1

Type Theories and Semantic Studies

The long history of the study of semantics has produced a number of theories of meaning. For example, the referential theory adopts a Platonic viewpoint and proposes that meanings are entities in the world; the internalist theory, such as that held by Chomsky, suggests that meanings are concepts in our minds, and the use theory, which is closely related to Wittgenstein's slogan of "meaning is use", advocates that meanings are embodied in the ways that language is used in social practice. Besides being very interesting themselves, these philosophical theories have had a profound impact on the ways in which researchers think of and approach formal semantics. For example, many semanticists have been influenced by the referential theory of meaning and believed that formal semantics should be model-theoretic (see, for instance, Portner (2005)), following Tarski's ideas in model theory for logical systems and Montague's ideas in set-theoretical semantics for natural language (Montague 1974). On the other hand, the use theory of meaning has convinced many others and has been substantially developed more recently, both by philosophers such as Dummett (1991) and Brandom (1994, 2000) on meaning theories in general and by logicians such as Gentzen (1935), Prawitz (1973, 1974) and Martin-Löf (1984, 1996) on proof-theoretic semantics for logical systems in particular.

This book studies formal semantics in modern type theories (MTT-semantics for short). We contend that MTT-semantics is both model-theoretic and proof-theoretic, a thesis that was first proposed by the second author in Luo (2014) and further elaborated in Luo (2019a), and will be explicated in this book. For natural language, both model-theoretic and proof-theoretic semantics have their own advantages and disadvantages, and it is not easy to imagine how one can combine them in a single semantic framework: in fact, up to now this has never been attempted, let alone done. We argue that MTT-semantics is the first framework with both characteristics: being model-theoretic, it provides powerful mechanisms to capture semantics for a wide range of linguistic features, and being proof-theoretic, it has a solid

meaning-theoretic foundation and can be directly implemented by means of the current proof technology to support computer-assisted reasoning in natural language. This gives MTT-semantics unprecedented advantages over other semantic frameworks.

In this chapter, we shall start with a brief account of the historical development of type theory for the study of the foundations of mathematics – the simple type theory for classical mathematics and dependent (modern) type theories for constructive mathematics. Simple type theory was employed by Montague and his followers as an intermediate language for the study of model-theoretic semantics of natural language, where set theory is taken as the foundational language. In MTT-semantics, on the other hand, modern type theories (MTTs) are themselves foundational languages. The new logical concepts and rich typing mechanisms in MTTs make them adequate to serve as foundational languages for formal semantics. We shall introduce MTT-semantics briefly, which will be developed further in the book, and summarize its advantages.

1.1. Historical development of type theories

Type theories are computational logical systems that were originally developed for the foundations of mathematics. At the beginning of the 20th Century, Russell (1903) developed the theory of types to solve a foundational problem of mathematics exposed as a number of well-known paradoxical contradictions in Cantor's naive set theory that are related to self-reference. Some researchers, including Russell himself, attributed such paradoxes to "vicious circles" in formations of logical formulae ("impredicativity", in a technical jargon), which is what Russell's Ramified Theory of Types (Whitehead and Russell 1925) was designed to circumvent.

However, Ramsey (1926) pointed out that it was the logical paradoxes such as Russell's paradox, not the semantic paradoxes such as Liar's paradox, that can (and should) be avoided in formulations of logical calculi and that Russell had mixed up these two kinds of paradoxes, leading to complications and problems in his ramified theory of types. As Ramsey argued, although impredicativity in formula formations is circular, it is not vicious. Based on this, Ramsey suggested[1] that the ramified theory of types can be "simplified" into Simple Type Theory, which was later formally formulated by Church (1940) using the λ-notation and used by Montague (1973) in his Intensional Logic (IL) to study the formal semantics of natural language.

The early developments of type theory, including those by Russell and Ramsey as mentioned above, were driven by the search for foundational languages for classical

1 Attribution for the development of simple type theory should also go to the Polish logician Chwistek (see, for example, Linsky 2009).

mathematics. In the 1970s, various logicians, notably Feferman, Friedman, Martin-Löf and Myhill, among others, studied foundational languages for constructive rather than classical mathematics. Besides other systems, Martin-Löf's type theory (Martin-Löf 1975, 1984) stood out with several new ground-breaking concepts that were not present in traditional logical systems. It adopts the notions of context, judgment and definitional equality, and introduces powerful typing mechanisms such as dependent types, inductive types and type universes. It also makes use of the principle of propositions as types, also called the Curry–Howard correspondence (Curry and Feys 1958; Howard 1980), to incorporate an embedded logic in the type system. These features have not only made Martin-Löf's type theory a very interesting candidate for the foundation of constructive mathematics but, more importantly, opened up new avenues to study dependent type theories as attractive foundational languages for various other applications such as computer-assisted reasoning and linguistic semantics.

In particular, the study of Martin-Löf's type theory, together with that of Church's simple type theory, has led to the development of a family of (intensional) type theories that we call MTTs,[2] including Martin-Löf's predicative type theory (Martin-Löf 1975; Nordström *et al*. 1990) and the impredicative type theories such as the Calculus of Constructions (Coquand and Huet 1988) and the Unifying Theory of dependent Types (UTT) (Luo 1994). In computer science, MTTs have been implemented in proof assistants, computer systems for formal proof development, such as Agda (2008), Coq (2010) and Lego/Plastic (Luo and Pollack 1992; Callaghan and Luo 2001), and used in applications to formalization of mathematics and verification of programs.

It is worth remarking that, although formalizing constructive mathematics motivated the early development of dependent type theory, it is not true that MTTs can only be employed constructively. Put in another way, powerful typing is not monopolized by constructive mathematics or constructive reasoning; instead, it can be used in much wider applications such as linguistic semantics: the MTT-semantics to be studied in this book is one such example.[3]

---

2 The term "modern type theory" is adopted to distinguish MTTs, as used in MTT-semantics, from Church's simple type theory, as used in Montague semantics, and it appeared for the first time in Luo (2009c).

3 In this respect, interested readers are referred to Logic-enriched Type Theories (LTTs), proposed by Aczel and Gambino in their study of type-theoretic interpretations of constructive set theory (Aczel and Gambino 2000; Gambino and Aczel 2006), and to the work by the second author and Adams on a type-theoretical framework of LTTs to support formal reasoning with different logical foundations, including classical inference as well as intuitionistic inference (Luo 2007; Adams and Luo 2010). Also related is the recent work on homotopy type theory where Martin-Löf's type theory is employed in such a way that the logic can be either intuitionistic or classical (HoTT 2013).

1.2. Foundational semantic languages

In the study of formal semantics, one considers a *foundational semantic language* in which semantic interpretations of natural language sentences and phrases are given; in other words, it is meaning-carrying language in the sense that its sentences/phrases interpret the natural language sentences/phrases. A foundational semantic language is usually a formal mathematical language that, besides being precise and usually unambiguous, has several important salient features, including the following:

1) the language has rich and powerful mechanisms and is hence capable of giving adequate semantic descriptions for a variety of diverse linguistic features;

2) the language contains (or embeds) a useful logic to be used in semantic interpretations;

3) the language is regarded as well understood (and, hence, so are the semantic interpretations).

In Montague's model-theoretic semantics (Montague 1973, 1974), axiomatic set theory is the foundational semantic language. First, it is clear to see that set theory is very powerful and capable of adequate semantic descriptions: this has been proved to be the case in the development of formal semantics in the Montagovian setting. Second, set theory incorporates a useful logic, which may be different from the logical system (e.g. first-order logic) based on which set theory is formulated. However, this is not very obvious, and it is one of the reasons that Montague introduced an intermediate language IL (Intensional Logic) (Montague 1973): it gives the syntax of a simple type theory in which, among other things, the syntax of usual logical operators is made explicit. Interpreting a natural language phrase in IL, we can indirectly obtain its set-theoretical interpretation from the semantics of IL in set theory (see, Henkin's (1950) model). The intermediate language IL makes the task of semantic description easier and clearer. In fact, as Gallin (1975) shows, although IL only formulates the syntax of logical operators and gives their meanings semantically in set theory, an alternative system TY_2 can be axiomatized as a variant of Church's simple type theory (Church 1940), of which we shall give a presentation below and discuss its use in Montague semantics.[4]

Whether set theory satisfies the third requirement of a foundational semantic language, that is, whether it is well understood, is subjective and less clear. Some may say that people have a rather good understanding of *naive* set theory. But, of

4 Although the logic in IL can be used to describe a lot of linguistic features, there are many others still found very difficult, if not impossible, to be characterized in IL (or simple type theory); in the literature, these phenomena are rather studied in set theory directly – see such "direct interpretations" in, for example, Winter (2016). This is one of the reasons that, for Montague semantics, set theory itself is considered the foundational semantic language, rather than the intermediate language IL.

course, this is not enough or even misleading – it is not the naive set theory in play here; rather, in Montague semantics, we use an axiomatic set theory as the foundational semantic language. It is fair to say that it is not easy to understand an axiomatic set theory such as ZFC (for example, for those familiar with formal set theory, think of the complicated and non-intuitive axioms in set theory!)

In MTT-semantics, modern type theories are foundational semantic languages (see, the notes on the related historical development in section 1.4.2). They have powerful mechanisms and rich type structures: this satisfies our first requirement – an MTT is a powerful language for formal semantics. In particular, like sets in set theory, types represent collections and the rich typing mechanisms in MTTs provide powerful tools to describe various linguistic features. Although types in MTTs are different from sets in set theory, they provide powerful mechanisms for formal semantics, as to be demonstrated in this book, among other things. We say that, although MTTs are specified proof-theoretically, MTT-semantics is model-theoretic – for formal semantics, the rich typing mechanisms in MTTs are powerful, just like the set-theoretical mechanisms in set theory.[5]

In MTTs, the correctness of typing judgments of the form $a : A$, stating that a is of type A, is decidable in the sense that we can mechanically decide whether such a judgment can be correctly made. For computer scientists, this is equivalent to saying that a computer system can automatically decide whether a is of type A. Thanks to this decidability result and because of the rich typing mechanisms in MTTs, the Curry–Howard principle of propositions-as-types (Curry and Feys 1958; Howard 1980) gives us an embedded logical mechanism for semantic interpretations – the second requirement above. In this book, we shall elaborate this in detail to illustrate how the logical mechanisms work and how the typing mechanisms facilitate semantic formalizations.

MTTs are specified by means of proof-theoretic rules (see Chapter 2) and, because of this, there are two advantageous facets that are not available in previous set-theoretic semantics: the first is that an MTT can have a use theory of meaning in that its judgments (sentences) can be understood proof-theoretically by means of their inferential uses. Such a proof-theoretic understanding of a foundational semantic language was not available before: we cannot understand set theory in such a way. Therefore, this offers us a new possibility: we may claim that an MTT, as the foundational semantic language, is well understood by means of its proof-theoretic meaning theory. Second, because they are proof-theoretically specified, MTTs (and

5 A remark may be necessary here: we are only emphasizing the similarities between type theory and set theory here. However, they are very different – for example, it is their difference that enables us to say that MTT-semantics is also proof-theoretic and can be successfully implemented on computers for natural language reasoning. For some readers, the difference will only be apparent later on, say, after reading Chapter 2 to become more familiar with MTTs.

hence MTT-semantics) can be readily implemented on computers to support computer-assisted reasoning in natural language. In fact, this is supported by the current proof technology provided by the proof assistants based on MTTs, as mentioned above, and we shall describe how MTT-semantics can be implemented on computers to perform reasoning tasks in natural language.

1.3. Montague's model-theoretic semantics

Since its development in the late 1960s and early 1970s, Montague semantics (Montague 1974) has been the dominant approach to formal semantics. There are several textbooks about Montague semantics including, for example, that by Dowty *et al.* (1981). As explained above, Montague has introduced an intermediate language, called Intensional Logic (IL), for his model-theoretic semantics. For instance, to interpret the sentence in (1.1), we could first give its interpretation (1.2) in IL, where the semantics of John is an entity j and the semantic interpretation $talk$ of the verb "talk" is a predicate over entities of the world, which can be applied to j to form the interpretation (1.2):

(1.1) John talks.

(1.2) $talk(j)$

This then gives the set-theoretical interpretation of (1.1), i.e. the interpretation of (1.2) in set theory (according to, for example, Henkin's model interpretation), which says that the set-theoretic interpretation of j is a member of the set that interprets the predicate *talk*.

In what follows, we shall first describe simple type theory[6] and then how it is used in Montague semantics.

1.3.1. *Simple type theory: a formal description*

Montague's IL (Montague 1973; Gallin 1975) consists of an extensional core, which is Church's simple type theory (Church 1940) (we call it $\mathcal{C}$ in this book, with $\mathcal{C}$ standing for "Church"), and a part concerning intensionality.[7] We shall focus on describing the former and then briefly comment on the part about intensionality.

6 Our description of simple type theory is formal and, in section 7.2, will be extended with dependent event types, one of the applications of dependent types in Davidsonian event semantics.

7 IL as described in Montague (1973) also contains a modal operator used to describe tense. As noted in Montague (1973), it can be considered as a special predicate symbol to be interpreted as intended. We omit it here.

Our description of Church's simple type theory $\mathcal{C}$ follows that by Luo and Soloviev (2017), where it is described by means of natural deduction rules that derive judgments – sentences in the type theory. For instance, a judgment may be of the form $\Gamma \vdash a : A$, which means that a is of type A under the assumptions described in context Γ. We shall first explain what contexts and judgments are in $\mathcal{C}$, and then describe its rules.[8] (All of $\mathcal{C}$'s inference rules are listed in Appendix A1.1.)

Contexts and Judgments. A context is a sequence of entries either of the form $x : A$ or of the form $P\ true$. Informally, the former assumes that the variable x be of type A and the latter that the proposition P be true. Only valid contexts are legal and context validity is governed by the following rules:

$$\frac{}{\langle\rangle\ valid} \qquad \frac{\Gamma \vdash A\ type \quad x \notin FV(\Gamma)}{\Gamma,\ x{:}A\ \ valid} \qquad \frac{\Gamma \vdash P : \mathbf{t}}{\Gamma,\ P\ true\ \ valid}$$

where $\langle\rangle$ is the empty sequence and $FV(\Gamma)$ is the set of free variables in Γ, defined as: (1) $FV(\langle\rangle) = \emptyset$; (2) $FV(\Gamma, x{:}A) = FV(\Gamma) \cup \{x\}$; (3) $FV(\Gamma, P\ true) = FV(\Gamma)$.

Judgments are sentences of $\mathcal{C}$, whose correctness is governed by the inference rules below. In $\mathcal{C}$, there are four forms of judgments:

– $\Gamma\ valid$, which means that Γ is a valid context (the rules of deriving context validity are given above);

– $\Gamma \vdash A\ type$, which means that A is a type under context Γ;

– $\Gamma \vdash a : A$, which means that a is an object of type A under context Γ;

– $\Gamma \vdash P\ true$, which means that P is a true formula under context Γ.

Inference rules. Besides the context validity rules given above, $\mathcal{C}$ has the following rules:

– Rules for base types: **e** and **t** are the types of entities and logical formulae, respectively.[9]

$$\frac{\Gamma\ valid}{\Gamma \vdash \mathbf{e}\ type} \qquad \frac{\Gamma\ valid}{\Gamma \vdash \mathbf{t}\ type}$$

8 For those who are unfamiliar with natural deduction rules, it may be worth remarking that the rules below define a natural deduction system whose sentences are called judgments. A judgment J is correct (formally called derivable) if there is a derivation of J, that is, a finite sequence of judgments $J_1,\ ...,\ J_n$ with $J_n = J$ such that, for all $1 \leq i \leq n$, J_i is the conclusion of an instance of some inference rule whose premises are in $\{\ J_k \mid k < i\ \}$.

9 Here, the names of the base types **e** and **t** follow the Montagovian tradition. In Church (1940), **e** and **t** are named ι and o, respectively.

– Assumption rules: one can prove what have been assumed in valid contexts.

$$\frac{\Gamma,\ x{:}A,\ \Gamma'\ valid}{\Gamma,\ x{:}A,\ \Gamma' \vdash x : A} \qquad \frac{\Gamma,\ P\ true,\ \Gamma'\ valid}{\Gamma,\ P\ true,\ \Gamma' \vdash P\ true}$$

– Rules for function types: The formation rule (of function types), introduction rule (of λ-abstraction) and elimination rule (for applications) are as follows, where the terms in the forms of λ-abstractions and applications are related to each other computationally by the relation of β-conversion.[10]

$$\frac{\Gamma \vdash A\ type\ \ \Gamma \vdash B\ type}{\Gamma \vdash A \to B\ type} \qquad \frac{\Gamma, x{:}A \vdash b : B}{\Gamma \vdash \lambda x{:}A.b : A \to B} \qquad \frac{\Gamma \vdash f : A \to B\ \ \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}$$

– Rules for logical formulae formed by implication and universal quantification: their formation, introduction and elimination rules.

$$\frac{\Gamma \vdash P : \mathbf{t}\ \ \Gamma \vdash Q : \mathbf{t}}{\Gamma \vdash P \Rightarrow Q : \mathbf{t}} \qquad \frac{\Gamma,\ P\ true \vdash Q\ true}{\Gamma \vdash P \Rightarrow Q\ true} \qquad \frac{\Gamma \vdash P \Rightarrow Q\ true\ \ \Gamma \vdash P\ true}{\Gamma \vdash Q\ true}$$

$$\frac{\Gamma \vdash A\ type\ \ \Gamma, x{:}A \vdash P : \mathbf{t}}{\Gamma \vdash \forall x{:}A.P : \mathbf{t}} \qquad \frac{\Gamma, x{:}A \vdash P\ true}{\Gamma \vdash \forall x{:}A.P\ true} \qquad \frac{\Gamma \vdash \forall x{:}A.P(x)\ true\ \ \Gamma \vdash a : A}{\Gamma \vdash P(a)\ true}$$

where, in the last rule, $P(a)$ is obtained by substituting a for x in $P(x)$.

– Conversion rule for truth of formulas (see footnote 10 for the conversion relation $\simeq$):

$$\frac{\Gamma \vdash P\ true\ \ \Gamma \vdash Q : \mathbf{t}}{\Gamma \vdash Q\ true}\ \ (P \simeq Q)$$

Logical operators. The other usual logical operators can be defined by means of $\Rightarrow$ and $\forall$. For example, the operators for conjunction and existential quantification can be defined as follows. Other operators such as disjunction and negation can be defined similarly (see Appendix A1.2).

(1.3) $P \wedge Q = \forall X : \mathbf{t}.\ (P \Rightarrow Q \Rightarrow X) \Rightarrow X$

(1.4) $\exists x : A.P(x) = \forall X : \mathbf{t}.\ (\forall x : A.(P(x) \Rightarrow X)) \Rightarrow X$

10 As in λ-calculus, β-conversion holds: we have that $(\lambda x{:}A.b[x])(a)$ is convertible to $b[a]$, notation $(\lambda x{:}A.b[x])(a) \simeq b[a]$, where $b[a]$ is obtained from $b[x]$ by substituting a for all free occurrences of x. In other words, these two expressions $(\lambda x{:}A.b[x])(a)$ and $b[a]$ are computationally the same – the former computes to the latter, and the relation $\simeq$ of β-conversion is the reflexive, symmetric and transitive closure of this basic computational relation.

	Example	Montague semantics
CN	man, human	$man,\ human : \mathbf{e} \to \mathbf{t}$
IV	talk	$talk : \mathbf{e} \to \mathbf{t}$
ADJ	handsome	$handsome : (\mathbf{e} \to \mathbf{t}) \to (\mathbf{e} \to \mathbf{t})$
ADV_{VP}	quickly	$quickly : (\mathbf{e} \to \mathbf{t}) \to (\mathbf{e} \to \mathbf{t})$
Modified CN	handsome man	$handsome(man) : \mathbf{e} \to \mathbf{t}$
Quantifier	some	$some : (\mathbf{e} \to \mathbf{t}) \to (\mathbf{e} \to \mathbf{t}) \to \mathbf{t}$
S	A man talks	$\exists x : \mathbf{e}.\ man(x)\ \&\ talk(x) : \mathbf{t}$

Table 1.1. *Examples in Montague semantics*

1.3.2. *Montague semantics: examples and intensionality*

Simple examples in Montague semantics can be found in Table 1.1, briefly showing how to interpret some of the basic linguistic categories in the Montagovian setting, including common nouns (CN), verbs (IV), adjectives (ADJ), verb-modifying adverbs (ADV_{VP}), modified CNs, quantifiers and sentences (S). In the following section, we can find Table 1.3 with examples for these categories interpreted in MTT-semantics. For example, in Montague semantics, verb phrases are interpreted as predicates of type $\mathbf{e} \to \mathbf{t}$ and sentences as formulas of type $\mathbf{t}$, as (1.5) and (1.6) illustrate (see, the lines for IV and S in Table 1.1):

(1.5) $talk : \mathbf{e} \to \mathbf{t}$

(1.6) $\exists x : \mathbf{e}.\ man(x) \wedge talk(x)$

A remark we should make is that, in type theories, typing judgments and logical formulas are different. In particular, a typing judgment is not a logical formula: (1.5) is a typing judgment stating that $talk$ is of type $\mathbf{e} \to \mathbf{t}$, while (1.6) is a logical formula of type $\mathbf{t}$. With such examples, the difference seems to be apparent and there is no need to be emphasized. However, this becomes more important in the setting of modern type theories where much richer typing mechanisms are employed (see, for example, Chapter 2).

A notational convention is adopted in this book: we shall use a different font to represent semantics of a natural language phrase. For instance, for the natural language words "man" and "talk", we shall use *man* and *talk* for their semantics (as in Table 1.1).

A linguistic feature that has been studied carefully in the Montagovian framework is intensionality, following the proposal made by Carnap (1947) that the intension of an expression be modeled by a function on possible states of affairs whose value, at a particular state, is the extension of the expression in that state. Nearly all constructions in English have some intensional instances (see, for example, a discussion on this in Partee (1988), among others). This has led Montague to give a special treatment of

intensionality in his IL. It can be treated by adding a special base type **s** to simple type theory $\mathcal{C}$, which allows one to consider types like $\mathbf{s} \to A$.[11]

$$\frac{\Gamma\ valid}{\Gamma \vdash \mathbf{s}\ type}$$

For instance, we may interpret the intension of a sentence as a function from possible worlds to truth values, of type $\mathbf{s} \to \mathbf{t}$. This approach to intensionality has been extensively studied, although it is not regarded as completely satisfactory in some aspects: for example, it suffers from the so-called problem of logical omniscience in that logically equivalent sentences are regarded as having the same meaning, which may be incorrect in an intensional context. In this book, we shall largely ignore the intensionality issue, with some exceptions,[12] not because it is not important (it obviously is), but rather that it is orthogonal to most, if not all, of the issues we are going to discuss.

1.4. MTT-semantics: formal semantics in modern type theories

In this section, some simple examples in formal semantics in modern type theories (MTT-semantics) are sketched for illustrations. Then, we shall describe the historical developments and some of the major merits for MTTs to be employed as foundational semantic languages.

1.4.1. *A glance at MTT-semantics*

Here, we sketch some simple examples of MTT-semantic interpretations, which give some ideas of what it is like, although a full-scale introduction to MTT-semantics (in Chapter 3) will not be possible until after the introduction to MTTs in Chapter 2. Let us start by considering the simple example (1.1), repeated here as (1.7), whose MTT-semantics is given in (1.8):

(1.7) John talks.

(1.8) $talk(j)$

In appearance, the MTT-interpretation (1.8) seems to be the same as the Montagovian interpretation (1.2). However, although similar, they have subtle differences, as summarized in Table 1.2.

---

11 This is slightly simpler than Montague's treatment in IL (Montague 1973), although it gives the same power. For its justification, see Gallin's work on TY_2 (Gallin 1975).

12 We shall deal with aspects of intensionality when discussing non-committal adjectives in section 3.3.4 and intensional adverbs in section 4.5.4.

	Type in Montague semantics	Type in MTT-semantics
j	**e**	$Human$
talk	**e** $\rightarrow$ **t**	$Human \rightarrow Prop$
talk(j)	**t**	$Prop$

Table 1.2. *Semantics of "John talks"*

In particular, the typings of these interpretations are different:

– In MTT-semantics, the sentence (1.7) is interpreted as the proposition $talk(j) : Prop$ where $Prop$ is the type of all logical propositions – an internal totality that only exists in impredicative type theories such as UTT.[13]

– In MTT-semantics, $talk(j)$ is a well-typed proposition because the semantics of "talk" is a predicate $talk : Human \rightarrow Prop$, whose domain is the type *Human*, the collection of humans to which *talk* can be meaningfully applied and to which *j* belongs as well. Note that, different from Montague semantics, the domain of *talk* is *Human*, rather than the type **e** of all entities.[14]

	Example	MTT-semantics
CN	man, human	$Man,\ Human : Type$
IV	talk	$talk : Human \rightarrow Prop$
ADJ	handsome	$handsome : Man \rightarrow Prop$
ADV_{VP}	quickly	$quickly : \Pi A{:}\mathsf{CN}.\ (A \rightarrow Prop) \rightarrow (A \rightarrow Prop)$
Modified CN	handsome man	$\Sigma m : Man.\ handsome(m) : Type$
Quantifier	some	$some : \Pi A{:}\mathsf{CN}.\ (A \rightarrow Prop) \rightarrow Prop$
S	A man talks	$\exists m : Man.\ talk(m) : Prop$

Table 1.3. *Examples in MTT-semantics*

MTT-semantic interpretations for various basic linguistic categories are exemplified in Table 1.3, where the natural language examples are the same as those in Table 1.1 for their Montague semantics. This makes it possible for a quick comparison in order for one to approach MTT-semantics more easily, albeit the understanding can be sketchy and imprecise at this stage. Here are brief explanations of the example interpretations in Table 1.3: some type-theoretical concepts are used

13 In predicative MTTs such as Martin-Löf's type theory, one would use a predicative universe which is only a type of some propositions, not all of them. Although we may relate $Prop$ to the type **t** of truth values in Church's simple type theory, they have subtle differences. The differences between MTTs and Church's simple type theory include whether the theory is classical or constructive and whether there are proof objects, among others. These are beyond the scope of our discussions here.

14 This constitutes the basis of dealing with selectional restriction by means of decidable typing, among many other things (more details later).

without explanation, including polymorphism, Σ-type and universe which will only be introduced in Chapter 2.

– A common noun (CN) can be interpreted as a type and the interpretations of CNs form a universe called CN, the type of the types that interpret CNs.

– A verb (IV) or an adjective (ADJ) can be interpreted as a predicate over a type D that interprets the domain of the verb or adjective, i.e. a function of type $D \to Prop$.

– A verb-modifying adverb (ADV_{VP}) can be interpreted as a polymorphic function from predicates of type $A \to Prop$ to predicates of the same type, where A ranges over CNs in the universe CN.

– Modified common nouns (modified CN), when the adjectives are intersective, can be interpreted by means of Σ-types of pairs.

– A quantifier is interpreted as a polymorphic function that takes a type A that interprets a common noun and a predicate over A, and returns a proposition.

– A sentence (S) can be interpreted as a proposition of type $Prop$.[15]

Please note that the semantic interpretations in Table 1.3 are only indicative with typical examples. In some cases, there are further elaborations or completely different ways of interpretation. For example, although CNs modified by intersective adjectives can be interpreted by Σ-types, adjectival modifications by means of other classes of adjectives may better be interpreted with the help of other type constructors (see section 3.3).

A semantic interpretation may be refined. For instance, the semantics of the word "man" may be defined as a Σ-type (1.9). In other words, Man does not have to be a constant type; instead, in (1.9), it is defined by means of type $Human$ and a predicate $male : Human \to Prop$: a man is a human who is male.

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

Similarly, although we have only showed in Table 1.3 the semantic typing of "some", generalized quantifiers can be defined in type theory (Sundholm 1989). In simple cases, they can be defined by existing quantifiers in type theory: for example, in UTT, *some* can be defined by means of the existential quantifier as shown in (1.10), where $A : \mathsf{CN}$, $P : A \to Prop$ and $\exists$ is the existential quantifier (see section 2.3.1 and Appendix A3.2 for its definition).

(1.10) $some(A, P) = \exists x{:}A.P(x)$

The type of *some* as defined in (1.10) is that as shown in Table 1.3.

15 In MTT-semantics, we also study judgmental interpretations of sentences and how to turn them into propositional forms – see section 3.2.3 for an informal introduction and section 7.1 for a formal treatment.

1.4.2. *MTTs as foundational semantic languages: historical notes*

The application of dependent type theory to natural language semantics can be traced back to the middle of the 1980s, when Mönnich (1985) and Sundholm (1986) showed how to use Martin-Löf's (extensional) type theory to deal with the problem of anaphoric representation as found necessary in interpreting donkey sentences such as those studied by Geach (1962). The key novelty is to use the type constructor Σ for existential quantification so that, for example, some of the donkey sentences involving intriguing anaphoric references can be interpreted as intended.[16] Offering an elegant solution to the long-standing problem in linguistic semantics and providing an alternative to dynamic semantics (Kamp 1981; Heim 1983; Groenendijk and Stokhof 1991), the type-theoretical approach has attracted attention of formal semanticists who have been puzzled by this problem for a long time.

A systematic study in applying Martin-Löf's (intensional) type theory (Martin-Löf 1975; Nordström *et al*. 1990) to formal semantics was not conducted until Ranta's work in the early 1990s (see Ranta (1994) and related papers). Although Ranta himself may not have regarded his work as studying logical semantics (see, for example, the preface of Ranta (1994)), his work is widely regarded as the first systematic and significant development in employing a modern type theory as a foundational semantic language, studying a variety of topics including anaphora, temporal reference and contextual environments. For example, following Mönnich and Sundholm's proposal of using types to represent common nouns, Ranta has carefully studied how to use Σ-types to represent (intersectively) modified CNs, pointing out that the CNs-as-types approach meets with the problem of multiple categorization of verbs, a crucial obstacle that calls for a solution.[17] Another topic

16 In Martin-Löf's type theory with propositions-as-types (the so-called PaT logic), existential quantification is represented by Σ, which is strong in the sense that an object of a Σ-type is a pair whose first component can be projected out as the witness of the truth of the existentially quantified sentence. For example, as shown in Sundholm (1986), the donkey sentence (1.11) can be interpreted as (1.12), where $Farmer$ and $Donkey$ are the types interpreting "farmer" and "donkey", respectively:

(1.11) Every farmer who owns a donkey beats it.

(1.12) $\Pi z : (\Sigma x{:}Farmer \Sigma y{:}Donkey.\ own(x, y)).\ beat(\pi_1(z), \pi_1(\pi_2(z)))$

Note that, in (1.12), π_i's are projection operators for Σ-types and, in particular, π_1 projects out the witness of a proof of an existential formula represented by a Σ-type (see section 2.2.2 for more details). However, such a use of Σ-types as both existential quantifier and a structural mechanism causes problems such as counting – see the second last paragraph in this section and footnotes 25/26 on p.15/p.16 for an example and explanations.

17 Ranta (1994) discussed several possible solutions but failed to obtain a satisfactory solution and, in particular, he did not realize that subtyping should be employed to solve the problem. In fact, coercive subtyping (Luo 1999) which, unlike the ordinary subsumptive subtyping, is

that has been carefully studied in Ranta (1994) is how to use the type-theoretical notion of contexts to represent possible worlds (or situations in the informal sense).[18] This idea has been taken up and further explored by many other researchers to study contextual representations including the work by Boldini (2000) in Martin-Löf's (1984) type theory, Ahn (2001) in pure type systems (Barendregt 1992) and their extensions, and Dapoigny and Barlatier (2010) in the Extended Calculus of Constructions (Luo 1990a, 1994).[19]

In the last decade, the study of MTT-semantics has made fruitful and significant developments, starting with the employment of coercive subtyping (Luo 2009c, 2012b) and leading to a full-blown semantic framework. The term Modern Type Theories (MTTs) was first used by the second author in Luo (2009c) which, on the one hand, distinguishes MTTs from simple type theory used in Montague semantics and, on the other hand, incorporates a wider class of (intensional) dependent type theories as foundational languages for MTT-semantics including, for example, the impredicative type theories such as UTT (Luo 1994).[20] The developments so far have made significant contributions that may be summarized in the following three respects, which are described in this book:

– *Fundamentals*. MTTs are shown to be adequate foundational semantic languages. In this respect, there are two noteworthy developments: *subtyping* (Luo 2009c, 2012b) and *signatures* (Luo 2014) for MTT-semantics. For the former, the employment of coercive subtyping (Luo 1999; Luo *et al*. 2012) has been proposed and shown to be a necessary and crucial mechanism for MTT-semantics.[21] For the latter, a novel notion of signatures has been studied and proposed for MTT-semantics – MTTs are extended with signatures which, in particular, may contain subtyping entries (Luo 2014; Lungu and Luo 2018), so that everyday situations (or incomplete worlds in the

---

adequate for dependent type theories and can be used to rectify this – see section 3.3.1 and footnote 21 on p68 for more on this topic.

18 In this book, the phrases such as *possible worlds* and *situations* are used informally and, in particular, they do not refer to any of their formal meanings that have been used in the literature.

19 The type-theoretical notion of context is not quite proper to model situations since abstraction can always be done over entries in a context. A further development has been made in Luo (2014) as well as Lungu and Luo (2018) to introduce the notion of signature, which can be used to represent situations in a more adequate way. In this book, we shall use type theory with signatures – for more details, see Chapter 2, especially section 2.1, section 2.4 and section 2.5.

20 One may consider the point of view (and we agree with it) that an impredicative type theory with an internal totality $Prop$ of logical propositions is better suited for formal semantics – see Luo (2019b) on proof irrelevance for one of the arguments for this.

21 In particular, it offers a satisfactory solution to Ranta's multiple categorization problem as mentioned above and footnote 17 on p13, and hence overcomes the major obstacle for MTTs to be applied to formal semantics – for more on this, see section 3.2.2.

informal sense in model-theoretic semantics) can be described adequately by means of this new contextual mechanism.[22]

– *Applications*. MTTs are shown to be powerful for semantic constructions. Using MTTs' rich type structure, various linguistic features have been studied and given semantics in MTTs including, for example, the study of various forms of modifications (Luo 2011a; Chatzikyriakidis and Luo 2013, 2017a) (see section 3.3 and Chapter 4). Two other noteworthy developments are as follows: the development of dot-types for copredication in MTT-semantics[23] (Luo 2009c, 2012b; Chatzikyriakidis and Luo 2018) (see Chapter 5 for more on copredication) and the use of coercive subtyping as a useful tool in various semantic constructions including, for example, sense disambiguation (Luo 2011b) (see section 3.2.2) and construction of dot-types (see Chapter 5), among many other examples as shown later in this book.[24]

– *Reasoning*. MTT-semantics is shown to be a sound basis for computer-assisted reasoning in natural language. MTTs are proof-theoretic: they are not only specified by proof-theoretic rules but also have proof-theoretic meaning theories (Luo 2014), which allow them to be implemented efficiently and used effectively for reasoning on computers. They are implemented by computer scientists in *proof assistants* and we have used the proof assistant Coq (Coq 2010) to implement MTT-semantics and performed various reasoning tasks and experiments based on such implementations (Luo 2011b; Chatzikyriakidis and Luo 2014, 2016) (see Chapter 6 for more details).

It is worth remarking that, when applied to formal semantics, whether a type theory is predicative or impredicative makes a difference. Although using Σ-types in a predicative type theory both as the existential quantifier and a structural mechanism solves the anaphora problem for simple donkey sentences (see footnote 16 on p13), such interpretations are inadequate when *counting* is involved (see, for example, those sentences involving "most" (Sundholm 1989; Tanaka 2015)).[25] As proposed

22 Formally, the type theories in this book are extended with signatures (Luo 2019a) – see Chapter 2.

23 For people familiar with type theory, it is worth mentioning that dot-types (Luo 2009c) are not ordinary inductive types, rather they are specially developed for dealing with copredication.

24 The idea of using coercive subtyping for formal semantics was first proposed by the second author in Luo and Callaghan (1998), but it was not until much later (a decade later) when Luo (2009c) was published where coercive subtyping was employed in semantic interpretations of modified CNs and constructing dot-types for copredication.

25 This problem with Σ playing a "double role" can be illustrated by considering (1.13), whose interpretation in Martin-Löf's type theory would be (1.14), where "most" is interpreted by means of the quantifier $Most$ defined in Sundholm (1989). (The second author thanks Justyna Grudziñska for a discussion about this example.)

(1.13) Most farmers who own a donkey beat it.

(1.14) $Most\ z : (\Sigma x{:}Farmer \Sigma y{:}Donkey.\ own(x,y)).\ beat(\pi_1(z), \pi_1(\pi_2(z)))$

and discussed in Luo (2012a, 2019b), one would solve this problem by imposing proof irrelevance (i.e. all proofs of a logical proposition are the same); however, such an imposition is only OK for type theories that distinguish logical propositions from other types (as in UTT), not when they are identified (as in Martin-Löf's type theory). With proof irrelevance in UTT, where both ordinary existential quantifier and Σ-types exist, a satisfactory semantics can be given to a sentence with both anaphora and counting.[26] Such a solution is also available for predicative type theories as well: for example, as proposed and studied in Luo (2019b), we can employ MLTT$_h$, an extension of Martin-Löf's type theory with the h-logic studied by Voevodsky in the HoTT project (HoTT 2013), to obtain an adequate foundational semantic language, which accommodates both strong and weak quantifiers in a similar fashion.

The study of MTT-semantics, especially its development in the last decade, is a part of a wider research endeavor by many researchers who have recognized the potential advantages of rich type structures in constructing formal semantics. For instance, besides work on MTT-semantics mentioned above, Retoré (2013) has employed the system F (Girard 1972; Reynolds 1974) to study how to use polymorphism in semantic constructions and Bekki (2014) has studied Dependent Type Semantics to discuss issues such as anaphoric expressions and underspecification in dependent type theories.[27] Besides these, there are also several

---

The interpretation (1.14) fails to respect correct counting because, with Σ, the proof objects of $own(x, y)$ contribute to counting in a wrong and unintended way.

26 As pointed out by the second author in Luo (2019b), in a type theory such as UTT where we can enforce proof irrelevance (see section 3.3.1 and footnote 22 on p. 69), the donkey sentence (1.13) in the above footnote 25 can be given a satisfactory interpretation (1.15) by means of Σ and the weak propositional quantifier $\exists$, where the propositional quantifier $Most$ is defined in UTT:

(1.15) $Most\ z : [\Sigma x{:}Farmer \exists y{:}Donkey.\ own(x, y)]$
$\forall y' : [\Sigma y{:}Donkey.own(\pi_1(z), y)].\ beat(\pi_1(z), \pi_1(y'))$

Note that $\exists$ and Σ are both traditional binding operators. Although we may not give an intended semantics to (1.13) using only either of them alone, it can be done if both are available. This implies that, in order to solve such a problem, we may not have to abandon traditional binding mechanisms to consider new mechanisms, a strategy adopted by proponents of dynamic semantics, such as Kamp (1981) and Groenendijk and Stokhof (1991). For example, dynamic predicate logic (Groenendijk and Stokhof 1991) is a non-standard logical system: among other things, it is non-monotonic and the notion of dynamic entailment fails to be reflexive or transitive – such a move seems to be paying too big a price that could have been avoided.

27 DTS has two versions as far as the underlying type theory is concerned: Martin-Löf's type theory in Bekki (2014) and some impredicative type system in Bekki and Mineshima (2017). In the DTS approach, CNs are interpreted as predicates as in the traditional Montagovian semantics and, because of this, we would not need the rich type structures and DTS mainly uses Π/Σ-types as logical operators. This means that it does not have the advantages in the CNs-as-types paradigm (see section 3.2.1), either.

pieces of very interesting work on linguistic semantics based on ideas of dependent typing, but not on formal type theories, including those by Asher (2012), Cooper (2005)[28] and Grudzińska and Zawadowski (2017), among others.

1.4.3. *Merits of MTT-semantics*

We contend that, as foundational semantic languages, MTTs are advantageous and MTT-semantics has unique merits, as compared with simple type theory and Montagovian semantics. Here, we outline some of its merits, organized below into two respects, which are further elaborated in the following chapters (and related papers). Please note that this is in no way to cover all, or even most, attractive aspects of MTT-semantics and it only serves the purpose of highlighting some notable features, hopefully making it easier to understand the following pages of the book.

Rich typing for semantic constructions. MTTs have a rich type structure: unlike simple type theory in which there are only the base types **e** and **t** and arrow types $A \to B$ (or $\langle A, B\rangle$, in a traditional notation), there are many ways to form types including, for example, dependent types (e.g. Π-types and Σ-types), inductive 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).[29] As we mentioned briefly earlier, the typing judgments of the form $a : A$, intuitively saying that a is an object of type A, are very different from the set-theoretical membership statements of the form $s \in S$: for example, the typing judgment $a : A$ is mechanically decidable, while the membership relation $s \in S$ (expressing that s satisfies predicate S) is a formula in the first-order logic and its truth is undecidable. Among other things, this decidability property is essential for any type theory to have a logic based on the propositions-as-types principle.

The rich typing disciplines also give us several advantageous points in semantic constructions. A simple advantage is that rich typing allows us to employ typing judgments, rather than membership statements, to deal with selectional restriction. In MTT-semantics, we can use the typing judgment $j : Man$ to express that "John is a man" and this is different from Montague semantics where we use the formula $man(j)$ to represent the meaning. Therefore, typing also offers us a means to deal with selectional restriction – to exclude meaningless sentences or phrases. This

28 It may be worth clarifying that the system called Type Theory with Records (TTR) (Cooper 2005, 2017) is not a type theory, as the term is usually understood, but rather a set-theoretic notational framework, where $a : T$ if, and only if, a is a member of the set whose name is T in set theory. Although it is set-theoretical, the development of TTR, especially in the early days, was influenced by the study of record types by Tasistro (1997) and Betarte (1998) in Martin-Löf's logical framework.

29 See Chapter 2, and sections 2.2 and 2.3 in particular, for more details.

allows us to use typability as a formal criterion to judge whether a sentence is meaningful: for example, we would usually think (in a non-fictional world) that a sentence like "Tables talk" is meaningless with a category error – this is captured by means of typing in MTT-semantics: the MTT-interpretation of "Tables talk" is ill-typed (or an illegal expression). This is different from Montague semantics where the interpretation of "Tables talk" is just a false statement but it is a well-formed formula. (See section 3.1 for a further explanation.)

The rich type structure in MTTs enables us to use types to play the role of representing various collections,[30] the role played by sets in the Montagovian set-theoretical semantics. Besides acting as logical propositions, types can also be used in various ways in semantic constructions. For example, this allows one to interpret common nouns as types rather than predicates (see the above, (Luo 2012a) and section 3.2.1); in section 3.3, we show how various adjectival modifications can be modeled by means of rich typing, with Σ-types for intersective modification, Π-types (and the associated polymorphism) for subsective modification, disjoint union types and Π-polymorphism for privative modification, and a logical modal collection operator for non-committal modification. We shall also show in Chapters 4 and 5 that the typing structure can be used to model more advanced linguistic features such as gradable adjectival modification and copredication.

As mentioned earlier, subtyping is not only essential for MTT-semantics, but also very useful in semantic constructions. For instance, coercive subtyping is employed in defining dot-types (Luo 2009c, 2012b) for giving adequate interpretations of sentences involving copredication, an example of which is the sentence in (1.16), where a delicious lunch refers to food and a lunch that took forever refers to a process. Such copredication phenomena have been studied by many researchers such as Pustejovsky (1995) and Asher (2012). For MTT-semantics, the second author has proposed to use dot-types to deal with copredication (Luo 2009c): for this example, lunch involves two aspects: that of being food and that being a process. This is formalized by means of the subtyping relationship (1.17). It is then straightforward to see that the formula in (1.18), which interprets (1.16), is well-typed because of the subtyping relation in (1.17), where $l : Lunch$ is the interpretation of "the lunch".

(1.16) The lunch was delicious but took forever.

(1.17) $Lunch \leq Food \bullet Process$

(1.18) $delicious(l) \wedge take\_forever(l)$

When defining dot-types such as $Food \bullet Process$, coercive subtyping is used: we have, for example, $Food \bullet Process \leq Food$ and $Food \bullet Process \leq Process$ and therefore (1.18) is well-typed: both predicates $delicious$ and $take\_forever$ can be

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

applied to $l : Lunch$ because both domains $Food$ and $Process$ are supertypes of $Lunch$. (For more formal details on subtyping, see section 2.4 and section 3.2.2, and for more about copredication and dot-types, see Chapter 5.)

MTT-semantics is both model-theoretic and proof-theoretic. MTT-semantics has a unique important feature: it is *both* model-theoretic and proof-theoretic, as argued for in Luo (2014, 2019a), and this has made MTTs a particularly suitable 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.[31] Usually, an MTT is presented as a natural deduction system where, in particular, every type constructor is specified by inference rules among which introduction rules are to declare how the type (formula) is inhabited (proved) and the elimination rules to specify what consequences can be derived under the assumption that the type (formula) is inhabited (proved). These rules are harmonious and, as an important consequence, MTTs themselves have proof-theoretic semantics, as studied by logicians such as Gentzen (1935), Prawitz (1974, 1973) and Martin-Löf (1984, 1996) and discussed by philosophers such as Dummett (1975, 1991) and Brandom (1994, 2000), among others.[32] For example, Martin-Löf has studied and developed meaning theory and given a solid foundation for his type theory (Martin-Löf 1984). Therefore, MTT-semantics, the formal semantics in MTTs, is proof-theoretic in the sense that its foundational semantic languages have proof-theoretic meaning theory based on inferential uses.[33]

---

31 For details, see Chapter 2 and related references for MTTs.

32 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. For example, to have a proof-theoretic semantics, the introduction and elimination rules for any logical operator or type constructor must be in harmony, as discussed by Dummett (1991) and others.

33 Note that MTT-semantics itself is still denotational (and model-theoretic – see below) in the sense that the meaning of a natural language phrase/sentence is given by its denotation in an MTT, not directly by proof rules, although its foundational languages have a proof-theoretic meaning theory, as described above in the sense of "logical inferentialism" or "proof-theoretic semantics" (Kahle and Schroeder-Heister 2006). Can we extend logical inferentialism to natural language? For example, Francez and his colleagues (Francez 2015) have studied natural language semantics in a way that directly adopts the methodology of proof-theoretic semantics for logical systems and applies it to natural language. However, in order for such semantics to be adequate or viable for natural language, one would have to answer some difficult questions. For example, for a logical language, logicians have recognized that, to capture the meaning of a formula in the use theory, it is central for us to spell out two aspects of its use

Furthermore, the fact that MTTs are proof systems with proof-theoretic semantics has another significant and practical consequence in natural language reasoning based on MTT-semantics. In particular, this makes it possible for MTTs to be implemented in proof assistants such as Agda (Agda 2008), Coq (Coq 2010) and Lego/Plastic (Luo and Pollack 1992; Callaghan and Luo 2001) – computer-assisted reasoning systems that computer scientists have developed and successfully used for the formalization 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 (Chatzikyriakidis and Luo 2014; Luo 2011b) and Plastic (Xue and Luo 2012) and used for natural language reasoning (Chatzikyriakidis and Luo 2016) (see Chapter 6 for more details).

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.[34] This is mistaken – MTT-semantics is also model-theoretic. 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 model-theoretic semantics to natural language. That is, in MTT-semantics, an MTT plays the role of meaning-carrying language – in the traditional model-theoretic semantics, this is the role played by set theory. In

– how to prove it and how to derive consequences from it, as captured by introduction and elimination rules, respectively. Could this be generalized directly to natural language? This would be a big leap and, unfortunately, a convincing justification has not been forthcoming. In fact, philosophers such as Brandom who advocate "general inferentialism" do not restrict the rule patterns in this way as the following remark by Peregrin (2013) explains:

> ... It is important to realize that this logical kind of inferentialism must be classified as a special case of general inferentialism not just because it is restricted to logical constants, but also because strict constraints are posed on the inferential patterns that can constitute the (meanings of the) logical constants. By contrast, general inferentialism only claims that the meaning of a word is its role vis-à-vis an inferential pattern; there is no claim that each word must have its own constitutive inferential pattern, let alone a claim that this pattern must be of a shape prescribed by Gentzen.

In other words, the meaning of an NL phrase/sentence may not be captured by only introduction and elimination rules and there is a good reason for us to be pessimistic for the viability of such an approach.

34 An exception is (Ranta 2015, p. 346) where Ranta 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.

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 semantics, types in MTTs are employed to represent collections 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 semantics.

As mentioned above, MTTs have a rich type structure and, as to be demonstrated in Chapters 3, 4 and 5, in MTT-semantics these types play an important role in representing different kinds of collections such as those given by CNs modified by various kinds of adjectives. Such representations by means of types are not available for simple type theory, which has only base types and arrow types. Types in MTTs are powerful and can provide various different representations in semantic constructions. This is the first facet to explicate that MTT-semantics is model-theoretic.

The second facet is that the contextual structures in MTTs, which are not available in traditional logical systems, provide very useful means in semantic constructions and, in particular, they offer effective support for model-theoretic descriptions of incomplete possible worlds. We shall describe the notion of *signatures* (Luo 2014) (see section 2.1) that allow MTTs to be more suitable to support model-theoretic descriptions.

The notion of signature in type theory, as far as we know, first appeared in Edinburgh Logical Framework (Harper *et al*. 1993), where signatures with membership entries are used to describe a logical system.[35] We shall introduce signatures that do not just have the usual membership entries of the form $x : A$ but also contain two new forms of entries, subtyping entries and manifest entries (see section 2.4), which strengthen the power of signatures in representing (incomplete) possible worlds, even in cases where, for example, situations are infinite or involve more sophisticated phenomena. As shown in Lungu and Luo (2018) and section 2.5, 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 normalization (and hence logical consistency).

35 Historically, signatures have been used in describing algebraic structures and, for example, more recently they were used in describing many-sorted structures in the study of algebraic specifications (Goguen and Burstall 1983). 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.

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 recognized that there is such a possibility (in particular, set theory is not such a language since it is not proof-theoretic). MTT-semantics is both model-theoretic and proof-theoretic and sheds a new light on the division between model-theoretic semantics and proof-theoretic semantics and allows us to study formal semantics from a new perspective.