here is another example: we can use Π to define the equality between any two objects of the same type. For any type A and any two objects $a,\ b : A$,

	Example	Montague semantics	Semantics in MTTs
CN	man, human	$man,\ human : e \to t$	$Man,\ Human : Type$
IV	talk	$talk : e \to t$	$talk : Human \to Prop$
ADJ	handsome	$handsome : (e \to t) \to (e \to t)$	$handsome : Man \to Prop$
MCN	handsome man	$handsome(man)$	$\Sigma m : Man.\ handsome(m)$
S	A man talks	$\exists m : e.\ man(m)\ \&\ talk(m)$	$\exists m : Man.\ talk(m)$

Figure 2: Examples in formal semantics.

the proposition that a and b are equal can be defined as (12).

(12) $(a =_A b) =_{\mathrm{df}} \Pi P : A \to Prop.\ P(a) \to P(b).$

The equality thus defined is called the Leibniz equality; it is intensional in the following two senses: (1) logical equivalence does not imply Leibniz equality: i.e., it is possible that $P \neq_{Prop} Q$ while $P \equiv Q$; and (2) functional extensionality does not hold: for example, for $f,\ g : A \to B$, it is possible that $f \neq_{A\to B} g$, although it is the case that $\forall x{:}A.\ f(x) =_B g(x)$. Leibniz equality will be used in §6 to propose a formalisation of belief collections.

3 MTT-Semantics: a Brief Introduction

In this section, we give a brief introduction to formal semantics in modern type theories (MTT-semantics) [52, 21]. The development of MTT-semantics is a part of a wider research endeavour by many researchers who have recognised the potential advantages of rich type structures in constructing formal semantics [77, 72, 54, 9, 23, 28, 74, 8, 36]. MTT-semantics is in the style of Montague's semantics [61] but, while Montague's semantics is based on Church's simple type theory [22, 33], MTT-semantics is based on modern type theories, as introduced above in §2.

In Figure 2, we give some basic examples to illustrate how linguistic categories are interpreted in MTTs and compare them to those in Montague's semantics.

One of the key differences between MTTs and simple type theory is that MTTs have rich type structures with many types, much richer than those in simple type theory. Because of this, in such a rich type system, common nouns can be interpreted as types rather than predicates (see, for example, Ranta's proposal on this in Martin-Löf's type theory [72]). We can call this the CNs-as-types paradigm: it is different from Montague's semantics, which adopts the CNs-as-predicates paradigm, where CNs are interpreted as predicates of type $e \to t$ (and so are verbs and adjectives).[14] For instance, consider the CN 'book': it is interpreted in Montague's semantics as a predicate of type $e \to t$, while in MTT-semantics, it is interpreted as a type $Book$.

It has been argued that interpreting CNs as types rather than predicates has several advantages including, for example, better treatment of selectional restriction by typing, compatible use of subtyping in semantic interpretations and satisfactory treatment of some advanced linguistic features such as copredication (see, for example, [47, 20] for further details). Because CNs are interpreted as types, verbs and adjectives are interpreted as predicates over the types (eg, $[\![human]\!]$) that interpret the domains in which they are meaningful: examples are given in Figure 2.

Note that subtyping is crucial for the formal semantics in MTTs. For instance, consider the MTT semantics of the sentence 'A man talks' in Figure 2: for m of type $[\![man]\!]$ and $[\![talk]\!]$ of type $[\![human]\!] \to Prop$, $[\![talk]\!](m)$ is only well-typed because we have that $[\![man]\!]$ is a subtype of $[\![human]\!]$. For MTTs, coercive subtyping as studied in [44, 57] is an adequate framework to be employed for formal semantics [52].

Several remarks may be made for MTT-semantics.[15]

- Since logical propositions are used to interpret sentences, the usual simple inferences follow as intended. For instance, we have that the interpretation of (15) is implied by that of (13), but not by that of (14), and this follows directly from the use of $\wedge$ and $\vee$ to interpret (13) and (14), respectively. (See, for example, [15] for explanations of how coordination can be done in MTT-semantics.)

  (13) John can both read and speak Dutch.

  (14) John can both read and speak Dutch.

  (15) John can both read Dutch.

- Just like Montague's semantics, MTT-semantics is compositional in the sense that, for example, the meaning of a sentence is obtained functionally from those of its components. This can be understood

[14]In the Montague semantics whose underlying logic can be regarded as 'single-sorted' (it only has one type e of all entities although, strictly speaking, there is another base type t of truth values). In contrast, MTTs can be regarded as 'many-sorted' logical systems where there are many types (see §2).

[15]Thanks for a reviewer to get attention of these points to the author.

by just saying that, for example, the type constructors can be applied to any types and any families of types, which may be used to give semantics of new entities. For instance, we may use Σ-types $\Sigma x{:}A.B(x)$ to interpret intersective adjective modifications (see §4.1) – the CN semantically represented by A modified by an intersective adjective semantically represented by B. These As and Bs are open for new CNs and new intersective adjectives.

- Some ambiguities can also be captured in MTT-semantics. For instance, with $read : Editor \to Manuscript \to Prop$,[16] the sentence (16) is ambiguous with two possible readings, whose MTT-interpretations are (17) and (18), respectively, where the former is a ∃∀-formula and the latter a ∀∃-formula.

  (16) Some editors read every manuscript.
  (17) $\exists x{:}Editor \forall y{:}Manuscript.\ read(x, y)$
  (18) $\forall y{:}Manuscript \exists x{:}Editor.\ read(x, y)$

4 MTT-Semantics Is Both Model-Theoretic and Proof-Theoretic

In this section, we shall present our arguments that formal semantics in modern type theories is both model-theoretic and proof-theoretic, showing that the framework has unique advantages: being model-theoretic, it provides a wider coverage in semantic treatments of linguistic features and, being proof-theoretic, it has inferential meaning theories and supports practical reasoning on computer-assisted proof assistants.

4.1 Model-Theoretic Characteristics: Types for Representation

The model-theoretic characteristics of MTT-semantics can be considered in terms of two aspects. First, types in MTTs can be employed to represent collections (just like sets in set theory) and, since MTTs have a rich type structure, their types provide powerful means for representation in linguistic semantics. Secondly, the notion of context or signature, as a part of

[16]Here, *read* can have a more general type, for example, its first argument can be any human (of type $Human$), but since $Editor \leq Human$, *read* of such a more general type is of type $Editor \to Manuscripot \to Prop$. So, subtyping is essential – see our remark on subtyping above.

judgements in MTTs, is extremely useful (and can be extended further to be more suitable) in describing contextual information that is often found difficult to be captured in formal semantics without set-theoretical means.

None of the above is available in traditional logical systems such as first-order logic or simple type theory. First, the type structure in MTTs are much richer than that of simple type theory: not just the base types and arrow types, MTTs have other type constructors such as those summarised in §2.2 which can be used effectively to represent various collections such as those corresponding to different CNs and to provide useful formal tools such as Π-polymorphism in constructing formal semantics. Secondly, the contextual mechanisms in MTTs (contexts and, as we'll study and develop in §5, signatures) are very useful in representing incomplete or partial worlds in semantic studies and these contextual mechanisms are not available in traditional logical systems, either.

In this subsection, after a discussion on the representational facet of model-theoretic semantics, we shall focus on the first aspect of the model-theoretic characterisatics of MTT-semantics, while the second aspect will be studied in §5.

Model-theoretic Semantics. Model-theoretic semantics has been the dominating formal semantics in linguistics since the pioneering work by Montague in the late 60s and early 70s [61] where, with simple type theory as an intermediate language, Tarski's work in set-theoretical model theory has been applied to natural language semantics.[17] It is an accepted approach by most linguists, partly for historical reasons[18] and partly for the reasons to be elaborated in the following two paragraphs.

An important virtue of model-theoretic semantics is that it is *representational*.[19] For instance, in Montague's semantics that employed simple type

[17]There are other approaches to model-theoretic semantics including, for example, situation semantics [6] and variants of Montague's semantics. For our discussions, however, they do not make differences and hence we shall use Montague's semantics as a typical representative.

[18]Partee has once expressed her view on why model-theoretic (possible worlds) semantics is popular among linguists: 'Linguists tend to be instrumentalists about metaphysics, in the sense that they will tend to judge competing foundational theories more by their fruitfulness in helping to lead to insightful explanations of linguistic phenomena than by any other kinds of arguments' ([64], p.107).

[19]We use the term 'representational' without any historical connotations and, especially, it does not refer to either the approaches often classified as mental representation-based theories of meaning or the so-called 'representationalist' approaches to semantics (see, for example, Partee [64]). We do not use the term in such a way and, in particular, do not

theory (and its models in set theory), where representations (or sometimes called denotations) are given as meanings of natural language sentences and phrases. For instance, when considering representations of advanced linguistic features, people often directly resort to the set-theoretical models that are powerful and flexible (see, for example, [30], among others).

Another often-cited advantage of model-theoretic semantics is that a set-theoretical semantics connects language to the real world.[20] Some people think that such views are illusionary because set theory is just a theory of first-order logic and, compared with other semantic approaches, giving meaning representations set-theoretically does not make the semantic studies any closer to reality, if at all. However, the author does not take such an extreme view; instead, we argue that the representational approach does make the semantic studies *intuitively* more realistic, although not formally. For example, the notion of a collection of things is rather natural in everyday conversation and reasoning and set theory does provide powerful means to represent collections by means of sets. This has arguably contributed to the intuitive understanding of set-theoretical semantics and could be a reason why people think that such a model-theoretic semantics connects language to reality.

A common misunderstanding about formal semantics in modern type theories (MTT-semantics) is that it is only proof-theoretic, but not model-theoretic, since MTTs are proof-theoretically specified. This is mistaken and, in fact, MTT-semantics is model-theoretic and, in particular, it is representational. In MTT-semantics, types are employed to represent collections. This is very much like using sets in set theory to represent collections. Since MTTs have a rich type structure in that there are a lot of types with different and useful structures, these types can be effectively used for representation in semantic studies.

Representing Collections by Types in MTT-semantics The type structure of modern type theories[21] is much richer than the simple arrow types $A \to B$ as found in the Montagovian setting. Types can be employed to represent various collections such as those for CNs and this provides ad-

view that model-theoretic semantics as non-representational (in fact, just the opposite).

[20]There are many researchers expressing such views in slight different ways. See, for example, Portner's introductory book on model-theoretic semantics [66], among many others.

[21]See Footnote 1. In this paper, the type theory UTT [43], one of the MTTs, is used when concrete examples are given.

equate and useful tools for semanticists to construct formal semantics.[22] Here, we give several examples without going into too much details, because most of them have been explained in other places and the appropriate references are given in each case.

1. Σ-types, as introduced in §2.2.1, have been employed to interpret adjectival modifications when the adjectives are intersective, an idea originated from Ranta's work [72] and further elaborated in MTT-semantics [48, 19]. For instance, if $beautiful : Woman \to Prop$ is a predicate over type $Woman$, representing the collection of women, the Σ-type in (19) represents the collection that *beautiful woman* denotes.

   (19) $\Sigma x{:}Woman.\ beautiful(x)$

   It is worth remarking that this method of using Σ-types to represent adjective modifications is only adequate and viable because of two conditions. The first is that we have a proper notion of subtyping for MTTs, for otherwise such a Σ-type representation does not work.[23] Fortunately, the framework of coercive subtyping [44, 57] provides exactly this. In §5, we shall introduce signatures with coercive subtyping entries, a feature very useful for semantic studies.

   The second condition is subtler and less recognised – proof irrelevance should be enforced, for otherwise, the representation of adjective modification by Σ-types would be inadequate in that it would not have correct identity criteria. Taking the above interpretation (19) of *beautiful*

[22](Many thanks for a reviewer to make a remark that has led to this footnote.) A different way of giving semantics to CNs is the traditional Montagovian approach: to declare a type *Ent* and then interpret CNs as predicates of type $Ent \to Prop$. In such an approach, of course, one would not need the rich type structure in MTTs and, in fact, one can work in a simple type theory already. The idea of interpreting a CN as a type, rather than a predicate, was first considered by Ranta [72], where he used Σ-types to represent adjective modification. This CNs-as-types paradigm, different from the Montagovian CNs-as-predicates paradigm, has been argued to have several advantages including, for example, a better treatment of selectional restriction by typing, compatible use of subtyping in semantic interpretations and satisfactory treatment of some advanced linguistic features such as copredication (see, for example, [47, 52, 20] for further details). It may be interesting to remark that, recognising that interpreting CNs as types has several advantages, some researchers have suggested that *both* paradigms need be considered including, for instance, Retoré's idea in this respect [74]. A related issue in type-theoretical semantics is how to turn judgemental interpretations into corresponding propositional forms, as studied in [81] which proposes an axiomatic solution for such transformations that can be justified by means of heterogenous equality of type theory.

[23]See §3.3 of [72] for a very interesting discussion of the problem of 'multiple categorisation of verbs'. A satisfactory solution is an adequate subtyping mechanism for MTTs.

woman as an example. To use (19) for the representation adequately, it must be the case that two beautiful women are the same if, and only if, they are the same woman – how they are shown to be handsome should be irrelevant. This requires that some notion of proof irrelevance is enforced and, being recognised, this can be done appropriately – see [50, 53] for details.

2. Dependent product types and universes, as introduced in §2.2.2, can be used to formalise various linguistic features by means of Π-polymorphism. For example, in MTT-semantics, we employ the type universe $\textsc{cn}$ for the collection of (the interpretations of) CNs: each CN is interpreted as a type and therefore the type of all CNs is a universe whose objects are the types that interpret CNs. Let us use $\textsc{cn}$ as an example to illustrate how Π-polymorphism can be used in various useful representations: we illustrate this by showing that how to give semantics to verb-modifying adverbs [52, 19], subsective adjectives [16, 19] and quantifiers [41, 20], as exemplified as follows.

   (a) Verb modifying adverbs such as *loudly* and *simply*. In Montague's semantic setting, there is a single type e of all entities and, therefore, the type of *loudly* can be given as (20):

   (20) $loudly : (e \to t) \to (e \to t).$

   But when we have different types for different CNs, what do we do?[24] Here, Π-polumorphism over the universe $\textsc{cn}$ gives a useful solution: in MTT-semantics *loudly* can be given the following type (21):

   (21) $loudly : \Pi A : \textsc{cn}.\ (A \to Prop) \to (A \to Prop),$

   where for each common noun A, $loudly(A)$ has type $(A \to Prop) \to (A \to Prop)$, as expected. For instance, for $talk : Human \to Prop$, the phrase (22) can be interpreted as (23), which is of type $Human \to Prop$:

   (22) talk loudly

   (23) $loudly(Human, talk)$

[24]This has indeed been a seemingly difficult problem in MTT-semantics until (21) was proposed in [49].

(b) Subsective adjectives such as *small* and *large*. Such adjectives actually have different meanings when applied to different CNs: for example, *large* means differently in size when used in modifying *elephant* and *mouse*. This dependency on CNs is adequately captured by Π-polymorphism, as in the type (24) of *large* which, when applied to different CNs, denote different predicates as exemplified by (25) and (26).

(24) $large : \Pi A : \textsc{cn}.\ (A \to Prop)$

(25) $large(Elephant) : Elephant \to Prop$

(26) $large(Mouse) : Mouse \to Prop$

With the above interpretation of *large*, the sentence (27) is interpreted as (28), where c stands for the semantics of *the cat*:

(27) A large mouse was chased by the cat.

(28) $\exists x{:}LargeMouse.\ chase(c, m)$,
where $LargeMouse = \Sigma m{:}Mouse.\ large(Mouse, m)$.

(c) Generalised quantifiers such as *some* or *three*. Π-polymorphism is also useful in giving semantic typings for quantifiers. For example, *some* can be interpreted as having the type (29):

(29) $Some : \Pi A : \textsc{cn}.\ (A \to Prop) \to Prop$

Then, for instance, we can interpret the sentence (30) as (31), where $Student : \textsc{cn}$ and $speak : Human \to Prop$ (and $Student \leq Human$):

(30) Some students spoke.

(31) $Some(Student, speak)$

Furthermore, for *some* and *three*, we can give their definitions as well as typings. For instance, *some* can be defined as the logical existential quantifier ∃ restricted to the universe CN as in (32), which is of type (29).

(32) $Some = \lambda A{:}\textsc{cn}\ \lambda P{:}A \to Prop.\ \exists(A, P)$.

The sentence (33) is interpreted as (34) where *ast* is the predicate of type $Student \to Prop$, as defined in (35), that gives the semantics of *attack some student* and $attack : Human \to$

$Human \to Prop$ (and, since $Student \leq Human$, we have $attack : Student \to Student \to Prop$). Note that (34) is equal to (36), after expanding the definition (29) of *Some*.

(33) Some student attacked some student.

(34) $Some(Student, ast)$

(35) $ast = \lambda x{:}Student.\ Some(Student, attack(x))$

(36) $\exists x{:}Student \exists y{:}Student.\ attack(x, y)$

The quantifier *three* can also be defined (and, in more sophisticated situations, see [20] for detailed discussions.)

3. Disjoint union types, as briefly introduced in §2.2.2 and as pointed out there, have been proposed to interpret privative adjective modifications [48, 16, 19]. For instance, the phrases involving the privative adjective *fake* may be formalised as follows: if we use types G_R and G_F to represent the collections of (real) guns and that of fake guns, respectively, then the disjoint union type G in (37) can be used to interpret the collection of all guns:[25]

   (37) $G = G_R + G_F$.

   The type G consists of objects $inl(r)$ for real gun $r : G_R$ and $inr(f)$ for fake gun $f : G_F$. It may be worth pointing out that, in order to make the interpretations like (37) to be adequate, the associated injection operators are declared as coercions as in coercive subtyping [44, 57] (we omit the detailed technical justification here).

   (38) $G_R \leq_{inl} G$ and $G_F \leq_{inr} G$.

   For example, the predicates $real\_gun$ and $fake\_gun$ of type $G \to Prop$ can now be defined as follows: for any $r : G_R$ and $f : G_F$,

   (39) $real\_gun(inl(r)) = True$ and $real\_gun(inr(f)) = False$;

   (40) $fake\_gun(inl(r)) = False$ and $fake\_gun(inr(f)) = True$.

   Thus defined, the above predicates behave as intended: for example, it is easy to see that, for any $g : G$, $real\_gun(g)$ if, and only if,

[25] A reviewer has questioned this representation since it seems to be silent on whether we regard '*a fake fake gun*' as a real gun or fake gun, or even belonging to another type (then G as the type of all guns should be the disjoint union of three types, rather than just two). I'll leave this for future investigations.

Classification	***Example Adj(N)***	***Characterisation***	***MTT-semantics***
intersective	handsome man	N & Adj	$\Sigma x{:}Man.handsome(x)$
subsective	large mouse	N & Adj$_N$	$large : \Pi A{:}\textsc{cn}.(A \to Prop)$
privative	fake gun	¬ N	$G = G_R + G_F$
non-committal	alleged criminal	nothing implied	$\exists h{:}Human.\ H_{h,alleged}(...)$

Figure 3: MTT-semantics of adjective modifications.

$\neg fake\_gun(g)$. Note that, because injections inl and inr are coercions as in (38), we can simply write, for example, $real\_gun(f)$ for fake gun $f : G_F$ and this facilitates understanding and make things easier as well.

The rich typing structure in MTTs allows them to be powerful representational languages for formal semantics. Types in MTTs play powerful roles in representing various collections that are represented as sets or predicates in Montague's semantics. For instance, the interpretations of various adjective modifications, as classified traditionally, can be summarised as Fig 3, where the interpretations of the first three classes are explained above in 1, 2b, and 3, respectively, and that of the last uses the operators for non-committal adjectives A.[26]

Besides those discussed above, there are other examples, some of which involve some interesting linguistic features including, for example, the linguistic phenomenon of *copredication*, as studied by Pustejovsky [70], Asher

[26]Let's give an example to see how the operators $H_{h,A}$, for non-committal adjectives A, are defined. For instance, let us consider *alleged*. For any human agent $h : Human$, we can represent allegations of h by a predicate $\Sigma_{h,alleged} : Fin(n) \to Prop$, assuming that there are n allegations by h, where $Fin(n)$ is the finite type of n objects (see Appendix C). Then, the H-operator can be defined as (41), which informally says that 'h alleges that P is the case'.

(41) $H_{h,alleged}(P) = \exists i : Fin(n).\ \Sigma_{h,alleged}(i) =_{Prop} P.$

By this definition, that h alleges P means that there exists a proposition in h's allegation collection that is the same as P. Here, the sameness is represented by Leibniz equality, as defined in §2.2.2 and note that, technically, the above definition (41) is similar to that of belief operator in §6. Based on this, we can interpret the sentences involving *alleged*: for example, (42) can be interpreted as (43) and we shall not get into details here.

(42) John is an alleged criminal.

(43) $\exists h{:}Human.\ H_{h,alleged}(John\ is\ a\ criminal).$

[4] and others. It has been proposed that a new form of types in MTTs, called dot-types of the form $A \bullet B$, can be adequately used to deal with copredication.[27]

To summarise, we have discussed in this subsection how to use several type constructors introduced in §2.2 in constructing formal semantics, showing that types in MTTs can be effectively used to represent various collections. Put in another way, they offer flexible choices and powerful means in various representations in semantic studies. Of course, types are not sets – they are more manageable (for example, type-checking is decidable while the set-theoretical membership is not). And, as discussed in Introduction, that is also why MTTs can have proof-theoretic semantics and we can use MTT-semantics to deal with NL reasoning in proof assistants.

4.2 Proof-Theoretic Characteristics: Meaning Theories and Computer-Assisted Inference

MTT-semantics has proof-theoretic semantics, in the sense of [39], not just because that they are specified by inference rules, but more importantly, in their specifications the introduction and elimination rules are *in harmony*[28] and has nice meta-theoretic properties such as strong normalisation[29], among others, and hence they have proof-theoretic meaning theories, on the one hand, support computer-assisted reasoning based on proof assistants, on the other.

Proof-Theoretic Semantics. Besides model-theoretic semantics, there is another kind of formal semantics: *proof-theoretic semantics.* Proof-theoretic semantics was pioneered by Gentzen [35], developed by Prawitz [68, 69] and Martin-Löf [59, 60] (among other logicians) to study meaning theories of logical systems, and studied by Dummett [31] and Brandom [11, 12] (among other philosophers) to study general philosophical theories of meaning of NLs. Proof-theoretic semantics is a form of inferential semantics in that it not only takes inference seriously but regards it as the central concept of

[27]This extends the type structure of MTTs in a proper way: dot-types, first proposed in [47] and then further studied in [52, 20], are not ordinary inductive types. We do not have space to have in-depth discussions about dot-types here.

[28]See Dummett [31] for a proposal and interesting discussions of this concept and it has since been further investigated by others in the context of proof-theoretic semantics.

[29]Technically, the strong normalisation property says that every computation starting from a well-typed term terminates. This is a fundamental property that usually guarantees that the proof system well-behaves: it usually implies logical consistency of the type theory concerned.

meaning theories and such a direct link to inference is regarded as a key advantage of proof-theoretic semantics.[30]

The MTT-semantics does not only have model-theoretic characteristics, as studied above in §4.1 and further developed in §5, but proof-theoretic characteristics. This is because that the meanings of judgements, which are the basic sentences in MTTs, can be understood in a proof-theoretic meaning theory. Martin-Löf has carried out a whole programme of proof-theoretic semantics, studying and developing it for his type theory [59, 60]. Therefore, once MTT-semantics has been constructed, the statements in the semantics (i.e., judgement in the MTT) can be understood proof-theoretically.

Studying proof-theoretic meaning theories, people have considered two aspects of use: verification and consequential application. For MTTs, the first aspect is to consider how a judgement can be correctly asserted while the other (second) aspect is to consider what consequences it has to accept that a judgement is correct. The verificationist thinks that the first aspect of verification is the (possibly only) central conceptual focus in meaning theories while the pragmatist thinks that the second aspect of consequential application is instead central. For example, under the verificationist view on meaning explanations for type theory, one may consider such a meaning explanation of the judgement $a : A$ (I omit the context Γ here): it can correctly asserted if a computes to a canonical object of type A.

The formal property of strong normalisation (see Footnoot 29) is closely related to canonicity and, in many cases, normalisation implies canonicity and even harmony. Although to explain the proof-theoretic properties like normalisation is beyond the scope of the current paper, these properties do play a fundamental role in justifying computer implementations of MTTs in the so-called proof assistants – a topic we discuss below.

NL Reasoning in Proof Assistants. In computer science, MTTs have been implemented in computer-assisted reasoning systems, called proof assistants, such as Coq [24] (France), Agda [3] (Sweden) and Lego/Plastic [56, 14] (U.K.), and used in applications such as the formalisation of mathematics and verification of programs. Based on this, MTT-semantics has been directly implemented, in Coq and Plastic, and used for NL reasoning [18, 17, 80, 49].

[30]There are philosophical arguments that conceptual role semantics (eg, proof-theoretic semantics) supports the view in cognitive science that something is meaningful just because that it plays a certain role (eg, inferential role) in a person's psychology. I ignore this because here proof-theoretic semantics is considered as an approach to formal semantics.

The implementation of an MTT in a proof-assistant replies on the proof-theoretic properties: it depends on the fact that the implemented system well-behaves. For example, type-checking is a crucial operation in such a system. In a dependent type theory, type-checking requires (and, hence, is equivalent to) type inference and, to do this, a typical algorithm is to compute the relevant terms to their normal forms (or, technically, head-normal forms) and then perform the checking (see, for example, Chaoter 5 of [43]). This requires that every well-typed term is strongly normalising, i.e., that the strong normalisation property holds. As another example, implementing some crucial tactics concerning Σ-types requires that the property called strengthening (or contraction) hold, for otherwise the tactic concerned cannot be implemented in certain ways.

Put in another way, in order to justify that the implementation of a proof assistant based on an MTT is correct, we need to know that the type system has some nice properties which are crucial in the understanding of the system, on the one hand, and in its correct implementation, on the other. In other words, as the normalisation property is closely related to having a proof-theoretic semantics (or, we can say that the former is a formal reflection of the latter), the MTT concerned having proof-theoretic semantics is a fundamental basis for it to be a well-behaving foundational language for formal semantics for practical reasoning of computer-assisted proof assistants as well as for better understanding of the semantic studies.

5 Signatures with Subtyping and Manifest Entries

As mentioned above, there are two notable features of MTTs that make MTT-semantics model-theoretic: using types to represent collections, as described in §4.1 above, is one of them. The other is the mechanism of contextual declarations: we shall formally extend type theories with *signatures* and use them in representing (incomplete) worlds in formal semantics.

Contextual mechanisms do not exist in traditional logical systems such as simple type theory. Contexts (and signatures as to be introduced below in §5.2) are new to logical systems and provide us with useful *internal* mechanisms to describe incomplete worlds. This has been recognised by Ranta [72], among others, and we think that it is one of the important aspects that MTTs provide useful model-theoretic means to describe formal semantics.

Compared with the existing notion of context in type theory, our notion of signature provides, on the one hand, a more restricted form of contextual mechanism since they do not allow abstraction over their entries (see below for explanations why this is preferable) and, on the other hand, more powerful and flexible tools since they allow new forms of entries to describe subtyping and manifest definitions. In §5.1, we formally introduce signatures, but with only membership entries first, and show with an example how signatures can be used to represent incomplete worlds. Then, in §5.2, we extend signatures with two forms of new entries: the subtyping entries and manifest entries, which can be used to describe more sophisticated situations in the world, including those involving infinitely many entries or linguistic coercions.

5.1 Representing Incomplete Worlds with Signatures

Signatures: Motivations. A context Γ is a part of a judgement $\Gamma \vdash a : A$, the basic form of sentences in existing type theory. As mentioned above, it has been proposed that contexts can be used to represent incomplete worlds in semantic studies. However, using contexts to represent incomplete worlds has problems. There are two issues. The first is that a world should be represented by assuming *constants*, rather than by assuming variables in contexts. For instance, to declare that John is a man, one might use an entry in a context to do so (e.g., the last entry of the context (44)). However, this is not quite adequate because one can always abstract over the contextual entry $j : Man$ by means of abstraction operators such as the universal quantifier: for instance, assuming that P is a predicate such as *talk*, if under (44) we can prove $P(j)$ (i.e., the judgement (45) is derivable), we could then derive (46), which says P holds for every man – this is obviously an incorrect conclusion that should not have been derived.

(44) $\Gamma,\ j : Man$

(45) $\Gamma,\ j : Man \vdash P(j)\ true$

(46) $\Gamma \vdash \forall j : Man.P(j)\ true$

Such an over-generation happens because of the incorrect use of a context entry to describe a constant (j to represent *John* in the above example) that can be abstracted over by abstraction operators. This is why we introduce signatures whose entries introduce constants which cannot be abstracted over.

Secondly, to use only membership entries of the form $c : K$ to represent more sophisticated situations is not easy, if possible at all. For instance, one often want to assume that a collection is contained in another and, when collections are represented as types, one would need to make subtyping assumptions such as $A \leq B$, but to do so, we'd need to allow a new form of entry in our signature: *subtyping entries*. As another example, one may want to represent a world consisting of infinitely many entities by means of a signature, which like a context always consists of finitely many entries. So-called *manifest entries* in a signature allow one to declare entities containing infinite amount of information and, thanks to that, declaring infinitely many entities becomes easier. In the next subsection §5.2, we shall introduce these two new forms of entries in signatures, which allow flexible ways of representing sophisticated (incomplete) worlds.[31]

Signatures with Only Membership Entries. A signature is similar to but different from a context: it is similar to a context in that it is also a finite sequence of entries like those in (3) except that an entry in a signature can either be a membership entry as those in (3) or one of the two new forms – a subtyping entry $A \leq_\kappa B$ or a manifest entry $c \sim a : A$, to be introduced in the next subsection §5.2. We shall first introduce signatures with only membership entries.[32]

Introducing signatures into type theory, we extend its judgements with signatures: for example, the membership judgement of the form (2) now becomes the following (47), where Σ is a signature:

(47) $\Gamma \vdash_\Sigma a : A$

Like entries in context Γ, the assumptions given in the entries of a signature are all taking into effect in reasoning. However, there is a crucial difference: the membership entry $c : K$ in a signature declares that c is a *constant* of kind K. This is different from a contextual entry $x : K$ that declares x to be a *variable*. Note that a variable can be abstracted by, for example, quantification or λ-abstraction as exemplified by a rule like the one below:

$$\frac{\Gamma,\ x : K \vdash_\Sigma P : Prop}{\Gamma \vdash_\Sigma \forall x{:}K.P : Prop}$$

However, constants in signatures can never be abstracted in this way – that is why they are called constants. Therefore, compared with contexts in

[31] Another reason to introduce signatures is because that extending contexts to contain these two new forms of entries is not easy and, actually, an earlier attempt [55] to do so has met with some difficulties; in particular, its meta-theoretic study is rather sophisticated and has been a difficult open question. Instead, it is easier to add them as entries in signatures – this is what we propose to do.

[32] The notion of signatures (with only membership entries) was first introduced into type theory in Edinburgh Logical Framework [37], where signatures are used to describe object logical systems.

type theory, signatures are more suitable to be used to represent incomplete worlds.[33] Because of this, we shall often use c's in signatures, for example, writing a signature with only membership entries as

(48) $c_1 : K_1,\ c_2 : K_2,\ ...,\ c_n : K_n$

We shall now formally describe the extension of MTTs by means of signatures with the traditional membership entries and then give an example to show, when an MTT is used as a representational language, how various situations of the world can be represented by means of signatures.[34]

Signatures: a Formal Presentation. Type theories can be specified in a logical framework such as Martin-Löf's logical framework [62] or its typed version LF [43]. We extend LF with signatures to obtain LF$_\Sigma$, whose formal presentation (its judgement forms and its inference rules) can be found in Appendix B.1.

For people who are familiar with LF, LF$_\Sigma$ augments each of the judgement forms in LF with signatures (for example, in the form of (47)) and all of the inference rules of LF (i.e., those in Figures 9.1 and 9.2 of Chapter 9 of [43]) become inference rules of LF$_\Sigma$ after replacing $\vdash$ by $\vdash_\Sigma$ (and changing the judgement form 'Γ *valid*' to '$\vdash_\Sigma \Gamma$'). For instance, the following rule for λ-abstraction[35] in LF

$$\frac{\Gamma, x : K \vdash b : K'}{\Gamma \vdash [x{:}K]b : (x{:}K)K'}$$

becomes, in LF$_\Sigma$,

$$\frac{\Gamma, x : K \vdash_\Sigma b : K'}{\Gamma \vdash_\Sigma [x{:}K]b : (x{:}K)K'}$$

In addition, in LF$_\Sigma$, we have rules for signatures and contexts, how they are formed and how the assumptions in them can be used to contribute to inferences – see Appendix B.1 for formal details.

In LF$_\Sigma$, various type constructors, as introduced in §2.2, can be specified by declaring constants (as done in LF [43], with the only difference

[33]Also, because the constants in signatures cannot be abstracted, it is easier meta-theoretically to add new forms of entries to signatures than to contexts (see §5.2.3).

[34]Contexts with such traditional entries have been used by Ranta [72] and others [10, 29] to represent situations, where they do not consider the issue of difference between variables and constants. We consider signatures rather than contexts here. Note that signatures may contain other forms of entries which are studied in the next section §5.2.

[35]In LF, we use the notation $[x{:}K]b$ for $\lambda x{:}K.b$ and $(x{:}K)K'$ for $\Pi x{:}K.K'$.

that we now have signatures). For instance, Σ-types can be specified as in Appendix B.2. Technically, compared with the direct rules as given in §2.2.1 and Appendix A, the LF_Σ specifications (and LF specifications) of these type constructors are more concise, but more laborious in notations. We omit further explanations here.

Representing Incomplete Worlds by Signatures: a Simple Example. That signatures can be used to represent situations in the world to be described is the other facet that the MTT-semantics is model-theoretic. Here, we use an example given in [51] to illustrate how signatures can be used to represent situations. From the example below, it is easy to see that it would not be adequate to represent those entries in Σ as entries in a context, since they are constants, not variables that can be abstracted over. Our example is simple in that, for instance, the domain in the example is finite. This will serve as a basis for our discussions on how more sophisticated situations such as those involving infinity or other special circumstances should be represented in §5.2.

Example 5.1 *The example, taken from Chapter 10 of Saeed's book [76], is about an (imagined) scenario in the Cavern Club at Liverpool in 1962 where the Beatles were rehearsing for a performance. It can be represented as follows.*

1. *The domain consists of several peoples including the Beatles (John, Paul, George and Ringo), their manager (Brian) and a fan (Bob). This can be represented be means of the following signature Σ_1:*

$$\begin{aligned}\Sigma_1 \;\equiv\;& D : \mathit{Type},\\ & \mathit{John} : D,\ \mathit{Paul} : D,\ \mathit{George} : D,\ \mathit{Ringo} : D,\ \mathit{Brian} : D,\ \mathit{Bob} : D\end{aligned}$$

2. *The assignment function assigns, for example, predicate symbols such as B and G to the propositional functions expressing 'was a Beatle' and 'played guitar', respectively. We can introduce the following in our signature to represent such an assignment function:*

$$\begin{aligned}\Sigma_2 \;\equiv\;& B : D \to \mathit{Prop},\ b_J : B(\mathit{John}),\ ...,\ b_B : \neg B(\mathit{Brian}),\ b'_B : \neg B(\mathit{Bob}),\\ & G : D \to \mathit{Prop},\ g_J : G(\mathit{John}),\ ...,\ g_G : \neg G(\mathit{Ringo}),\ ...\end{aligned}$$

The signature that represents the overall situation will be of the form $\Sigma \equiv \Sigma_1,\ \Sigma_2,\ ....$ We shall then have, for instance,

$$\vdash_\Sigma G(\mathit{John})\ \ \mathit{true}\quad \text{and}\quad \vdash_\Sigma \neg B(\mathit{Bob})\ \ \mathit{true}.$$

where $G(\mathit{John})$ and $B(\mathit{Bob})$ are semantically interpret John played Guitar and Bob was a Beatle, respectively. □

5.2 Subtyping and Manifest Entries in Signatures

In the last subsection, we have extended MTTs with signatures consisting of traditional membership entries. In this subsection, we consider two new forms of signature entries – subtyping entries and manifest entries, which will make the notion of signature be more powerful in describing advanced and sophisticated phenomena of an incomplete world. We shall introduce the new forms of entries, illustrate their uses in semantic representations and discuss the meta-theoretic properties of the extended type theories.

In earlier work, these forms of entries were proposed and studied as contextual entries: contextual manifest entries in type theory were first studied in [46] and contextual subtyping entries (in the name of coercion contexts) in [52]. Here, we consider them as entries in signatures as proposed in [51]. As mentioned earlier in the above subsection, having subtyping entries in contexts make meta-theoretic studies much harder (a preliminary study was reported in [55]). As signature entries, we can prove that the extension well behaves in the sense that the properties of the original type theory are all preserved (see §5.2.3). In other words, signatures, as supposed to contexts, are not only more useful and more adequate in semantic representations, but also simpler meta-theoretically.

5.2.1 Subtyping Entries

Coercive subtyping has been studied for subtyping and abbreviations in MTTs and the associated proof assistants [44, 57].[36] Introducing subtyping entries (to either signatures or contexts) is to localise the coercive subtyping mechanism, which has been studied globally in earlier research.[37] For exam-

[36]The word ‘coercion’ has been used for related but maybe different things including coercions in programming languages and coercions in linguistics. See Asher and Luo [5] for a use of coercive subtyping in modelling linguistic coercions and Retoré et al [7] for another proposal of using coercions to deal with some linguistic coercions in lexical semantics.

[37]In previous research on coercive subtyping [44, 57], one has considered the subtyping extensions $T[C]$, where T is a type theory and C is a set of coercive subtyping judgements. So, coercions were assumed at the system level – done *globally*. This is in contrast to the local treatments of subtyping in signature, as we propose here.

ple, if Man and $Human$ are base types[38], one may have a subtyping entry (51) to assume that every man is a human. Note that such an entry will occur in a signature Σ (say, (52)), and when we have a derived judgement under Σ as in (53), it may not be derivable under a different signature Δ, under which one cannot have (51), as shown in (54).

(51) $Man \leq Human$

(52) $\Sigma = (\Sigma_1, Man \leq Human, \Sigma_2)$

(53) $\vdash_\Sigma Man \leq Human$

(54) $\not\vdash_\Delta Man \leq Human$

Syntactically, the system LF_Σ, as introduced in the last subsection, is extended with the judgement forms $\Gamma \vdash_\Sigma A \leq_\kappa B : \mathit{Type}$ (we shall often just write $A \leq_\kappa B$ even when A and B are types) and $\Gamma \vdash_\Sigma K \leq_\kappa K'$. A subtyping entry to signatures can be introduced by means of the following rule:

$$\frac{\vdash_\Sigma A : \mathit{Type} \quad \vdash_\Sigma B : \mathit{Type} \quad \vdash_\Sigma \kappa : (A)B}{\Sigma,\ A \leq_\kappa B \ \ \mathit{valid}}$$

where $(A)B$ is the kind of functional operations from A to B. The assumed subtyping relationships in a signature are derivable, as reflected in the following rule:

$$\frac{\vdash_{\Sigma, A \leq_\kappa B, \Sigma'} \Gamma}{\Gamma \vdash_{\Sigma, A \leq_\kappa B, \Sigma'} A \leq_\kappa B}$$

Then the rules for coercive subtyping [57], albeit extended for judgements with signatures, are all applicable. For instance, if signature Σ contains $A \leq_\kappa B$ and $B \leq_{\kappa'} C$, we can derive $A \leq_{\kappa' \circ \kappa} C$ under Σ, where $\kappa' \circ \kappa$ is the functional composition of κ and κ'.

It is worth pointing out that *validity* of a signature is not enough anymore when we consider subtyping entries in signatures. For signature Σ to be legal, we need the subtyping assumptions in Σ to be *coherent* in the sense

[38]This is just for the sake of giving examples. In usual cases, Man (or even $Human$) may not be a base type. For instance, we may have Man defined as (49), which is a subtype of $Human$ through the first projection as coercion, as shown in (51).

(49) $Man = \Sigma x{:}Human.\ male(x)$

(50) $Man \leq_{\pi_1} Human$

that, informally, all coercions between any two types are equal, i.e., in some appropriate subsystem,[39] if $\Gamma \vdash_\Sigma A \leq_\kappa B$ and $\Gamma \vdash_\Sigma A \leq_{\kappa'} B$, then $\Gamma \vdash_\Sigma \kappa = \kappa' : (A)B$.

Introducing subtyping entries makes using type theory for formal semantics much more convenient. First of all, it is now possible for one to *localise* subtyping assumptions. In the above, we have given a simple example to assume that Man is a subtype of $Human$. As another example, we may consider some special subtyping relations that only make sense and are reasonable in specific phenomena, but not in general. In such cases, the localised subtyping assumptions in signatures become especially useful. For instance, only in a cafe or restaurant would it be reasonable to say (56). To represent such a phenomenon in a cafe, one might reasonably assume the following subtyping entry (57) in the overall signature, which will then allow the sentence (56) to be semantically interpreted as intended.

(56) The ham-sandwich left without paying the bill.

(57) $Ham\text{-}sandwich \leq Human$

We'd like to note that the reference transfers such as above are studied by Nunberg [63] (and also see [5] for other such examples, among others).

It is worth noting that localisation of subtyping assumptions is very useful here. In a signature with (57) as an entry, one may then interpret (56) as intended. In other (most and usual) signatures, one would not be able to deduce (57) and that is intended as well. If one only had global coercive subtyping, it would be rather difficult to consider such a phenomenon, if possible at all.

[39]It is important that the condition is not stated for the whole system of coercive subtyping, for otherwise it would become trivial. To see this is the case, we note that the whole system has the following *coercive definition rule*:

$$(CD) \qquad \frac{\Gamma \vdash_\Sigma f : (x{:}K)K' \quad \Gamma \vdash_\Sigma k_0 : K_0 \quad \Gamma \vdash_\Sigma K_0 \leq_\kappa K}{\Gamma \vdash_\Sigma f(k_0) = f(\kappa(k_0)) : [\kappa(k_0)/x]K'}$$

If Σ is incoherent, i.e., $K \leq_{c_1} K'$, $K \leq_{c_2} K'$ and $c_1 \neq c_2$, then, by the above rule (and the $\beta\eta\xi$-rules), we'd have that $c_1 = c_2$, as shown by (55) – a contradiction.

(55) $c_1 = \lambda x{:}K.c_1(x) = \lambda x{:}K.Id_{K'}(c_1(x)) = \lambda x{:}K.Id_{K'}(x) = \lambda x{:}K.Id_{K'}(c_2(x)) = \lambda x{:}K.c_2(x) = c_2$

Therefore, it is important to state the coherence condition in the subsystem which does not contain the above (CD) rule. An interested reader may look at [57] for more details.

5.2.2 Manifest Entries

A *manifest entry* in a signature is of the form (58) which assumes that c be a constant that behaves exactly like a of type A. Alternatively, one can think that in any place that we could use an object of type A, we could use c which actually plays the role of a.

(58) $c \sim a : A$

Signatures in LF_Σ can be extended with manifest entries:

$$(*) \qquad \frac{\vdash_\Sigma A : Type \quad \vdash_\Sigma a : A \quad c \notin dom(\Sigma)}{\Sigma,\ c \sim a : A\ valid}$$

where *Type* is the kind of all types (in the object type theory). In fact, such manifest entries can be introduced by means of special membership entries with the help of the coercive subtyping mechanism. We now proceed with its formal description.

As the following example shows, manifest entries allow one to express things in a more concise and simpler way and can hence considerably reduce the complexity of semantic representations.

Example 5.2 *With manifest entries, the situation in Example 5.1 can be represented as the following signature:*

$$D \sim a_D : Type,\ B \sim a_B : D \to Prop,\ G \sim a_G : D \to Prop,\ \ldots\ \ldots \qquad (59)$$

where

- $a_D = \{John,\ Paul,\ George,\ Ringo,\ Brian,\ Bob\}$ *is a finite type;*[40]
- $a_B : D \to Prop$, *the predicate 'was a Beatle', is an inductively defined function such that* $a_B(John) = a_B(Paul) = a_B(George) = a_B(Ringo) = True$ *and* $a_B(Brian) = a_B(Bob) = False$*; and*
- $a_G : D \to Prop$, *the predicate 'played guitar', is an inductively defined function such that* $a_G(John) = a_G(Paul) = a_G(George) = True$ *and* $a_G(Ringo) = a_G(Brian) = a_G(Bob) = False$*.*

In other words, Σ_1 *in Example 5.1 is now expressed by the first entry of (59) and* Σ_2 *in Example 5.1 by the second and third entries of (59).*

[40] a_D is a finite type (not a finite set). Formally, a_D is the finite type $Fin(6)$, as defined in Appendix C, with $John = zero(5)$, $Paul = succ(5, zero(4))$, etc.

Manifest entries in signatures can also be used to represent infinitely many entities in finitely many entries. Note again that the length of a signature is always finite and this ability of representing infinite entity makes it rather handy in many cases. For instance, with traditional membership entries, we can only describe finite domains as we have done in Example 5.1. What if the domain D is infinite? This can be done by using a manifest entry – as in Example 5.2, we can assume that

$$D \sim Inf : Type,$$

where Inf is some inductively defined type with infinitely many objects. Similarly, one can assume an infinite predicate over the domain, represented as:

$$P \sim \text{P-defn} : D \to Prop,$$

where P-defn is also inductively defined.

Manifest entries can be regarded as abbreviations of special membership entries [46] with the help of the coercive subtyping mechanism [44, 57]. Formally, to add the above manifest entry (58) to a signature is to add the following two entries (60), where $\mathbf{1}_A(a)$ is the inductive unit type parameterised by $A : Type$ and $a : A$, whose only object is $*_A(a)$,[41] and $\xi_{A,a}(x) = a$ for every $x : \mathbf{1}_A(a)$.

(60) $c : \mathbf{1}_A(a),\ \mathbf{1}_A(a) \leq_{\xi_{A,a}} A$

Now, if an expression has a hole that requires a term of type A, we can use c to fill that hole; then the whole expression is equal to that with the hole filled by a. For example, if the expression is $f(\_)$, then $f(c)$ is equal to $f(a)$ (this is given by the coercive definition rule in coercive subtyping – see Footnote 39.)

Note that the subtyping entries involving ξ form coherent signatures; in particular, if for two manifest entries $c \sim a : A$ and $d \sim b : B$ we have $\mathbf{1}_A(a) = \mathbf{1}_B(b)$ and $A = B$, then $\xi_{A,a} = \xi_{B,b}$, as coherence requires. Put in another way, if the subtyping entries in a signature are coherent, the signature is coherent since its manifest entries do not cause incoherence. Therefore, the extension with manifest entries in signatures preserves the nice properties of the original type theory such as strong normalisation and logical consistency.

[41]See Appendix A.4 for the formal specification of the unit types.

5.2.3 Meta-theoretic Results

In this subsection, we briefly describe the meta-theoretic results of the extension of type theories with signatures. The following theorem states that the extension with signatures well-behaves as expected.

Theorem 5.3 *Let T be a type theory specified in LF and T_S the extension of T with signatures (with membership, subtyping and manifest entries in signatures). Then, T_S preserves the meta-theoretic properties of T (such as logical consistency, Church-Rosser and strong normalisation) for all coherent signatures.* □

For example, let SN stand for the property of strong normalisation (see Footnote 29.) A corollary of the above theorem is: signature coherence implies that SN is preserved, i.e., if T is SN (if $\Gamma \vdash a : A$, then a is strongly normalising), so is T_S for coherent signatures (if Σ is coherent and $\Gamma \vdash_\Sigma a : A$, then a is strongly normalising). It is worth remarking that, since manifest entries can be faithfully represented by means of membership and subtyping entries as explained above, to prove the above theorem, one only needs to study the extension with subtyping entries.

A detailed study of the metatheory is beyond the scope of the current paper. The interested readers are referred to, for example, Lungu's PhD thesis [40, 42] for details, where the conservativity result in [57] has been extended to the calculi with signatures.

6 Beliefs in MTT-semantics

MTTs provide powerful and flexible tools for various issues in formal semantics, including those which were only dealt with satisfactorily in a model-theoretic way. One of such issues involves the so-called hyperintensional problems, which have been studied in various formal frameworks including, for example, Cresswell [26], Thomason [78], Pollard [65], Fox and Lappin [32], among others.

One of the hyperintensional problems is that of logical omniscience: for instance, if an agent believes in (or knows) something, then s/he believes (or knows) everything that is logically equivalent – this is obviously not the case as the following example shows, where John is a young boy who is five years old:

(61) John believes that John is John.

(62) John believes that there are infinitely many prime numbers.

In this section, we propose a (new) formal treatment of beliefs in type theory.

In Martin-Löf's type theory, Ranta [72] has proposed a notion of belief context to model beliefs as sequences of membership entries each of which represents a belief of an agent. For an agent h, the belief context of h is a finite sequence $x_1 : A_1, ..., x_n : A_n$, where A_i's stand for h's beliefs. A belief operator R_h is then introduced: $R_h(A)$ informally says that h believes A. Formally, for any proposition A, $R_h(A)$ is true just in case that A is true in h's belief context; in other words, if h's belief context is $x_1 : A_1, ..., x_n : A_n$, then $R_h(A)$ is true if, and only if, $x_1 : A_1, ..., x_n : A_n \vdash A\ true$, and equivalently, if and only if the Π-type $\Pi x_1{:}A_1 ... \Pi x_n{:}A_n.\ A$ is inhabited. Unfortunately, although useful in many applications, this notion of belief context does not solve the problem of hyperintensional problems such as logical omniscience since whether an agent believes in something is closed under derivability and hence, for example, if h believes in A (i.e., $R_h(A)$ is true), then h would believe in every proposition that is logically equivalent to A.

Our proposal offers a more restrictive and intensional notion to reflect the idea that, given a collection of beliefs of an agent h, a statement is believed by h if, and only if, it is intensionally equal to one of the existing beliefs in the collection of h's beliefs. Assume that an agent h have n beliefs. Then, as in (63), the collection of beliefs of h, notation Σ_h, can be represented as a predicate over type $Fin(n)$, i.e., a function of type $Fin(n) \to Prop$, where $n : Nat$ is a natural number and $Fin(n)$ is the finite type with n objects:[42]

(63) $\Sigma_h : Fin(n) \to Prop$

Now, 'h believes that ...' can be formally represented by the belief operator $B_h : Prop \to Prop$ defined in (64) for any proposition $P : Prop$, where $=_{Prop}$ is the Leibniz equality between propositions, as defined in §2.2.2 when Π-types and their uses are discussed.[43]

(64) $B_h(P) = \exists i : Fin(n).\ \Sigma_h(i) =_{Prop} P.$

Informally, $B_h(P)$ says that there exists a belief in h's belief collection that is the same as P.

[42]See Appendix C for a more formal description of $Fin(n)$. For notational easiness, in this section, we abuse the language to use $0, 1, ..., n-1$ to stand for the objects of $Fin(n)$.

[43]Here we work in the impredicative type theory UTT, an impredicative type theory. In a predicative type theory such as Martin-Löf's type theory [62], one may use the identity type Id and a universe U: we shall then have $\Sigma_h : Fin(n) \to U$ and $B_p : U \to U$ and, for any $A : U$, $B_h(A) = \exists i{:}Fin(n).\ Id_U(\Sigma_h(i), A)$.

For instance, let $j : Human$ be the semantics of John, who believes n propositions P_1, ..., P_n where, for instance P_1 is the proposition in (64) and none of P_i's ($i = 1, ..., n$) is Leibniz equal to the proposition P as expressed informally as (66). But since $P \neq_{Prop} P_1$, and actually not Leibniz equal to any of P_i's, we shall have (67), i.e., John does not believe that (66) is true.

(65) 2 is the only prime number.

(66) {2} is the set of all prime numbers.

(67) $\neg B_j(66)$

As another example, using the belief operator defined in (64), the sentence (68) can be interpreted as (69), where j and m interpret *John* and *Mary*, respectively.

(68) John does not believe that Mary believes that ...

(69) $\neg B_j(B_m(...))$

Note that the Leibniz equality is intensional (see §2.2.3). In particular, propositional extensionality does not always hold: logical equivalence does not imply Leibniz equality. For a typical hyperintensional context such as 'h believes that ...' where substitution with logically equivalent formulas does not preserve meanings, our proposed belief operator B_h provides an intensional treatment in type theory: for propositions P and Q, $P \Leftrightarrow Q$ does not imply that $P =_{Prop} Q$. Put in another way, if $B_h(P)\ true$ and $P \Leftrightarrow Q\ true$, it is not necessarily the case that $B_h(Q)\ true$. So, the belief operator B_h does not suffer from the problem of logical omniscience.[44] This is in contrast with the traditional possible world semantics or Ranta's notion of belief context both of which, as mentioned above, would suffer in such a hyperintensional context.

The above treatment of beliefs can be genralised to deal with other hyperintensional problems such as that about knowledge. Similar formal mechanisms may also used to give semantic interpretations of non-committal adjectives (see Footnote 26), but we omit the details here.

7 Conclusion

In this paper, we have argued that MTT-semantics can be seen as both model-theoretic and proof-theoretic, focussing on the arguments for the for-

[44]Of course, by definition (64), the belief operator is closed under the Leibniz equality between propositions.

mer, partly because it is less discussed and not so easy to appreciate. In particular, two facets of MTT-semantics' model-theoretic characteristics have been illustrated: the first one with types representing collections and the other with signatures representing situations. It is worth emphasising again that it may be new to many researchers that a proof-theoretic language can be employed to give model-theoretic semantics to natural language. MTTs are such foundational languages.

There are several interesting directions for future work. For example, it is important to develop further the meaning theories of modern type theories in general. For instance, it is interesting to study and obtain a better meaning explanation of an impredicative universe like that found in UTT. Even just for predicative type theories like that of Martin-Löf, people may argue that it is still unclear how hypothetical judgements should be explained satisfactorily. Further studies are called for.

Acknowledgement. The author is very grateful to many researchers for discussions on related topics including Nicholas Asher, Stergios Chatzikyriakidis, Nissim Francez, Justyna Grudzińska, Georgiana Lungu, James McKinna, Koji Mineshima, Glyn Morrill, Larry Moss, Aarne Ranta, Christian Retoré, Sergei Soloviev, among others.

References

[1] Peter Aczel and Nicola Gambino. Collection principles in dependent type theory. In Paul Callaghan, Zhaohui Luo, James McKinna, and Robert Pollack, editors, *Types for Proofs and Programs: International Workshop, TYPES 2000, Durham, UK, December 8–12, 2000. Selected Papers*, volume 2277 of *LNCS*, pages 1–23. Springer-Verlag, 2002.

[2] R. Adams and Z. Luo. Weyls predicative classical mathematics as a logic-enriched type theory. *ACM Transactions on Computational Logic*, 11(2), 2010.

[3] The Agda proof assistant (version 2). Available from the web page: `http://appserv.cs.chalmers.se/users/ulfn/wiki/agda.php`, 2008.

[4] N. Asher. *Lexical Meaning in Context: a Web of Words.* Cambridge University Press, 2012.

[5] N. Asher and Z. Luo. Formalisation of coercions in lexical semantics. *Sinn und Bedeutung 17, Paris*, 2012.

[6] J. Barwise and J. Perry. *Situations and Attitudes.* MIT Press, 1983.

[7] C. Bassac, B. Mery, and C. Retoré. Towards a type-theoretical account of lexical semantics. *Journal of Logic, Language and Information*, 19(2), 2010.

[8] D. Bekki. Representing anaphora with dependent types. *LACL 2014, LNCS 8535*, 2014.

[9] P. Boldini. Formalizing context in intuitionistic type theory. *Fundamenta Informaticae*, 42(2):1–23, 2000.

[10] P. Boldini. Formalizing contexts in intuitionistic type theory. *Fundamenta Informaticae*, 4(2), 2000.

[11] R. Brandom. *Making It Explicit: Reasoning, Representing, and Discursive Commitment.* Harvard University Press, 1994.

[12] R. Brandom. *Articulating Reasons: an Introduction to Inferentialism.* Harvard University Press, 2000.

[13] R. Burstall and J. Goguen. Introducing institutions. *LNCS 164*, 1984.

[14] P. Callaghan and Z. Luo. An implementation of LF with coercive subtyping and universes. *Journal of Automated Reasoning*, 27(1):3–27, 2001.

[15] S. Chatzikyriakidis and Z. Luo. An account of natural language coordination in type theory with coercive subtyping. In Y. Parmentier and D. Duchier, editors, *Proc. of Constraint Solving and Language Processing (CSLP12). LNCS 8114*, pages 31–51, Orleans, 2012.

[16] S. Chatzikyriakidis and Z. Luo. Adjectives in a modern type-theoretical setting. In G. Morrill and J.M Nederhof, editors, *Proceedings of Formal Grammar 2013. LNCS 8036*, pages 159–174, 2013.

[17] S. Chatzikyriakidis and Z. Luo. Natural language reasoning in Coq. *J. of Logic, Language and Information*, 23, 2014.

[18] S. Chatzikyriakidis and Z. Luo. Proof assistants for natural language semantics. lacl 2016. nancy. 2016.

[19] S. Chatzikyriakidis and Z. Luo. Adjectival and adverbial modification: The view from modern type theories. *J. of Logic, Language and Information*, 26, 2017.

[20] S. Chatzikyriakidis and Z. Luo. Identity criteria of common nouns and dot-types for copredication. *Oslo Studies in Language*, 10(2), 2018.

[21] S. Chatzikyriakidis and Z. Luo. *Formal Semantics in Modern Type Theories.* Wiley & ISTE Science Publishing Ltd., 2019. (to appear).

[22] A. Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5(1), 1940.

[23] R. Cooper. Records and record types in semantic theory. *J. Logic and Compututation*, 15(2), 2005.

[24] The Coq Development Team. *The Coq Proof Assistant Reference Manual (Version 8.3), INRIA*, 2010.

[25] Th. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2/3), 1988.

[26] M. Cresswell. Hyperintensional logic. *Studia Logica*, 34, 1975.

[27] H.B. Curry and R. Feys. *Combinatory Logic*, volume 1. North Holland Publishing Company, 1958.

[28] R. Dapoigny and P. Barlatier. Modeling contexts with dependent types. *Fundamenta Informaticae*, 21, 2009.

[29] R. Dapoigny and P. Barlatier. Modelling contexts with dependent types. *Fundamenta Informaticae*, 104, 2010.

[30] D. Dowty. *IWord Meaning and Montague Grammar.* D. Reidel Publishing Co., 1979.

[31] M. Dummett. *The Logical Basis of Metaphysics.* Duckworth, 1991.

[32] R. Fox and S. Lappin. *Foundations of Intensional Semantics.* Blackwell, 2005.

[33] D. Gallin. *Intensional and higher-order modal logic: with applications to Montague semantics. North-Holland, Amsterdam.* 1975.

[34] Nicola Gambino and Peter Aczel. The generalised type-theoretic interpretation of constructive set theory. *J. Symbolic Logic*, 71(1):67–103, 2006.

[35] G. Gentzen. Untersuchungen über das logische schliessen. *Mathematische Zeitschrift*, 39, 1934.

[36] J. Grudzińska and M. Zawadowski. Generalized quantifiers on dependent types: A system for anaphora. *S. Chatzikyriakidis and Z. Luo (eds.). Modern Perspectives in Type-Theoretical Semantics*, 2017.

[37] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[38] W. A. Howard. The formulae-as-types notion of construction. In J. Hindley and J. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic*. Academic Press, 1980.

[39] R. Kahle and P. Schroeder-Heister, editors. *Proof-Theoretic Semantics.* Special Issue of Synthese, 148(3), 2006.

[40] G. Lungu. *Subtyping in Signatures.* PhD thesis, Royal Holloway, Univ. of London, 2018.

[41] G. Lungu and Z. Luo. Monotonicity reasoning in formal semantics based on modern type theories. *Logical Aspects of Computational Linguistics 2014 (LACL 2014), Toulouse. LNCS 8535*, 2014.

[42] G. Lungu and Z. Luo. On subtyping in type theories with canonical objects. post-proceedings of the 22nd int. conf. on types for proofs and programs (types 2016), leibniz international proceedings in informatics. 2018.

[43] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science.* Oxford University Press, 1994.

[44] Z. Luo. Coercive subtyping. *Journal of Logic and Computation*, 9(1):105–130, 1999.

[45] Z. Luo. A type-theoretic framework for formal reasoning with different logical foundations. In M. Okada and I. Satoh, editors, *Proc of the 11th Annual Asian Computing Science Conference*, volume 4435 of *LNCS*, pages 179–194. Tokyo, 2006.

[46] Z. Luo. Manifest fields and module mechanisms in intensional type theory. In S. Berardi, F. Damiani, and U. de'Liguoro, editors, *Types for Proofs and Programs, Proc. of Inter. Conf. of TYPES'08, LNCS 5497.*, 2009.

[47] Z. Luo. Type-theoretical semantics with coercive subtyping. *Semantics and Linguistic Theory 20 (SALT20), Vancouver*, 2010.

[48] Z. Luo. Adjectives and adverbs in type-theoretical semantics. Notes, 2011.

[49] Z. Luo. Contextual analysis of word meanings in type-theoretical semantics. *Logical Aspects of Computational Linguistics (LACL'2011). LNAI 6736*, 2011.

[50] Z. Luo. Common nouns as types. In D. Bechet and A. Dikovsky, editors, *Logical Aspects of Computational Linguistics (LACL'2012). LNCS 7351*, 2012.

[51] Z. Luo. Formal Semantics in Modern Type Theories: Is It Model-theoretic, Proof-theoretic, or Both? Invited talk at Logical Aspects of Computational Linguistics 2014 (LACL 2014), Toulouse. LNCS 8535 (pp 177-188). 2014.

[52] Z. Luo. Formal semantics in modern type theories with coercive subtyping. *Linguistics and Philosophy*, 35(6):491–513, 2012.

[53] Z. Luo. Proof irrelevance in type-theoretical semantics. *Logic and Algorithms in Computational Linguistics 2018 (LACompLing2018), Studies in Computational Intelligence (SCI)*, 2019.

[54] Z. Luo and P. Callaghan. Coercive subtyping and lexical semantics (extended abstract). *LACL'98 (extended abstracts), available in request to the first author or as* `http://www.cs.rhul.ac.uk/home/zhaohui/LACL98.abstract.ps`, 1998.

[55] Z. Luo and F. Part. Subtyping in type theory: Coercion contexts and local coercions (extended abstract). *TYPES 2013, Toulouse*, 2013.

[56] Z. Luo and R. Pollack. LEGO Proof Development System: User's Manual. LFCS Report ECS-LFCS-92-211, Dept of Computer Science, Univ of Edinburgh, 1992.

[57] Z. Luo, S. Soloviev, and T. Xue. Coercive subtyping: theory and implementation. *Information and Computation*, 223:18–42, 2012.

[58] P. Martin-Löf. An intuitionistic theory of types: predicative part. In H.Rose and J.C.Shepherdson, editors, *Logic Colloquium'73*, 1975.

[59] P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, 1984.

[60] P. Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1), 1996.

[61] R. Montague. *Formal Philosophy.* Yale University Press, 1974. Collected papers edited by R. Thomason.

[62] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: An Introduction.* Oxford University Press, 1990.

[63] G. Nunberg. Transfers of meaning. *Journal of Semantics*, 12(2):109–132, 1995.

[64] B. Partee. Possible Worlds in Model-Theoretic Semantics: A Linguistic Perspective. In A. Sture, editor, *Possible Worlds in Humanities, Arts and Sciences*, 1988.

[65] C. Pollard. Hyperintensions. *Journal of Logic and Computation*, 18(2), 2008.

[66] P. Portner. *What is Meaning? Foundamentals of Formal Semantics.* Blackwell, 2005.

[67] D. Prawitz. *Natural Deduction, a Proof-Theoretic Study.* Lmqvist and Wiksell, 1965.

[68] D. Prawitz. Towards a foundation of a general proof theory. In P. Suppes *et al.*, editor, *Logic, Methodology, and Phylosophy of Science IV*, 1973.

[69] D. Prawitz. On the idea of a general proof theory. *Synthese*, 27, 1974.

[70] J. Pustejovsky. *The Generative Lexicon.* MIT, 1995.

[71] F.P. Ramsey. The foundations of mathematics. *Proceedings of the London Mathematical Society*, 25:338–384, 1925.

[72] A. Ranta. *Type-Theoretical Grammar.* Oxford University Press, 1994.

[73] A. Ranta. Constructive type theory. *In S. Lappin and C. Fox (eds.) The Handbook of Contemporary Semantic Theory (2nd ed.). Wiley-Blackwell.*, 2015.

[74] C. Retoré. The Montagovian generative lexicon $\lambda\mathsf{Ty}_n$: a type theoretical framework for natural language semantics. In R. Matthes and A. Schubert, editors, *Proc of TYPES2013*, 2013.

[75] B.A.W. Russell. *The Principles of Mathematics.* Routledge, 1903. Paperback edition, 1992.

[76] J. Saeed. *Semantics.* Wiley-Blackwell, 1997.

[77] G. Sundholm. Proof theory and meaning. *D. Gabbay and F. Guenthner (eds.). Handbook of Philosophical Logic, Vol III*, 1986.

[78] R. Thomason. A model theory for propositional attitudes. *Linguistics and Philosophy*, 4, 1980.

[79] A.N. White and B.A.W. Russell. *Principia Mathematica.* Cambridge University Press, 2nd edition, 1925.

[80] T. Xue and Z. Luo. Dot-types and their implementation. *Logical Aspects of Computational Linguistics (LACL 2012). LNCS 7351*, 2012.

[81] T. Xue, Z. Luo, and S. Chatzikyriakidis. Propositional forms of judgemental interpretations. *Proc of Workshop on Natural Language and Computer Science. Oxford*, 2018.

A Formal Rules of Some Type Constructors

We give the inference rules for several type constructors. They are presented as in MTTs without signatures. To obtain the rules of these types in an MTT with signatures, one can simply change $\vdash$ into $\vdash_\Sigma$.

A.1 Π-types

$$(\Pi)\qquad \frac{\Gamma \vdash A\ type \quad \Gamma,\ x{:}A \vdash B\ type}{\Gamma \vdash \Pi x{:}A.B\ type}$$

(Abs) $$\frac{\Gamma,\ x{:}A \vdash b : B}{\Gamma \vdash \lambda x{:}A.b : \Pi x{:}A.B}$$

(App) $$\frac{\Gamma \vdash f : \Pi x{:}A.B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : [a/x]B}$$

$(Conv)$ $$\frac{\Gamma,\ x{:}A \vdash b : B \quad \Gamma \vdash a : A}{\Gamma \vdash (\lambda x{:}A.b)(a) = [a/x]b : [a/x]B}$$

A.2 Disjoint Union Types

$(+)$ $$\frac{\Gamma \vdash A\ \mathit{type} \quad \Gamma \vdash B\ \mathit{type}}{\Gamma \vdash A + B\ \mathit{type}}$$

(Inl) $$\frac{\Gamma \vdash a : A \quad \Gamma \vdash B\ \mathit{type}}{\Gamma \vdash \mathit{inl}(a) : A + B}$$

(Inr) $$\frac{\Gamma \vdash b : B \quad \Gamma \vdash A\ \mathit{type}}{\Gamma \vdash \mathit{inr}(b) : A + B}$$

$(Case)$ $$\frac{\Gamma \vdash c : A + B\ \Gamma, x{:}A \vdash f(x) : C(\mathit{inl}(x))\ \Gamma, y{:}B \vdash g(y) : C(\mathit{inr}(y))}{\Gamma \vdash \mathit{case}(x.f(x),\ y.g(y),\ c) : C(c)}$$

$(Case1)$ $$\frac{\Gamma \vdash a : A\ \Gamma, x{:}A \vdash f(x) : C(\mathit{inl}(x))\ \Gamma, y{:}B \vdash g(y) : C(\mathit{inr}(y))}{\Gamma \vdash \mathit{case}(x.f(x),\ y.g(y),\ \mathit{inl}(a)) = f(a) : C(\mathit{inl}(a))}$$

$(Case2)$ $$\frac{\Gamma \vdash b : B\ \Gamma, x{:}A \vdash f(x) : C(\mathit{inl}(x))\ \Gamma, y{:}B \vdash g(y) : C(\mathit{inr}(y))}{\Gamma \vdash \mathit{case}(x.f(x),\ y.g(y),\ \mathit{inr}(b)) = g(b) : C(\mathit{inr}(b))}$$

A.3 *Prop* and Logical Propositions

$(Prop1)$ $$\frac{\Gamma\ \mathit{valid}}{\Gamma \vdash \mathit{Prop}\ \mathit{type}}$$

$(Prop2)$ $$\frac{\Gamma \vdash A : \mathit{Prop}}{\Gamma \vdash A\ \mathit{type}}$$

$(\forall)$ $$\frac{\Gamma \vdash A\ \mathit{type} \quad \Gamma,\ x{:}A \vdash P : \mathit{Prop}}{\Gamma \vdash \forall x{:}A.P : \mathit{Prop}}$$

(Abs_P) $$\frac{\Gamma,\ x{:}A \vdash b : P}{\Gamma \vdash \lambda x{:}A.b : \forall x{:}A.P}$$

(App_P) $$\frac{\Gamma \vdash f : \forall x{:}A.P \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : [a/x]P}$$

$(Conv_P)$ $$\frac{\Gamma,\ x{:}A \vdash b : P \quad \Gamma \vdash a : A}{\Gamma \vdash (\lambda x{:}A.b)(a) = [a/x]b : [a/x]B}$$

A.4 The Parameterised Unit Type

$(F_{\mathbf{1}})$ $$\frac{\Gamma \vdash A\ \mathit{type} \quad \Gamma \vdash a : A}{\Gamma \vdash \mathbf{1}_A(a)\ \mathit{type}}$$

$(I_{\mathbf{1}})$ $$\frac{\Gamma \vdash A\ \mathit{type} \quad \Gamma \vdash a : A}{\Gamma \vdash *_A(a) : \mathbf{1}_A(a)}$$

$(E_{\mathbf{1}})$ $$\frac{\Gamma \vdash A\ \mathit{type} \quad \Gamma \vdash a : A \quad \Gamma \vdash c : C(*_A(a)) \quad \Gamma, x{:}\mathbf{1}_A(a) \vdash C(x)\ \mathit{type}}{\mathcal{E}(A,a,C,c)\ :\ (x{:}\mathbf{1}_A(a))C(x)}$$

$(C_{\mathbf{1}})$ $$\frac{\Gamma \vdash A\ \mathit{type} \quad \Gamma \vdash a : A \quad \Gamma \vdash c : C(*_A(a)) \quad \Gamma, x{:}\mathbf{1}_A(a) \vdash C(x)\ \mathit{type}}{\mathcal{E}(A,a,C,c,*_A(a)) = c\ :\ C(*_A(a))}$$

B The Formal System LF_Σ

LF$_\Sigma$ extends LF, the logical framework described in Chapter 9 of [43], by adding signatures to its judgement forms and inference rules. In the logical framework LF$_\Sigma$ (and LF), types are called *kinds* in order to be distinguished from types in the object type theory. Type theories can be specified in a logical framework such as Martin-Löf's logical framework [62] or its typed version LF [43]. We extend LF with signatures to obtain LF$_\Sigma$.

B.1 LF$_\Sigma$

LF$_\Sigma$ has the following six forms of judgements:

- Σ *valid*, which asserts that Σ is a valid signature.
- $\vdash_\Sigma \Gamma$, which asserts that Γ is a valid context under Σ.
- $\Gamma \vdash_\Sigma K$ *kind*, which asserts that K is a kind in Γ under Σ.
- $\Gamma \vdash_\Sigma k : K$, which asserts that k is an object of kind K in Γ under Σ.
- $\Gamma \vdash_\Sigma K_1 = K_2$, which asserts that K_1 and K_2 are equal kinds in Γ under Σ.

- $\Gamma \vdash_\Sigma k_1 = k_2 : K$, which asserts that k_1 and k_2 are equal objects of kind K in Γ under Σ.

The inference rules of LF$_\Sigma$ are given below, where $\langle\rangle$ is the empty sequence and $dom(p_1 : K_1,\ ...\ p_n : K_n) = \{p_1, ..., p_n\}$.[45]

Signatures and Contexts[46]

$$\frac{}{\langle\rangle\ valid} \qquad \frac{\langle\rangle \vdash_\Sigma K\ kind \quad c \notin dom(\Sigma)}{\Sigma,\ c : K\ valid} \qquad (*)\ \frac{\vdash_{\Sigma,c:K,\Sigma'} \Gamma}{\Gamma \vdash_{\Sigma,c:K,\Sigma'} c : K}$$

$$\frac{\Sigma\ valid}{\vdash_\Sigma \langle\rangle} \qquad \frac{\Gamma \vdash_\Sigma K\ kind \quad x \notin dom(\Gamma)}{\vdash_\Sigma \Gamma,\ x : K} \qquad (*)\ \frac{\vdash_\Sigma \Gamma, x : K, \Gamma'}{\Gamma, x : K, \Gamma' \vdash_\Sigma x : K}$$

Equality Rules

$$\frac{\Gamma \vdash_\Sigma K\ \ kind}{\Gamma \vdash_\Sigma K = K} \quad \frac{\Gamma \vdash_\Sigma K = K'}{\Gamma \vdash_\Sigma K' = K} \quad \frac{\Gamma \vdash_\Sigma K = K' \ \ \Gamma \vdash_\Sigma K' = K''}{\Gamma \vdash_\Sigma K = K''}$$

$$\frac{\Gamma \vdash_\Sigma k : K}{\Gamma \vdash_\Sigma k = k : K} \quad \frac{\Gamma \vdash_\Sigma k = k' : K}{\Gamma \vdash_\Sigma k' = k : K} \quad \frac{\Gamma \vdash_\Sigma k = k' : K \ \ \Gamma \vdash_\Sigma k' = k'' : K}{\Gamma \vdash_\Sigma k = k'' : K}$$

$$\frac{\Gamma \vdash_\Sigma k : K \ \ \Gamma \vdash_\Sigma K = K'}{\Gamma \vdash_\Sigma k : K'} \quad \frac{\Gamma \vdash_\Sigma k = k' : K \ \ \Gamma \vdash_\Sigma K = K'}{\Gamma \vdash_\Sigma k = k' : K'}$$

The Kind Type

$$\frac{\Gamma\ \ valid}{\Gamma \vdash_\Sigma Type\ \ kind} \quad \frac{\Gamma \vdash_\Sigma A : Type}{\Gamma \vdash_\Sigma El(A)\ \ kind} \quad \frac{\Gamma \vdash_\Sigma A = B : Type}{\Gamma \vdash_\Sigma El(A) = El(B)}$$

Dependent Product Kinds

$$\frac{\Gamma \vdash_\Sigma K\ \ kind \ \ \Gamma, x{:}K \vdash_\Sigma K'\ \ kind}{\Gamma \vdash_\Sigma (x{:}K)K'\ \ kind} \quad \frac{\Gamma \vdash_\Sigma K_1 = K_2 \ \ \Gamma, x{:}K_1 \vdash_\Sigma K_1' = K_2'}{\Gamma \vdash_\Sigma (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

$$\frac{\Gamma, x{:}K \vdash_\Sigma k : K'}{\Gamma \vdash_\Sigma [x{:}K]k : (x{:}K)K'} \quad \frac{\Gamma \vdash_\Sigma K_1 = K_2 \ \ \Gamma, x{:}K_1 \vdash_\Sigma k_1 = k_2 : K}{\Gamma \vdash_\Sigma [x{:}K_1]k_1 = [x{:}K_2]k_2 : (x{:}K_1)K}$$

$$\frac{\Gamma \vdash_\Sigma f : (x{:}K)K' \ \ \Gamma \vdash_\Sigma k : K}{\Gamma \vdash_\Sigma f(k) : [k/x]K'} \quad \frac{\Gamma \vdash_\Sigma f = f' : (x{:}K)K' \ \ \Gamma \vdash_\Sigma k_1 = k_2 : K}{\Gamma \vdash_\Sigma f(k_1) = f'(k_2) : [k_1/x]K'}$$

$$\frac{\Gamma, x{:}K \vdash_\Sigma k' : K' \ \ \Gamma \vdash_\Sigma k : K}{\Gamma \vdash_\Sigma ([x{:}K]k')(k) = [k/x]k' : [k/x]K'} \quad \frac{\Gamma \vdash_\Sigma f : (x{:}K)K' \ \ x \notin FV(f)}{\Gamma \vdash_\Sigma [x{:}K]f(x) = f : (x{:}K)K'}$$

[45]For those who are familiar with LF, it may be easier to note that, besides the rules for signatures, all of the inference rules of LF (i.e., those in Figures 9.1 and 9.2 of Chapter 9 of [43]) become inference rules of LF$_\Sigma$ after replacing $\vdash$ by $\vdash_\Sigma$ (and changing the judgement form 'Γ *valid*' to '$\vdash_\Sigma \Gamma$').

[46]Note that the assumptions in a signature or in a context can be derived – this is characterised by the rules marked by $(*)$.

B.2 Σ-types in LF$_\Sigma$

In LF$_\Sigma$, various type constructors can be specified by declaring constants as done in LF [43]. For instance, the specification of Σ-types in LF$_\Sigma$ can be specified by declaring the following constants:

$$\begin{array}{rcl} \Sigma & : & (A{:}Type)(B{:}(A)Type)Type \\ pair & : & (A{:}Type)(B{:}(A)Type)(x{:}A)(y{:}B(x)\Sigma(A,B) \\ \pi_1 & : & (A{:}Type)(B{:}(A)Type)(p{:}\Sigma(A,B))A \\ \pi_2 & : & (A{:}Type)(B{:}(A)Type)(p{:}\Sigma(A,B))B(\pi_1(p)) \end{array}$$

with $\pi_1(A,B,pair(A,B,a,b)) = a$ and $\pi_2(A,B,pair(A,B,a,b)) = b$. Then, the rules in Figure 1 are all derivable (note that, for pairs (a,b), we use $pair(a,b)$ in LF$_\Sigma$-notation). Detailed explanations are omitted.

C The Finite Types $Fin(n)$

Let Nat be the type of natural numbers. Then, $Fin(n)$, parameterised by natural number n, consists of n objects. The introduction rules for $Fin(n)$ are as follows.

$$\frac{\Gamma \vdash_\Sigma n : Nat}{\Gamma \vdash_\Sigma zero(n) : Fin(n+1)} \qquad \frac{\Gamma \vdash_\Sigma n : Nat \quad \Gamma \vdash_\Sigma i : Fin(n)}{\Gamma \vdash_\Sigma succ(n,i) : Fin(n+1)}$$

We omit its elimination and computation rules. Abusing the language, one may adopt her/his own choice to denote the objects of a finite type. For instance, in Example 5.2 in §5.2.2, *John*, *Paul*, *George*, *Ringo*, *Brian*, and *Bob* are used to denote the objects of a_D which is $Fin(6)$.