Parameterized monads in linguistics

Viet Ha Bui

A thesis submitted in partial fulfilment of the requirements of the University of Wolverhampton for the degree of Doctor of Philosophy

2021

This work or any part thereof has not previously been presented in any form to the University or to any other body whether for the purposes of assessment, publication or for any other purpose (unless otherwise indicated). Save for any express acknowledgements, references and/or bibliographies cited in the work, I confirm that the intellectual content of the work is the result of my own efforts and of no other person.

Signature: . . . .

Date: . . . . 17/06/2021 . . . .

Abstract

This dissertation follows the formal semantics approach to linguistics. It applies recent developments in computing theories to study theoretical linguistics in the area of the interaction between semantics and pragmatics and analyzes several natural language phenomena by parsing them in these theories. Specifically, this dissertation uses parameterized monads, a particular theoretical framework in category theory, as a dynamic semantic framework to reinterpret the compositional Discourse Representation Theory(cDRT), and to provide an analysis of donkey anaphora. Parameterized monads are also used in this dissertation to interpret information states as lists of presuppositions, and as dot types. Alternative interpretations for demonstratives and imperatives are produced, and the conventional implicature phenomenon in linguistics substantiated, using the framework. Interpreting donkey anaphora shows that parameterized monads is able to handle the sentential dependency. Therefore, this framework shows an expressive power equal to that of related frameworks such as the typed logical grammar and the dynamic predicate logic. Interpreting imperatives via parameterized monads also provides a compositional dynamic semantic analysis which is one of the main approaches to analysing imperatives.

Acknowledgements

I would like to thank my supervisors, Dr Ha Le An and Prof. Ruslan Mitkov, for welcoming me into their computational research group. I also would like to thank the examiners, Dr Chris Fox, Prof. Arm Sabry, and Prof. Fabrice Laussy, for the feedback in their report.

I would like to thank my friends: Sarah Girling, Dr Michael Oak, Dr Craig Williams from my time at Wolverhampton University. I also would like to thank Dr Peter Dybjer who introduced me to type theories and his logic programming group at Chalmers, and Dr Wouter Swiestra who introduced me to Hoare's state monad. Without them, the thesis could not have been completed. I also thank Dr Zhaohui Luo who introduced me to linguistics during my time at Royal Holloway, the University of London. Also worthy of mention here are Dr Robin Adam and Dr James Cheney.

Finally, I would like to thank my family and Linh Le for supporting me during the time when this dissertation was written. I dedicate this dissertation to my father.

Contents

Chapter 1

Introduction

This research is an attempt to provide a theory of meaning to natural languages. According to [1] [p. 22] and [2][p. 2] the theory of meaning, also called the theory of knowledge, focuses on the question of how linguistic meaning is constructed. It differs from meaning theory which answers the question of how to specify the meanings of words and expressions. For example, meaning theory builds a dictionary, while the theory of meaning examines how words are combined to form ideas.

In order to achieve the theory, I chose category theory, which is studied in both computing and mathematics fields. Hence, this dissertation belongs to computational linguistics, in that it applies computing theories to linguistics. Since it focuses on theoretical aspects, it also belongs to formal semantics.

The λ calculus by [3] provides a major framework for linguistic semantic analysis through Montagovian semantics in [4], as discussed in [5]. However, semantic approaches to linguistics suffer from a general problem: there are linguistic phenomena that cannot be analyzed by semantics alone. For example, since the pioneering research by [4], key phenomena such as quantifications, still lie outside the proper grammatical treatment of formal semantics.

[6] shows that quantifications in linguistics can be regarded analogically to side effects in functional programming languages' terminology, such as the terms *shift* and *reset* in composable continuation by [7] in programming languages. On the other hand, the side effects can be captured by the mathematical monads in category theory by [8]. However, these two notions of composable continuation and monads are related by [9]. Thus, this dissertation has redefined the notion of *side effects* in order to reduce the misunderstanding of this jargon in linguistics, and associate it with *contextually related phenomena* in chapter 4. Additionally, it includes the definition of effects by [10] as an interaction between semantics and pragmatics.

Indeed, I extend the research done by [11] and replace Shan's thesis by proposing monads as an underlying framework of interactions between semantics and pragmatics, in place of his continuation approach with recent research by [12]. Shan's in [6] proposed delimited continuation as a framework for side effects because he supposed that it had more expressive power than monads. However, a recent study by [13] rejected this idea and stated that it is still a conjecture under the current accepted research results.

Thus, this dissertation improves Montague's semantics by extending the theoretical foundation of the semantics to modularize and unify additional linguistic phenomena under monads in category theory. Monads do not change the nature of linguistic phenomena, nor do they provide any substance analysis of the phenomena. Instead, they unify related phenomena under a general structure, and we can use them to reason under the compositional principle. In this research, I use monadic expressions to represent types, and we use them to express the denotational semantics of a linguistic term, while λ expressions are used for the term's operational semantics.

Hence, a linguistic term is given both denotational and operational expressions. Through the use of types, the monadic expressions are guarded by a compositional rule under the typing principle. On the other hand, the operational expression in the λ calculus formation is more deliberate so as to express the meaning of a term, following the lead of [14] who also promoted types for restricted conditions of the λ calculus. Hence, we are extending their research by pointing out a clearer type system that uses modern type theories which will be discussed below in sections 2.1.2, 2.1.3 and 5.5.

Monads have an expressive power rich enough to capture other semantics such as dynamic semantics as per [15] and [16], and situation semantics as developed by [12] and [17][p. 6]. Furthermore, using the evaluation order discussed in [6], monads have an advantage over the type logical grammar approach by [18] in the flexible treatment of quantification scopes.

Hence, in the author's opinion, studying monadic applications in linguistics does not complete the extension of monadic applications. Rather, it is the beginning of a research trend in generalizing monads for linguistic applications in the interface between linguistic semantics and pragmatics. For example, [19], also develops a related idea.

1.1 Overview

Monads have been introduced to linguistics as a theoretical framework by [12, 11, 20, 15, 16, 21, 17, 22]. This dissertation investigates parameterized monads, a generalization of monads by [23], to capture dynamic semantics through the reinterpretation of the compositional discourse representation theory (cDRT) by [24] and the first order logic interpretation of natural languages by [25]. It also parses additional linguistic phenomena: the presuppositions and *ist* notion, dot types, the donkey sentence, the imperatives, the definite descriptions and demonstratives, and the conventional implicature using parameterized monads. Furthermore, it strengthens the expressive power of monads to be compatible with other theoretical frameworks such as the type-theoretical semantics of [26], the dynamic predicate logic of [27], the delimited continuation of [12], or the typed predicate logic of [28].

The thesis structure is organized as follows. The introduction and conclusion chapters establish the contribution to knowledge as well as limitations and the future research. Chapters 2,3, and 4 are the literature review chapters. Chapter 5 provides a brief introduction to parameterized monads in mathematics and computing. Finally, chapters 6,7,8, and 9 illustrate the formalization of the above linguistic phenomena in parameterized monads.

Chapter 2 provides the background knowledge of parsing in computing with the parsing as deduction hypothesis. It also introduces two notorious theoretical frameworks, in parallel with category theory, in computing: the λ calculus and type theory, and their applications in linguistics. The latter part of chapter 2 introduces the notion of ambiguity in linguistics under the theoretical computing perspective. Hence, it points out the characterization of the ambiguity notion in linguistics into the essential and spurious one. In addition, the chapter provides a brief introduction to the following linguistic phenomena: generalized quantifiers, scope taking, and conventional implicatures.

Chapters 3 and 4 introduce the notion of monads in mathematics [29, 30, 31] and its applications through the continuation, dynamic semantics, and the conventional implicatures [12, 17, 15, 21]. Notably, Chapter 3 introduces the interpretation of the λ calculus in category theory. This interpretation eliminates the criticism of the research by [32]; the criticism stated that their research did not cover the Montagovian grammar. Chapter 4 discusses further applications of monads in linguistics.

The limitation of the expressive power of monads has been noticed by [33]. Thus, chapter 5 introduces the definition of parameterized monads, a well known extension of monads, by [23, 34, 35] with a further extension by [36]. There are various extensions of monads which include the applicative functor [37]. The parameterized monads has less expressive power than the applicative functor but it holds the compositional principle. The chapter begins by introducing the mathematical definition of the parameterized monads. It then introduces their examples in computing which include the composable continuation by [9] and the IO monads by [38]. In addition, the notion of the specification and a type system for parameterized monads, which build upon the research by [39] and [40, 34], respectively, are introduced.

Chapters 6,7,8, and 9 introduce their applications in parsing instances of several linguistic phenomena. Chapters 6 builds upon the first order logic interpretation of natural languages in the programming language Haskell, a computing implementation of monads and category theory, by [25]. In addition, it investigates the interpretation of the dot types and the *ist* notion in parameterized monads. On the other hand, chapter 7 provides the dynamic semantic interpretation of parameterized monads by reinterpret the cDRT and the parsing process of the donkey sentence.

This dissertation categorizes the parameterized monads as a moderate contextualization formalism, according to [41], to parse natural languages. It is more expressive than the logical (static) approach to linguistics and less generic than the radical one. Furthermore, the author characterizes the parameterized monadic interpretation of the donkey anaphora as a dynamic semantic approach to the phenomenon. The formalization of the donkey sentence is straightforward: it uses a comprehensive analysis of the phenomenon by [42], with an extension to variable-binding as discussed by [43, 44].

In an argument for favouring the static approach to the phenomenon, [42] poses three problems for the dynamic approach, namely: disjunction, undistinguished participants, and neontological pronouns. However, [12] solves the first two problems. Hence, in the author's opinion, the gap in the interpretations of the phenomenon between the dynamic and static approaches is reduced.

However, the treatment of variable-binding in [12, 45, 17, 46] is not straightforward and does not provide a natural semantic interpretation of the phenomenon. They use delimited continuation and continuation monads or type lifting techniques to interpret and reason about the scope taking of the phenomenon in their theoretical frameworks. In order to achieve their objectives, they used the double negation law. Hence, the formalized sentence appears in a negative rather than an affirmative formation. According to [45, 12] for example, the sentence

if a farmer owns a donkey, he beats it.

is parsed (or formalized) as

$$\neg\exists x.\mathbf{farmer}(x) \wedge \exists y.\mathbf{donkey}(y) \wedge \mathbf{own}(y,x) \wedge \neg(\mathbf{beat}(y,x))$$

Intuitively, this semantic interpretation of the above sentence is not natural as a result of using the double negation law. Contrastingly, parameterized monads provide a clear solution. In order to formalize the phenomenon, the author divides the formalization process into two steps. First, we define the dynamic implication in dynamic semantics in section 7.3.1 to faithfully parse the sentence into

$$\exists x.(\mathbf{farmer}(x) \wedge \exists y.\mathbf{donkey}(y) \wedge \mathbf{own}(y,x)) \Rightarrow (\mathbf{beat}(y,x))$$

The state is omitted in the above representation and serves as a context in dynamic semantics. A detailed discussion can be found in chapter 7. Intuitively, it is also analogous to a situation in situation semantics by [47]. Second, we change the scope-binding of the pronoun variable by using a technique similar to that in [43] or [44], namely Egli's theorem. In our framework, this technique is called the *swapping technique* in section 7.4.2.

The idea behind the swapping technique is that the variable in an existence operator is free to change and take scope. We use the Brouwer–Heyting–Kolmogorov (BHK) interpretation to provide support for this idea. Intuitively, the BHK interpretation means that we can either interpret a term in a formula by the direct logical interpretation, or shift it into the context. If we shift the term into the context, then we can reuse it in the next formulae in any order. Thus, the formula is rewritten as

$$\exists x.\exists y.(\mathbf{farmer}(x) \wedge \mathbf{donkey}(y) \wedge \mathbf{own}(y, x) \Rightarrow \mathbf{beat}(y, x))$$

This formalization process differs from traditional logical interpretations of natural languages by having multiple stages of semantic analysis. Specifically, we add an additional scope-binding process to the logical parsing process. Adding this extra process results in the proper placement of the $\exists$ operators in the sentence. Besides, this dynamic semantic formalization has an equivalent interpretation in a logical formation by the previous research of [24, 42] or situation semantics by [42, 48].

If we interpret an indefinite as an existence in traditional logics, for example in [15], then this proposal provides an alternative mechanism for exceptional scopes taken by indefinites, in comparison with [17]. The purpose of scope-binding is to have a correct assignment of logical variables in the interpretation of natural languages. Hence, if we hypothesis an anaphora as a variable, then scope-binding is also a mechanism of anaphora resolution by allocating antecedents to a scope of suitable referred objects.

Chapter 8 studies the imperative phenomenon in parameterized monads, building on the previous research by [49, 50]. [49] provides an axiomatizing system of an imperatives logic which is based on Hoare logic by [51]. Moreover, [52] provides a solution to interpreting Hoare's logic in monads. In addition, the parameterized monads are also regarded as a Hoare's logic extension of monads by [53]. Therefore, the parameterized monads provides a dynamic interpretation, see [54], as well as establishing a new logic to the phenomenon. Related research includes [55]. Our solution, however, is a compositional treatment of the phenomenon by interpreting the cDRT in parameterized monads while [55] using classical discourse representation theory (DRT) by [56] only.

Finally, Chapter 9 uses the parameterized $\mathbb{IO}$ monads to parse the demonstratives by using the Wolter's hypothesis in [57]. Hence, it provides a theoretical framework for the research by [58]. In addition, it also uses the session type in parameterized monads to substantiate the interpretation of conventional implicature, from [20] in section 9.3. Session types can capture the data or information exchanged between the client and the server dimensions, while their solution as a writer monad cannot. The separation of the client and server dimensions in session types is analogous to the same aspect between the at-issue and conventional implicature dimensions.

1.2 Related research

Related research is discussed in both the overview and in the conclusion chapter. The key related research is as follows.

- Both [16] and this dissertation use the recent extensions of monads. Whereas Grove uses graded monads, I use parameterized monads. Both are shown to be equivalent by [53]. However, Grove based her framework on the possible-world semantics of [59], while we use the type theory discussed by [60]. Intuitively, it means that our approach is proof-oriented while hers is model theoretic-oriented. Finally, a recent research by [21] also introduces monads into linguistics. In contrast to them, I use a different theoretical background, and I interpret different linguistic phenomena.

- [17] also uses a monad-based framework, namely monad transformers ([61]) that provide a dynamic interpretation of the scope of indefinites in a sentence. In contrast to his framework, I use parameterized monads which provide a clearer solution to the donkey anaphora phenomenon. In addition, the advantages of using parameterized monads over monad transformers is that the former provide a compositional and solid mathematical theoretical framework in contrast to the latter.

In this dissertation, the compositional principle, in the sense of [62] and [5], is a major concern and is expressed as a type which is represented as a state in parameterized monads, driven in the sense of [14][p. 44] and [12][p. 25].

Thus, while both parameterized monads and monad transformers raise concerns about combining monads, parameterized monads focuses on explicit type or state declarations, i.e. the compositional aspect, while monad transformer research governs on the operation or λ abstraction aspect. Intuitively, the parameterized monad approach is semantic-oriented, while the monad transformers one is syntactic-oriented.

- [15, 63] interpret dynamic semantics in monads. I develop their research further by using parameterized monads, enabling capture of the donkey phenomenon.

- [10] provides algebraic effects and handler techniques to analyse side effects in natural languages. Our research is similar to his, in being based on type theories. The difference is that we are using monads and parameterized monads to illustrate side effects rather than the algebraic effects and handlers as developed in [64]. Furthermore, instead of using handlers to manage the scope of the effects, I propose to use our own *swapping technique*.

- [55] provide a dynamic approach to the imperative phenomenon. Their research is based on DRT, and I develop their research further by providing a compositional approach by interpreting the cDRT using parameterized monads.

- [65] reinterprets the cDRT by [24] and extends the framework to the PCDRT in order to capture the plurality phenomenon in linguistics. This dissertation also reinterprets the cDRT in chapter 7 with an alternative theoretical foundation in category theory.

1.3 Contribution to knowledge

Category theory has been used by previous researchers such as [66] with categorial grammar and [67]. [30][p. 2] recently advocated that category theory could be an alternative to set theory in mathematics and the sciences. Responding to criticism of set theory in linguistics including those by [68, 69], this dissertation takes a step towards proposing category theory as an alternative to set theory in linguistic semantics.

This dissertation examines a particular class of category theory, namely, monads by [8], and their extension to parameterized monads by [23], to formalize effects in several linguistic phenomena. Previous studies of monads in linguistics by [11, 15, 21, 70, 16, 17, 22] indicated that monads provide a proper framework for capturing linguistic effects. The effects are interpreted as pragmatics related to semantics phenomena by [10], or as anaphora resolutions by [17]. This dissertation redefined the meaning of effects in linguistics as context-related phenomena, as discussed by [71, 41, 72, 73].

[6, 12] based their theoretical foundation on a continuation approach by delimited control by [7, 74], and criticised the work of [8] as being less concrete, such as in [75][p. 142] and [6][p. 91]. However, the recent research of [13] shows that this criticism is incorrect. Hence, this dissertation contributes to current knowledge by using parameterized monads to provide an alternative foundation to composable continuation for interpreting effects in linguistics. This is a strong foundation on denotational semantics and proves the properties discussed in [50, 52, 33].

Monads cannot capture all of the effects in computing (an observation made by [33]). Consequently, this dissertation does not claim to capture all of the phenomena in the interaction between linguistic semantics and pragmatics; the research goal here is to observe and parse certain linguistic phenomena in monads and parameterized monads. Hence, this dissertation contributes to current knowledge by parsing the presuppositions, *ist* notion, and the dot types in parameterized monads in chapter 6. In addition, it also parsed the definite descriptions and imperatives in parameterized $\mathbb{IO}$ monads, related to [38], in sections 9.1 and 9.2.

In addition, it was observed that the pioneering research on applying the state monads in linguistics by [15] did not interpret the donkey sentence phenomenon due to the limit of the expressive power of monads. The interpretation has been studied recently by using extensions of monads by [17, 16] [1]. Parameterized monads, in conjunction with these extensions, are expressive enough to parse the donkey sentence. Indeed, the author used the direct interpretation of the dynamic condition in section 7.3.1 to interpret the condition *if* rather than using the double negation interpretation by [12]. This formalization implies that the parameterized monads achieve an expressive power equal to those of other successful theoretical frameworks such as the type-theoretical grammar by [26], the dynamic predicate logic (DPL) by [27], or the typed predicate logic by [28].

I also interpret the cDRT in parameterized monads to combine both the dynamic semantics and the Montagovian grammar found in chapter 7.[2] This interpretation uses Hoare-style logic rather than Dijkstra's weakest precondition calculus in the cDRT. Hence, this dissertation also contributes to current knowledge by reinterpreting the logic of imperatives, based on the research of [49]. This interpretation implies that we provide a compositional dynamic semantics to the phenomenon.

[1] intuitively, Charlow's approach is more operational approach oriented while ours is denotational one. Furthermore, this research is conducted in a parallel and separately with another extension by [16]

[2] The recent research discussion on the topic of combining dynamic semantics and Montague's semantics is at the end of section 4 in https://plato.stanford.edu/entries/dynamic-semantics/

According to [76], two major theoretical frameworks for interpreting the imperative phenomenon in linguistics are the dynamic semantics and modality by [55] and [77], respectively. However, the dynamic framework by [55] does not provide a compositional approach because their research is based on DRT by [56] rather than the cDRT.

This dissertation contributes to knowledge by providing a model that captures interaction between the at-issues and conventional implicature dimensions in the conventional implicature phenomenon in [78]. Specifically, the author interprets the conventional implicature as session types using parameterized monads in section 9.3. In previous studies of this phenomenon, [20] interpreted it as a writer monad with a principle of abandoning the interaction between the two dimensions in accordance with the research by [79]. However, an empirical study by [80] shows that there are linguistic phenomena that require interaction between the two dimensions.

Finally, [81] proposes a new analogy between the normalization of proofs and the evaluation of programs via the Curry–Howard correspondence. Through the use of our analysis of the *swapping technique*, this dissertation also proposes a new analogy between the scope evaluation problem in linguistics and the proof search problem in logics. This new analogy is based on an observation from previous research by [82] and [16], which states that the presupposition projection phenomenon is the proof search and scope evaluation phenomenon[3], respectively.

1.4 Thesis structure

This thesis is divided into the following ten chapters: introduction, parsed natural languages, introduction to monads, introduction of monads in linguistics, introduction to parameterized monads, linguistic structures in parameterized monads, the cDRT in parameterized monads, the imperative phenomenon in parameterized monads, others linguistic phenomena in parameterized monads, and the conclusion chapter.

Chapters 2,3, and 4 are the literature review ones. Firstly, chapter 2, the parsed natural languages chapter, consists of the following sections:

- Introduction to parsing as deduction hypothesis and λ calculus in linguistics.
- Introduction to type-theoretical semantics.
- Introduction to ambiguity in natural languages and associated linguistic phenomena.

Chapter 3, the basic theoretical or the introduction to monads chapter, covers:

- The duality between category and type theories

[3] A further discussion can be seen in by [83]

- The category theory background
- Major monads in computing.

Chapter 4 surveys monads in linguistics, and is organized as follows:

- Introduction to continuation in linguistics with the scope-taking phenomenon.
- Introduction to the state monad with dynamic semantics
- Introduction to the writer monad with the conventional implicature

Chapter 5, the parameterized monad chapter, introduces the recent extension of the monadic theoretical framework. It is structured as follows:

- Definition of parameterized monads
- Applications of parameterized monads in computing
- Specification structures of parameterized monads
- Introduction to the typed system for parameterized monads by the typed command calculus

Finally, chapters 6,7,8, and 9 applied the framework of parameterized monads to linguistics. Firstly, chapter 6 interprets linguistic structures in parameterized monads and it has following sections:

- Example of the first-order logic in category theory

- Berg's criteria for states
- Information states as presuppositions
- Lexical semantics of information states

Chapter 7, the cDRT in parameterized monads chapter, is organized as follows:

- Introduction to cDRT
- Interpretation of cDRT in parameterized monads with Hoare's logic
- Dynamic semantic definitions in parameterized monads
- Introduction to the parsed donkey sentence with the swapping technique and the analogy between proof-search and scope-taking

Chapter 8, the imperative phenomenon in parameterized monads one, is designed as follows:

- Literature review of the phenomenon with the revised Dubislav analogy
- Interpretation of the phenomenon in parameterized monads
- The imperative logic

Chapter 9, the additional phenomena chapter, parses the following additional linguistic phenomena in parameterized monads:

- Definite description

- Demonstrative
- Conventional implicature

Finally, the conclusion chapter summarizes the research, points out its limitations, and develops prospects for future research.

Chapter 2

Parsed natural languages

This chapter provides an overview and background knowledge for the dissertation. It focuses on the theoretical aspects of programming languages with applications of parsing in theoretical linguistics rather than discussing the whole area of computational linguistics. It uses logic as a methodological study, rather than statistics. The connection between logics and linguistics has long been recognized. For example, Bar-Hillel and Chomsky, quoted by [84][p. 2], show the striking relation between two fields:

> I think it is correct to say that the difference between the structural linguist and the formal logician is one of stress and degree rather than of kind.
>
> —*Bar-Hillel*

and

> The correct way to use the insights and techniques of logics is in formulating a general theory of linguistic structures.
>
> —*Chomsky*

2.1 An introduction to parsing natural languages

According to [60][p. 11] and [85][p. 61], the definition of *parsing* in compilers or programming languages is: the process of converting a language string into an formal, internal tree representation. Parsing is an essential part of a modern compiler. Parsing leads, in conjunction with other processes such as name resolutions, type checking, optimizations and code generations, to the abstract syntax tree.

Thus, the term *parsing* expresses the practical perspective on formalization, in the sense of [86]. According to [86], a formalism is a translated form of natural languages, rendered into a formal system such as logics or mathematical theory; an act of parsing natural languages to the theoretical expression is called formalization.

Taking another viewpoint, [87] provide a multi-disciplinary perspective on parsing natural languages. [87][p. 1] defined the term, 'parsing' etymologically, from classical origins:

> Like so many aspects of modern intellectual frameworks, the idea of parsing has its roots in the classical tradition; *(grammatical) analysis* is the Greek-derived term, *parsing* (from *pars orationis* 'part of speech') the Latin-derived one. In this tradition, which extends through medieval to modern times

In this dissertation, *parsing* follows the formal semantics tradition, i.e.

formalizing natural languages into computing-oriented theories or systems. From a theoretical perspective, a grammar for a parsed tree is an instance of a formal framework. In computing, the formal framework can be a regular language as in [85], or a more theoretical oriented as a type system, as in [88].

Therefore, in the author's opinion, the choice of formal semantic parsing provides a deeper analysis by bringing insights from computing theories to the study of linguistic phenomena. Furthermore, it provides new opportunities to test computing theories for its claimed strengths and weaknesses. In this regard, parsing natural language is an area of science where theories and empirical observation meet. In the author's opinion, this idea is similar to one in [78][p. 3], which also proposes to regard descriptive observations in a different way from theoretical proposals.

A related idea can be found in [89], which justifies the role of category theory in computer science. He claims that, in computer science, just as in physics, theories have to be tested. However, since category theory cannot be tested directly, we can take an alternative path by connecting computer science concepts to category-theoretic ones in order to determine what advantages are obtained by going through category theory.

If we take abstract computing theories such as category theory and shift their application from computer science to linguistics, the above idea is still valid. From the physicist viewpoint, the parsed languages, or formal languages, provide an abstract structure for studying linguistic phenomena.

They simplify and guide further theoretical or empirical research. It does not substitute for detailed research, however, where more subtle insights may be obtained.

In words, parsed natural languages, in the author's opinion, is the practical logics, i.e.

$$\boxed{\textit{parsed natural languages} = \textit{logical interpretation} + \textit{practical insight}}$$

Table 2.1: general picture.

This idea is similar to the idea of parsing as logical deduction in [25][p. 260]. It can be traced further to the paper by Pereira and Warren in 1983, or to a recent revision in [90]. The computational perspective on parsing is referred to by [85]. However, this dissertation focuses on its applications in linguistics, which is also presented in Chapter 9 of [25].

The generalized linguistic picture of the theoretical model of logical interpretation is treated in [46, chapter 1]. It can be further traced to previous research on formalizing natural languages in [91, 92, 62, 93, 94]. If we take a mathematician's view and follow Russell's thesis [92] that mathematics is linguistics, then deriving and analyzing an abstract linguistic structure is equivalent to proving a mathematical problem. Up to the author's knowledge, this idea still underlies recent research by contemporary semanticists.

In the author's opinion, a thorough linguistic analysis could lead to a better analysis of mathematical text as a domain specific language. Indeed, this direction has been investigated in [95, 94]. The first of these works provides a contemporary theoretical framework, including the dynamic semantics from [56], and a foundational notion of types with which to parse the language of mathematics.

However, the fact that there is no adequate system for formalized mathematics, nor for natural languages, has been widely accepted since Aristotle. Similarly, [60][p. 5] also cited Sapir's observation (1921) that all grammars leak. Thus, this dissertation stresses the practical insight aspect of parsed natural languages. The insight can be gleaned from the implementation and knowledge yields due to the implementation, such as the creation of new theory or interpretation. Research such as [6, 96, 12] shows how computing techniques, i.e. practical insight, such as continuation can provide further insight into the scope-taking phenomenon in linguistics in the 21$^{\text{st}}$ century. However, in the author's opinion, we should take care to seek a partial solution equipped for practical investigation rather than finding a total solution[1] for a phenomenon from this perspective. The practical investigation can alternatively take place via a statistical perspective which would, however, be outside the scope of this dissertation.

This direction is persuasive if we consider linguistics as a science of studying human natural communication, rather than of explaining the communi-

[1] A total solution can be criticized as providing only toy languages, as per [68]. However, we should consider its objects and associated research in other fields such as mathematics.

cated objects. As an interpersonal communication, a linguistic expression has both subject and object properties. It is objective since the expression should be understandable by others, and it is subjective because it includes the speaker's thought and information. The objective properties, in the author opinion, for semanticists to study, while the subjective properties are for pragmatic investigation in the sense of [97].

Indeed, research in programming languages can advance natural language research, and vice versa, through the exchange of insight between the two fields. For example, Chomsky's generative grammar [98] is successfully used as formal grammar in compilers, as discussed in [85][p. 19-34]. On the other hand, applying computing theories in linguistics generates formal semantics. Thus, the rest of this section introduces two major computing frameworks that linguistics uses, namely λ calculus by [3] and type theories by [99].[2] These are foundational frameworks for programming language semantics; the first of them was introduced into linguistics in [4, 100, 101, 14], and the second of them in [86]. Furthermore, the next section introduces a problem of ambiguity.

2.1.1 λ-calculus in linguistics

In this part, we are going to introduce the notion of λ-calculus in linguistics [3]. It is usually called Montagovian semantics, recognizing the pioneering research of Montague [4]. For an introduction to λ-calculus, I suggest [14]

[2] There are many type theories. I use Martin's in this dissertation.

for its simplicity and rich linguistic explanations. For more contemporary research, see, for example, [42, 102, 103].

To begin with, suppose that we want to express a function called f. Function f does the following: when a natural number is provided to f, f returns a value which is equal to that number plus one. We can describe f using the symbolic formation

$$f : \mathbb{N} \to \mathbb{N}$$

The above formula means that f is a function from the set of natural numbers to the set of natural numbers. This formula is called a 'typed declaration' of a function. Hence, we can describe the details of the function's operation or calculation as

$$f(x) = x + 1$$

.

However, there is a problem with the above declaration. Namely, we have to express the name f every time f is used. This demand may bring further issues such as not being able to organize the name of f if we are making a complex mathematical solution or computer program. To overcome these issues, we may rephrase the statement to

$$\lambda x : x \in \mathbb{N}.x + 1$$

The above notion means that there is a function which takes a natural number x as an argument, and returns $x + 1$. There is no requirement to express the name of a function, viz. f, as in the previous declaration. In general, functions are described in the λ calculus as follows:

$$\lambda x : \alpha.\phi(x)$$

Where $x, \alpha, \phi(x)$, are called the variable, the type or domain, and the value description, respectively. From this formulation, we can express the substitution of a specific value into the function:

$$[\lambda x : x \in \mathbb{N}.x + 1](10) = 10 + 1 = 11$$

The above formula constrains the variable to its domain, the natural numbers, by the set declaration $\{x : x \in \mathbb{N}\}$. Thus, $\mathbb{N}$ is a type of the variable x. According to [104], we can define the types separately as

$$f = \lambda x.x + 1 : \mathbb{N} \to \mathbb{N}$$

$$\begin{aligned} f(10) & \\ &= 10 + 1 \\ &= 11 \end{aligned}$$

In order to describe the domain for the λ notion, we can either use the traditional notion of set in set theory, or we use the notion of types in [104, 99, 86]. In the latter, the expression

$$a : A$$

means that an object a is an element of a type A. There are two major types in [14]. Namely, e is the type of individuals such as *John, Mary*, and t is the type of truth values $\{0, 1\}$. We can add a further type of situation s to describe the situations of an utterance rather than the whole complete worlds in Kripke's semantics [47, 42].

The definition of individuals is taken from the philosophy of logics, as discussed in [105, 106, 107]. The definition of truth values t are defined in classical logics, as discussed in [106, 62, 4]. However, the definition of situations is not quite clear despite its important role. The intensionality in

[14, 102] could also be viewed as an interpretation of situations. In the author's opinion, the situation type s in the sense of [48, 47] represents the pragmatic issues. Finally, from these basic types, we can construct the complex types using the application rule in [14, 4]:

> if ρ and τ are types, then $\langle \rho, \tau \rangle$ denotes the function from the type ρ to the type τ.

This rule exemplifies the compositional principle, following [62], which states that a formula is equivalent to the composition of its subformula. An example of a rule to express this principle is the substitution rule above. Another example is a fragment of English linguistic expression by Montague [4] with a clear illustration by typing declaration:

e	:	$entities$
t	:	$truth$
IV, intransitive vp,	:	$t \to e$
$T, term,$	:	$t \to IV$
TV, transitive vp,	:	$IV \to t$
IAV, IV modified adv,	:	$IV \to IV$
CN, common noun,	:	$t \to e$
adv	:	$t \to t$
$prep$	:	$IAV \to t$
vp	:	$IV \to t$
vp	:	$IV \to IV$

Besides these basic types, we describe the additional monadic and parameterized monadic types in Chapters 4, 6, 7, 8, 9. For related research, see [17, 21, 16] which show monadic types as the primitive objects.

From these typed declarations, we can define the denotation of a linguistic expression based on constraints on these types. For example, according to [14]

$$[\![\mathbf{smoke}]\!] = [\lambda x.x : e.\mathbf{smoke}(x)]$$

So, a predicate is the function, or a test, that maps the set of individuals to

1 if that individual smokes, and maps to 0 otherwise. Thus

$$[\![\textbf{smoke}]\!](\text{Ann}) = 1 \text{ if Ann smokes and 0 otherwise.}$$

If a predicate needs two arguments, we feed it with two λ notions such as

$$[\![\textbf{love}]\!] = [\lambda x : (x : e).[\lambda y : (y : e).y \ \textbf{love} \ x]\!]$$

A similar approach is used for the conjunction operator **and**, as in an example in [14]:

Ann sings and dances

is interpreted as

$$[\![\textbf{and}]\!] = [\lambda f : \langle e, t\rangle.[\lambda g : \langle e, t\rangle.[\lambda x : e.f(x) = g(x) = 1]]]$$

where f, g, x are **sing**, **dance**, and **Ann**, respectively. According to [14], other vacuous English words—possessives, to be, and indefinites—have denotations

$$[\![\textbf{of John}]\!] = [\![\textbf{John}]\!]$$

$$[\![\textbf{be rich}]\!] = [\![\textbf{rich}]\!]$$

$$[\![\textbf{a cat}]\!] = [\![\textbf{cat}]\!]$$

with their interpretation in Montague's semantics as

$$[\![\textbf{of}]\!] = \lambda x : e.x$$

$$[\![\textbf{be}]\!] = \lambda f : \langle e,t\rangle.f$$

$$[\![\textbf{a}]\!] = \lambda f : \langle e,t\rangle.F$$

Heim & Kratzer also interpret adjectives, preposition, and nouns as functions from individuals to truth values. Thus

$$[\![\textbf{cat}]\!] = \lambda x : e.\text{x is a cat}$$

$$[\![\textbf{gray}]\!] = \lambda x : e.\text{x is gray}$$

$$[\![\textbf{out}]\!] = \lambda x : e.\text{x is not in x's home}$$

$$[\![\textbf{part}]\!] = \lambda x : e.[\lambda y : e.\text{y is a part of x}]$$

$$[\![\textbf{fond}]\!] = \lambda x : e.[\lambda y : e.\text{y is fond of x}]$$

$$[\![\textbf{in}]\!] = \lambda x : e.[\lambda y : e.\text{y is in x}]$$

The compositional principle is expressed as below

If $[\![\textbf{Texas}]\!] = \textit{Texas}$ then

$[\![\textbf{in Texas}]\!] = [\![\textbf{in}]\!](\textit{Texas})$

$= [\lambda x : e.[\lambda y : e.\text{y is in x}]](\textit{Texas})$

$= \lambda y : e.\text{y is in Texas}$

Now, the question is, what happens if we have more than one argument for given lexical entries, such as **a city in Texas**? We are doing that by modifying the Montague semantics for the preposition, **in**.

$$[\![\textbf{in}]\!] = \lambda y : e.[\lambda f : \langle e,t\rangle.[\lambda x : e.f(x) = 1 \wedge \text{x is in y}]]$$

$$[\![\textbf{gray}]\!] = \lambda f : \langle e,t\rangle.[\lambda x : e.f(x) = 1 \wedge \text{x is gray}]$$

A problem, called a propositional problem, arises in giving semantics for adjectives. It occurs when we are making subjective comparisons. For example, in the sentences below from [14], "a small elephant" does not mean the same as "a small animal".

Jumbo is a small elephant.

Jumbo is a small animal.

To avoid the problem, [14] strengthens the semantic interpretation of adjectives:

$[\![\textbf{small}]\!] = \lambda f : \langle e, t\rangle.[\lambda xe.f(x) = 1 \wedge \text{size of } x \text{ is below the normal size of}[y : f(y) = 1]]$.

Or, if we are putting it into a context,

$$[\![\textbf{small}]\!] = \lambda x : e.x\text{'s size is below } c,$$

where c is the standard size of salient objects in the utterance context.

Furthermore, [14] interpreted the English definite description, **the**, as

$$[\![\textbf{the}]\!] = \lambda f : \langle e, t\rangle. \text{ there is } x \text{ such that } f(x) = 1 \wedge$$

if exists y such that f(y) = 1 then y = x.

Since the definition of the definite description is not universal uniqueness, the truth condition of uniqueness is local rather than global. Hence, the situation or contextual interpretation is

$$[\![\textbf{the}]\!] = \lambda f : \langle e, t\rangle. \text{ there is } x : C \text{ such that } f(x) = 1 \wedge \text{if exists } y : C$$

such that f(y) = 1 then y = x.

where C is a contextual subset of e

λ-calculus is designed to express computations, so it uses variables in a mathematical way. Thus, if we use it as theoretical linguistic semantics, then we must translate linguistic expressions to have a variable declaration in the λ notation. In other words, we are *abstracting* a linguistic expression. Pragmatically speaking, we also have variables in linguistics. We use variables to express referencing such as anaphora *it, he, she*, or demonstratives *that, this*, or relative clauses. According to [97], they are called *variables reference*. Their meaning are short abbreviations for a complex linguistic expression. Besides variables reference, we also have *individual variables* to analyze quantified propositions or representing λ abstraction. For example, in order to provide the semantics to the sentence,

Every dog is barking.

we need a variable to quantify over the set of *dogs* to give the logical interpretation of the sentence, i.e. $\forall x.\mathbf{dog}(x) \rightarrow \mathbf{barking}(x)$. Another example is the following sentence from [92]:

The king of France is bald.

which has a logical interpretation,

$$\exists x.(\textbf{king of France}(x) \wedge \forall y.\textbf{king of France}(y) \rightarrow y = x) \wedge \mathbf{bald}(x).$$

Semantically, according to [14], a variable denotes an individual which (or who) relates to an assignment of a value. An assignment f, from the mathematical view, is a map from the set of variables to the space of individuals. Thus, an *assignment* of a variable is an individual.

This interpretation is usually called Taski's variable truth assignment function in [106]. This is a single assignment, i.e. a map from variables to individuals. Thus, a trace of a variable under an assignment is the individual that is referred, in the assignment, by the variable. For example, if we have the set of variables $S = \{x, y, z, \cdots\}$, and the set of individuals $U = \{\text{Alice}, \text{Bob}, \text{Carol}, \cdots\}$, then the assignment function f, for example, is a particular map

$$\begin{aligned} x &\mapsto \text{Alice} \\ y &\mapsto \text{Bob} \\ z &\mapsto \text{Carol} \\ &\vdots \end{aligned}$$

Thus, $[\![x]\!]^f = \text{Alice}$, or the trace of x under the assignment f is *Alice*. It is worth noting that the real world contains many assignment functions. For a particular semantic interpreting model, we usually limit our choices to the meaningful assignment functions.

If an expression is true in all assignments, we simply omit the superscript of the assignment function, i.e.

$$\forall f, [\![\alpha]\!]^f = [\![\alpha]\!] = \lambda x : e.\alpha x$$

.

For example,

$$\forall f, [\![\textbf{laugh}]\!]^f = [\![\textbf{laugh}]\!] = \lambda x : e.\text{x laughs}$$

There is no formal definition of *variables* in linguistics up to the author's knowledge, as discussed in [14]. However, the phenomenon of variables is a foundational assumption in logics and mathematics, such as in [92]. In the author's opinion, the variables in theories of formal languages such as logics and mathematics has a domain of interpretation which, while implicit, is still clear and not vague. However, when we apply that notion to linguistics, it results in a vague and confused terminology because the background assumptions are changed. Thus, the author stresses at this point for readers to be aware of mismatches between natural languages and formal languages such as logics or mathematics.

Now, we come back to the basic logical operators of quantifications and how to interpret them in λ-calculus. The first question is, what are quantifications in linguistics? And the second is how to interpret these quantifications. To answer the first question, we follow the definition of generalized quantifications in [108, 14], described later in this chapter. Quantifications, in traditional logics, is expressed using an operator such as the $\forall$ and $\exists$ notations. Linguistically, their appearance takes various forms such as

everything, nothing, something, few,$\cdots$.

It should be noted that the idea of interpreting these quantifications as entities (or individuals, e), or as a function from individuals to truth values ($\langle e, t\rangle$) in λ-calculus is not possible. That is because they require other expressions to form a phrasal meaning, such as in the following sentence, **[every dog] barks**. Thus, we usually interpret quantifications as higher-order types that take a domain of interpretation, and an interpreting predicate, to complete the meaning. For example, in this sentence, the quantification, **every** requires the domain of interpretation, **dogs**, and a completing predicate, **bark**. This generalized approach to quantifications provides a general type for them as

$$[\![\textbf{everything}]\!] = \lambda f : \langle e, t\rangle . \forall x : e . f(x) = 1$$

$$[\![\textbf{something}]\!] = \lambda f : \langle e, t\rangle . \text{there is some } x : e . f(x) = 1$$

since we want to restrict quantifications over a specific domain rather than over the whole domain of individuals e. Thus, in the above sentence, we quantify over the set of particular **dogs** in the speaker's view, rather than the whole set of all **dogs** in all universes in all of present, past, and future. This is done by adding the extra function to restrict the domain of quantifying. Hence, their interpretations are

$$[\![\textbf{nothing}]\!] = \lambda f : \langle e,t\rangle.\lambda g : \langle e,t\rangle.\text{there is no } x : e\text{such that } g(x) = 1 \wedge f(x) = 1.$$

$$[\![\textbf{everything}]\!] = \lambda f : \langle e,t\rangle.\lambda g : \langle e,t\rangle.\forall x : e \text{ such that } g(x) = 1 \text{ and } f(x) = 1$$

$$[\![\textbf{something}]\!] = \lambda f : \langle e,t\rangle.\lambda g : \langle e,t\rangle.\text{ there is some } x : e.g(x) = 1 \text{ and } f(x) = 1$$

Finally, we are going to introduce the interpretation of *pronouns* and *bound variables* in the λ-calculus. The term *pronouns*, according to Heim & Katzer [14], is defined as

> A pronoun is used deictically when it receives its reference from the extralinguistic utterance context, and it is used anaphorically when it "picks up its reference" from another phrase in the surrounding text.

However, there is a case in which pronouns do not have a referencing object. This is a bound variable, and occurs in situations such as

Every man put a screen in front of him.

Here, the pronoun *him* does not refer to any real physical object, yet the meaning is clear. We usually have this interpretation in *propositional attitude predicates* such as *think, believe, know, hope, be aware.* For example, let us take an example from [97]

Sally thinks the kid who lives next door to Ann could be a top gymnast.

Here, the bound variable *the kid who lives next door to Ann* could not refer to any particular individual, yet the sentential meaning is clear. Theoretically, the interpretation of a bound variable is as a co-index with the restrictors under the quantification, as in the first sentence, or the opaque contexts, as in the second sentence. In the above examples, the bound variable is a particular *man* or *Sally's mind.* Similarly, the interpretation of pronouns is supplemented by the utterance contexts and variable assignments. For example, if we have an utterance situation c_1 with the assignment g_{c_1}:

$$g_{c_1} = \begin{bmatrix} 1 \rightarrow Kim \\ 2 \rightarrow Sandy \end{bmatrix}$$

Then, in the utterance,

She$_1$ is taller than *she*$_2$

means that

Kim is taller than Sandy.

A detailed analysis of the relation between utterance context and pronoun is referred to the discourse representation theory (DRT) by [56] or dynamic semantics by [109, 110, 44, 111]. Notably, combining λ-calculus and DRT yields the cDRT by [24]. The cDRT is analysed in the Chapter 7 and a short summary of dynamic semantics is given later in this chapter.

2.1.2 Type theories

This section provides as background knowledge on type-theoretical semantics by [86, 112], rather than the typed categorical grammar by [113] and [6]. It also serves as an intuitive explanation for the product and exponential in the next chapter. Furthermore, the towering notion in [17, 12] can be viewed as a type in this dissertation. Indeed, a similar format is derived by [114] to formulate the syntactic calculus in type theories. Finally, type-theoretical semantics also acts as a theoretical proof framework in which to express the syntax and semantics of Chapter 8.

To begin with, let us start with basic definitions.

2.1.2.1 Judgements

According to [86][p. 2], judgements are one kind of linguistic act, or an act in a broader interpretation. Thus, a judgements-oriented framework focuses more on the pragmatics aspect. Judgements have a special form:

$$\vdash A$$

which means an assertion that a proposition A is true. From judgements, we can set up inference rules to state relations between judgements. According to [115], we represent inferences rules horizontally, and separate them vertically by a line. The judgements above the line are called premises, and the judgements below the line are called conclusions. For example, let *John* be a subject, *runs* is a predicate in the sense of Aristotle [91]. We conclude that *John runs* is a sentence in linguistics as

$$\frac{John : NP \quad run : NP \to S}{John\ runs : S}$$

In the above sentence, we omit the $\vdash$ notion for convenience. A further note is that a judgement is an indicative mood by [86][p. 26]; it does not cover the emotional moods such as the interrogative or imperative moods. Another example of an inference rule is the sentence below:

$$\frac{\text{all men are mortal} \quad \text{Socrates is a man}}{\text{Socrates is mortal}}$$

If we go one step further to abstract from the *mortal* property as *Prop*, and *Socrates* as an individual, then we have the following inference rule:

$$\frac{\text{All men are Prop} \quad \text{I is a man}}{\text{I is Prop}}$$

where *Prop* and *I* are called metavariables. They are called metavariables because they interact with the judgements to make sense of judgement, rather than being concrete objects.

We can now understand an axiom as being an inference rule without premises. For example, we declare that a proper name *Alice* is a noun phrase

$$\frac{}{Alice : NP}$$

2.1.2.2 Proofs in linguistics

Intuitively, a proof system is a set of inferences rules. A proof is a particular instance or application of a proof system. For example, the derivation of the proof of the utterance, *John runs* as a sentence is

$$\frac{\overline{John : NP} \quad \overline{run : NP \to S}}{John\ runs : S}$$

The definition of proof in linguistics can be basically adopted from the definition of proof in logics. Past research, such as [116], has shown a close relation between logics and the semantics of natural languages. In particular, we can follow [6][p. 20] to use the term *proof* in linguistics as a derivation of connected inferences. Hence, the definition of *grammar* is a system of inference rules and the set of basic meaning interpretations. From this perspective, the derivation tree results from natural deduction. A full explanation of proof-theoretic semantics in linguistics is given in by [2]. For example, a natural deduction for the sentence *Alice thinks vanilla* is

$$\dfrac{\dfrac{}{\textbf{Alice}\text{ is a subject}}\,\textit{postulate} \qquad \textbf{thinks vanilla}\text{ is a predicate}}{\textbf{Alice thinks vanilla}\text{ is a sentence}}\,\textit{Composition}$$

This dissertation focuses on the typing aspect as a means of carrying semantic values, in contrast to [6], where Shan uses types as a syntactic classification. Thus, we are still carrying the computer science tradition—of interpreting types as semantic values—to linguistics analysis. Basically, types are used to classify linguistic expressions. However, as linguistic expressions are broadly classified rather than limited to expressions in programming languages, we consider the types in natural languages by their relevance. Ill-types in our research means that the types are not relevant to the situations of interpretation, rather than letting it be absent as in Shan's interpretation.

Thus, this research is similar to the pioneering investigations of type theory in linguistics such as in [86, 112]. The difference between this research and theirs is to interpret types according to relevance. We are not seeking the absolute truth in linguistics as is done in logic. Rather, we seek understandings that are true to some extent. This sense of being 'true to some extent' may be understood as applying the principle of relevance in a way similar to Frege's principle of compositionality [62]. This approach is also discussed as

Berg's criterion by [117] in section 6.2. Recognizing this principle is crucial since natural languages are not exactly the same as computer programming languages, even both are constructed by humans. The distinction is due to the empirical research showing that the semantic values in linguistics are more ambiguous than in programming languages. A discussion of the ambiguity can be seen in the next section 2.2 of this Chapter.

The idea of relevance is similar to the idea of possible-worlds semantics, or *intensionality* [6], or *approximating* techniques [86][p. 55]. A phrase is called intensional when we cannot find its references in a real world, for example in one's imagination, such as for *wish* or *believe*. Thus, to provide the semantics for such a phrase, we say that it is true in a given context, or in a possible world. In another words, the phrase has intensionality in a particular or a relevance context. See [102] for further details.

2.1.2.3 Untyped λ-calculus in a typed theory

We use the type theory, in the sense of [99], as a foundational framework to express monads and λ-calculus. There are various extensions of λ-calculus, which is the foundation of programming languages. In this dissertation, we refer to the seminal paper by [3]. On the linguistic applications of type theory, we refer to [86, 112, 118].

A related research theoretical framework with type theory is proof theory.

Proof theory's counterpart is model theory, as described in [119]. In this dissertation, I use a type theory to express inference rules of expressions' denotation, and λ-calculus to express an expression's operation. Firstly, let us follow [6] to explain how the λ-calculus and simply typed λ-calculus are expressed in a type theory. Γ is the notion of a context; E is an abbreviation of an expression. Thus, the rules in untyped λ-calculus are represented in a type theory as

$$\frac{}{x \vdash x}\mathit{Id}$$

$$\frac{\Gamma, x \vdash E}{\Gamma \vdash \lambda x.E}\mathit{Abstract}$$

$$\frac{\Gamma \vdash F \quad \Gamma \vdash E}{\Gamma \vdash F\ E}\mathit{Apply}$$

$$\frac{\Gamma, \Delta \vdash E}{\Gamma, \Delta, \Theta \vdash E}\mathit{Weaken}$$

$$\frac{\Gamma, \Delta, \Delta \vdash E}{\Gamma, \Delta \vdash E}\mathit{Contract}$$

$$\frac{\Gamma, \Delta, \Theta \vdash E}{\Gamma, \Theta, \Delta \vdash E}\mathit{Exchange}$$

$$\frac{\Gamma, (\Delta, \Theta), \Phi \vdash E}{\Gamma, \Delta, (\Theta, \Phi) \vdash E}\mathit{Associate}$$

Table 2.2: untyped λ-calculus in type theories

The typed λ-calculus, according to [120], has a general judgement of the form $\Gamma \vdash e : T$ where Γ is a context, e is an expression and T is its associated type. Hence, a typed system of the typed λ-calculus is represented as

$$\frac{}{x : T \vdash x : T}\,Id$$

$$\frac{\Gamma, x : T_1 \vdash E : T_2}{\Gamma \vdash \lambda x.E : T_1 \rightarrow T_2}\,Abstract$$

$$\frac{\Gamma \vdash F : T_1 \rightarrow T_2 \quad \Gamma \vdash E : T_1}{\Gamma \vdash FE : T_2}\,Apply$$

$$\frac{\Gamma, \Delta \vdash E : T}{\Gamma, \Delta, \Theta \vdash E : T}\,Weaken$$

$$\frac{\Gamma, \Delta, \Delta \vdash E : T}{\Gamma, \Delta \vdash E : T}\,Contract$$

$$\frac{\Gamma, \Delta, \Theta \vdash E : T}{\Gamma, \Theta, \Delta \vdash E : T}\,Exchange$$

$$\frac{\Gamma, (\Delta, \Theta), \Phi \vdash E : T}{\Gamma, \Delta, (\Theta, \Phi) \vdash E : T}\,Associate$$

Table 2.3: typed λ-calculus in type theories

The above notion keeps the context Γ both complete and global. We can add the local contexts to make the theory more accessible, as in the case of evaluation context in [6]. We can do so by using metavariables to model an evaluation context, as also stated in [6][p. 26]. Detailed applications of metavariables in type theory and monads are described in [121, 122]. Since we are working in linguistics, the author follows [123][p. 7] to define the metavariable slightly different. We said that the metavariable in [ ] is relative to the context Γ as (Γ)[]. Thus, the rules for the contextual evaluation and other grammar, for example, in [6][p. 33] are being kept unchanged as follows

$\dfrac{}{[]}$
$\dfrac{C[]}{C[\lambda(x : T).[]]}$
$\dfrac{C[] \quad \Gamma \vdash E : T_1}{C[((\Gamma)[]) \ E]}$
$\dfrac{\Gamma \vdash F : T_0 \ C[]}{C[F \ ((\Gamma)[])]}$

Table 2.4: local evaluation context

Generally speaking, the interpretation of λ-calculus in type theory is the interpretation of programming in logics by the propositions-as-types principle, as described below. Notable contemporary research on this includes [121, 86, 112]. The intuitive idea is that the dependent product types Π are being used to represent the type of the λ abstraction. A further discussion of the relation between λ calculus and constructive type theory can be found in, for example, [124].

In the author's opinion, the introduction of the product and sum operators to λ-calculus in [6] is represented by the dependent Π and Σ types in the type-theoretic semantic part in [86]. Thus, the current development of type theory, especially as described in [121], is expressive enough to capture the formal system in [6].

2.1.2.4 The Curry–Howard correspondence

The Curry–Howard correspondence has another name: the *propositions-as-types principle*. It states the correspondence between propositions and types, or between logics and types in general. According to [86], who followed Heyting, a proposition is an *expectation*, and to understand a proposition is to understand what fulfils the expectation. The word, 'fulfil' later become the word, *proof* in intuitionistic logic. These concepts are also called *problems* and *solutions* by Kolmogorov. A proposition is a statement of a problem, and a proof is a solution. Thus, a proposition A is true if

The problem A has a solution

According to [86] the problems (propositions) and solutions (proofs) have a corresponding form by Heyting.

Proposition	has a proof
$\bot$	-
$A\&B$	a proof of A and a proof of B
$A \vee B$	a proof of A or a proof of B
$A \supset B$	A method for obtaining a proof of B from any proof of A
$\neg A$	a method for obtaining a proof of$\bot$from any proof of A
$(\forall x : A)B(x)$	a method for obtaining a proof of B(a) for any a : A
$(\exists x : A)B(x)$	an element a : A and a proof of B(a)

In type theory, a type represents a proposition as the proposition as a type principle, and provides an element to a given type as the proof of the proposition. Thus, a proposition is true if the type (set) has an element. In short, we write $a : A$ for a reading that a is an element of type A. Hence, the judgement $a : A$, has the following explanations, in the view of each of the above interpretations

$a : A$	A true	*comment*
a is an element of the set A	A has an element	Curry–Howard
a is the proof of the proposition A	A is true	Gentzen
a fulfils the expectation A	A is fulfil	Heyting

According to [86] and [6], if we are taking λ-calculus as a semantics of proofs, then we have the corresponding formations

Proofs of	Formations
$\perp$	-
$A\&B$	$(a, b)$where a : A, b:B
$A \vee B$	a canonical injection i(a) when a : A or j(b) when b: B
$A \supset B$	an λ abstraction $\lambda x.b(x)$where b(x) : B (x :A)
$\neg A$	an λ abstraction $\lambda x.b(x)$where$b(x) : \perp(x : A)$
$(\forall x : A)B(x)$	an λ abstraction $\lambda x.b(x)$where b(x) : B (x :A)
$(\exists x : A)B(x)$	a pair (a,b) where a : A and b : B

Table 2.5: Curry–Howard correspondence

The distinction between this set of correspondences and the above inferences rule is that the earlier rule provides the semantics to a formula, while the latter set of correspondences describes how the formula is defined. Hence, in computing, the Curry-Howard correspondence leads to the insight that constructing mathematical proofs is equal to constructing computer programs. Problems, or specifications, or formulae, are types. Solutions, or proofs, are programs. Deriving a proof is executing a program. The author will use type theories, or an intuitionistic type theory in [99], as a particular example to demonstrate that idea later in this dissertation.

We should note a distinction between deriving a proof and the proof itself. Analogically, there is a distinction between executing a program and its output. Deriving a proof is a proof process where the process may be undecidable or non-terminating, while a proof is a concrete or canonical object which can be tested or verified.

This distinction has several implications in linguistics. For example, [125], following Van Benthem, shows the correspondence between Lamberk's categorial grammar and Montague's semantics. The categorial grammar is the typing style, while Montague's semantics is the proof's type. Similarly, [6][p. 38] interprets the correspondence in linguistics as the syntactic category for formulae or types, and semantics, for utterance meanings.

2.1.2.5 An intuitionistic type theory

We use Martin-Löf's type theory by [99] as a description of the theoretical foundation. Martin-Löf's type theory has influenced computer science as the foundation for theorem-proving programming languages such as Coq [126] and Agda [121]. The type-theoretical advantages are based on two major points. Firstly, the theory is suitable for formalizing computational processes by embedding λ-calculus, the theoretical foundation of programming languages, by using the introduction and elimination rules for abstraction and substitution. Thus, the normalization process in [3, 6] is interpreted as a set of elimination rules. Secondly, Martin-Löf's type theory is open for extensions. Additional types can be added to the original theoretical framework under specific circumstances. Notable extensional types include inductive datatypes, internal type theory, coercive subtyping, metavariables.

Through the Curry–Howard correspondence, there is an equivalence between types and logics. However, the Martin-Löf's type theory has a richer expressive power than first-order logic. This is because the theory has progressive conjunctions, so that latter parts of the formula can depend on previous parts. In linguistics, such back-reference is used to formalize the *donkey* sentence in [86]. We will come back to give further explanation on progressive conjunctions.

General speaking, the syntax of a type theory has a formula

$$\Gamma \vdash a{:}A$$

where Γ is the context, and $\vdash$ is a judgement. a is a term, and A is a type. According to [86], a judgement is in an *indicative mood*. This means that other expressing moods, such as *imperative, mental*, are not in the focus of the theory. Indeed, we can follow [93] to trace back a judgement $\vdash$ as Kant's short notion for a statement of *I assert that*. Therefore, a particular type theory is quite pragmatically oriented in comparison with the tradition of logics such as Aristotle's logic.

The context Γ contains a list of assumptions which have the form $a_i : A_i$. In other words, $\Gamma = \{a_1 : A_1, a_2 : A_2, \ldots, a_n : A_n\}$. A term a is, basically, introduced to the theory by introduction and elimination rules of a type A. The introduction rules show how we formulate a term, and the elimination rules show how we operate on these terms and their associated types. On the other hand, when a type is primitive in type theory, we use formation rules to construct the type A. Therefore, the above formula has an interpretation: under the context Γ, there is a conclusion that a term a has a type A.

It should be noted that the relationships between types and terms are far from easy. Indeed, if we are focusing on the operation of terms, we have the research domain on functional programming, while we have the research on proof assistants if we are focusing on type operation. The main issue here is the unification problem. Intuitively, the unification problem can be stated as, given a term a, how can we decide that it has an associated type A? For further research on this topic, see [127].

The Curry–Howard isomorphism, i.e. the *propositions-as-types* principle, plays a central role in the above formulae. The isomorphism states the analogy between logics and types. More clearly, it means that we can express and understand a proposition under typed notions. The proposition p is true if, and only if, there is a term a such that a has a type A, where a is an interpretation of p. In short, $p : \mathrm{Prop}$ if, and only if, $\Gamma \vdash a : A$. In computing, for example, this principle is applied as a type-checking notion in programming languages [128, 88].

Let us take the Π type to illustrate how to construct a type in type theory. Other basic types such as the type of natural numbers N, the sum type Σ, the ordering type W, or the universe U are covered by [99].

$$\dfrac{(A:\text{Type}) \quad \begin{matrix}[x:A]\\ \vdots \\ B(x):\text{Type}\end{matrix}}{(\Pi x:A)B(x):\text{Type}}\,\Pi\text{form} \qquad \dfrac{(x:A) \quad \begin{matrix}[x:A]\\ \vdots \\ b(x):B(x)\end{matrix}}{\lambda x.b(x):(\Pi x:A)B(x)}\,\Pi\text{intro}$$

$$\dfrac{a:A \quad f:(\Pi x:A)B(x)}{fa:B(a)}\,\Pi\text{elim} \qquad \dfrac{(a:A) \quad \begin{matrix}[x:A]\\ \vdots \\ b(x):B(x)\end{matrix}}{(\lambda x.b(x))a=b(a):B(a)}\,\Pi\text{equal}$$

Each type, for example the above Π type, contains four basic judgements, namely: formation, introduction, elimination, and equality rules. The formation rule sets up the syntax or symbolic definition of a particular type. The introduction rule defines how terms or canonical objects are constructed in the given type. On the contrary, the elimination rule combines or substitutes objects of complex types into the simple one. Finally, the equality rule states the relation between terms, and establishes the proof when two terms are equal.

In the above example, all rules are stated in natural deduction or Gentzen's style. Above the bar are the premises or conditions, and below the bar is the derivation or conclusion. The formation rule Πform establishes how the Π type is constructed. We have two premises, a type A, and a rule that associates each element x of type A to a type $B(x)$.

The judgement Πintroduction, i.e. the Πintro rule, shows the construction of canonical objects of the given type Π. Πintro can be read informally as follows: if, for each element x of type A, there is a function $b(x)$ of type $B(x)$, then $b(x)$ has a Π type $\Pi(x : A)B(x)$. Roughly speaking, the Π introduction rule provides the type for the λ abstraction $\lambda x.b(x)$. Indeed, we can view Π type as a functional space in mathematics, i.e. a list of all functions from A to B.

Similarly, the Π elimination rule, Πelim, plays the role of substitution in λ-calculus. It reduces complex types, such as Π types, to simple ones. Πelim can be read as: for a given term a of type A, and a function f of the Π type $(\Pi(x : A))B(x)$, we can get the term $f(a)$ of type $B(a)$.

Finally, Πequal lets us know when two terms are equal. It relates introduction and elimination rules by making an equation between a canonical object, which is generated by introduction rules, and its correspondent by operation of an elimination rule. It can be read as: for a specific term a of type A, a term which is introduced by the introduction rule, i.e. $\lambda x.b(x)$, with an application to a, is $b(a)$.

2.1.2.6 Extensions of a type theory

Type theories are based on proof theories so their strength is based on the construction of concrete objects. These objects are called canonical objects, which have an identical property by the **I** rule in [99]. Their deriving rules follow natural deduction rules and the *hypothetical judgement*, which means that, given $a : A$, we substitute a for x in $f(x)$, which results in a type B. Thus, if $y = x$, then $f(y)$ also has an element equal to $f(a)$, which has a type B.

Therefore, a type theory is a good candidate for a system that requires correct interpretation. An example of the system is a theorem prover such as in [126], or verification of mathematical texts or computer programs as in [129, 121].

This dissertation cannot cover all contemporary extensions of type theories. It is worth noting, however, extensions such as dependent type [130], logics-enriched type theories [131], UTT [132] and its implementation in Agda [121], Hoare type theory [133], modal type theory [123], and, recently, the development of homotopy type theory [134].

2.1.2.7 The multi-modal type-logical grammar

The multi-modal type-logical grammar is presented in [6] and [135]. We represent multi-modal operators in type-theoretical semantics by using [86][p. 150], which follows the idea on a choice sequence by Martin-Löf, where the necessary and possibility operator are interpreted as the Π and Σ types as

$$(\Box\Gamma)A(x) = (\Pi\Gamma)A(x) : prop,$$

$$(\Diamond\Gamma)A(x) = (\Sigma\Gamma)A(x) : prop$$

There is an alternative way to represent in contextual modal type theory [123]: a modality is represented by a hypothetic judgement.

2.1.3 Type-theoretical semantics

There are various approaches to type theories in linguistics. For example, [86] provides a computational linguistics approach to type theories. Recently, [112] provides a logical approach to the application of type theories in linguistics with a dedicated special issue in the *Journal of Language Modelling* [136]. In addition, [137] uses record type to unify linguistic frameworks such as dynamic semantics and head-driven phrase structure grammar under type theories. Hence, its fruitful result is the dialog system in [118]. Finally, there are also other researchers who apply type theories in linguistics such as [138, 139]. Thus, the basic concepts are described below.

2.1.3.1 Modern typed theoretical semantics

Type theory, as an instance of a proof theory, is one of several major formalisms for parsed natural languages. Typed theoretical semantics is established by using type theories as a formalism. To begin with, we start with an English sentence from [18]:

Walter snores.

The sentence consists of two words: *Walter*, and *snores*. The noun, *Walter* is a *name*, with an assumption that it is *referred* to the individual called "Walter". From that assumption, the expression *Walter* is completed, which means that it is self-explained. There is no need of extra linguistic variables to describe the name. In other words, the name is being taken as a constant. Thus, *Walter* has a type or category *Individual*, the set of all individuals; in short *Ind*.

On the other hand, the word, *snore* is a verb or a predicate. Intuitively, it is a function that requires additional parameters for completion. A predicate has no meaning when it stands alone, with one exception in a special context where the object is inferred implicitly. In other words, this word is a function.

In the logical tradition, the semantics of a sentence is a proposition which represents truth values. If we are taking the compositional principle in [62], which states that the meaning of a whole is the totality of the meaning of its parts, then the meaning of a sentence is a combination, for a short interpretation, of a verb and its associated noun or noun phrase in the sentence. For example, the semantics of the sentence, *Walter snores* is the combination of the meaning of a predicate *snores*, and a noun *Walter*. In traditional logics, the meaning of a sentence has a category, or type, *Proposition*. We abbreviate *proposition* as *Prop*.

For simplicity, we say that *snore* has a category $Ind \rightarrow Prop$, or type $\Pi(x : Ind)Prop$. It is a predicate that requires an individual to complete it, resulting in a proposition. In a summary,

$Walter : Ind$

$snore \ : \Pi(x : Ind)Prop$

$Walter\ snores = snore\ (Walter) : Prop$

The first of these three lines means that: *Walter* has a category *Ind*.

The second line means that *snore* is a predicate that takes an individual and returns the proposition.

The third, and final, line means that the composition of the predicate *snore* and a noun *Walter*, i.e. *snore (Walter)* has a syntactic sugar (see [86]) as *Walter snores*, and is a proposition.

Let us consider another example from [18]

Walter knows Kevin.

The verb, *know*, with its singular form *knows*, requires two parameters to complete its meaning. Thus, it has the predicate form

$$\textit{know} : \textit{Ind} \rightarrow \textit{Ind} \rightarrow \textit{Prop}.$$

or a type

$$\textit{know} : \Pi(x : \textit{Ind})(y : \textit{Ind})\textit{Prop}.$$

There is a difference between a predicate which focuses on representing the meaning, and a verb with its associated syntax. For example, in the above sentence, *Walter Knows Kevin* and *Kevin knows Walter* have two different meanings. Hence, the question of placing the individuals, such as *Walter* and *Kevin*, around the predicate, such as *know*, is concerning. If the verb is represented as a predicate regardless of words order, it is possible to write as *know Walter Kevin* for interpretation as either of *Walter knows Kevin* or *Kevin knows Walter*. However, to avoid this ambiguity, we must derive the order of composition of a predicate. In another words, we should have a prefix and postfix around the verb *know*.

Therefore, the type or category should include the linguistic fact of whether an individual appears on the right of the verb, or on its left. Following [140], two kinds of predicative categories are distinguished in [18]: A/B (A over B) which expects that the missing piece is on the right, and A\B (A under B) which assumes that the missing part is on the left.

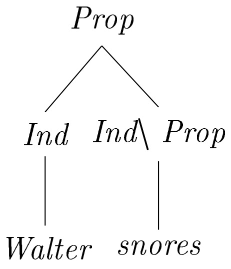
The adequacy formalization of the above example is

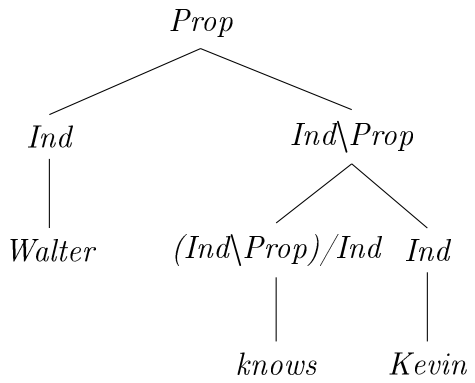
Walter, Kevin : Ind

Snore : Ind\Prop

Know : (Ind\Prop)/Ind

It has the following illustrated tree





2.1.3.2 Further examples

In the modern type theory in [112], a common noun is represented as a universe, whereas nouns are interpreted as types in [86]. For example, popular nouns such as *man, woman, cigarette, ...* are represented as types. On the other hand, verbs, such as *walk, run, light, talk, ...* are represented as predicates, as is the tradition in logics. Logical operators such as *and, or* conjunctions are represented by the Σ type, and implications are represented by the Π type. Let us demonstrate these ideas through the following examples.

1) The sentence, **[John walks]** is formalized as, $walk(J) : Prop$ where: J is *John*, and $walk : human \rightarrow Prop$. J is a term of type *man*, i.e. *J : man*. *man* is a subtype of *human*, i.e. $man \leq human$. Thus, *walk(J)* is a valid proposition under above assumptions.

2) The conditional sentence, **[John walks and Mary runs]** is formalized as, $walk(J) \wedge run(\mathrm{M}) : Prop$, where: $J : Man$; $M : Woman$; $Man,\ Woman \leq Human$; $walk,\ run : Human \rightarrow Prop$; $\wedge$ is a conjunction type as presented above.

3) The existence sentence, **[John takes a cigarette]** is formalized as, $(\exists x : cigarette) take(J, x)$, where, for simplicity: $take : Human \rightarrow Human \rightarrow Prop$.

4) The *donkey* sentence, **[Every man who owns a donkey beats it]** has the above interpretation as .

$(\Pi z : (\Pi(x : man)(\exists(y : donkey) own(x, y)))) beat(p(z), p(q(z))).$ [3]

This formalization follows the progressive conjunction approach. There is another, alternative, approach in which an anaphoric expression *it* is treated as a metavariable. However, that approach is outside this dissertation's scope.

The sentence can have various semantic interpretations which depend on the scope of its quantifications. In the above example, we set the scope of the quantification *every* over *man* rather than over *man who owns a donkey* [141]. Thus, we verify the correctness of a sentence by checking it over the

[3] p and q are the left and right projections π_1, π_2 in [99]

set of man.

2.1.3.3 Universes

Universes has been used to model the common noun of type $\overline{CN}$ in [112]. Basically, the universe $\mathbb{U}$ is a collection of names, where each name is associated with its corresponding type. Thus, it is convenient for us to write $x : \mathbb{U}$ to declare that x is a type rather than to list all types and state that x belongs to the list. Indeed, this lists is described as elements of $\mathbb{U}$. Formally,

$$\mathbb{U} = \{U, \tau : U \to \mathit{Type}\}.$$

where U is a collection of names, and τ is a function that maps each name in U to its associated type.

For example, the universe $\overline{CN}$ represents all common nouns:

$$U = \{\mathit{Man}, \mathit{Woman}, \mathit{Object}, \mathit{Ind}, \text{etc}\}.$$

Each element of U is a type. *Man*, for example, is a type of man. To illustrate this point, we define

$$\tau(\mathit{Man}) = \mathit{man},$$

where *man* is a type of all man. Thus, we can write $x : \tau(\mathit{Man})$, in short $x : \mathit{man}$. But we cannot write $x : \mathit{Man}$ since *Man* is just a name, not a type. Noteworthy, elements of U are constructed by rules in type theory as represented in [99].

In the authors opinion, we can use the universe $\mathbb{U}$ to represent the pragmatics in linguistics. Universes are quite useful when we formalize the proposition that includes quantifications so that we can describe the domain of quantifiers.

2.1.3.4 The progressive conjunction

The distinction between the applications of type theory and first-order logics in linguistics is the *progressive conjunctions* property, originally defined in [86]. The progressive conjunction allows the latter part of a sentence to depend on previous parts. We cannot express this property in first-order logic and the donkey sentence below, which is discussed in section 7.4, is an illustrated example

If Perdo owns a donkey, he beats it.

It is clear that the semantics of the second clause depends on the interpretation of the previous clause in the above sentence. It is problematic to formalize in the first-order logics since it does not express the subformula dependency. In type theories, both existence and conjunction are formalized as the Σ type in type theory, and existence is a special case of conjunctions where the second part depends on the first part. Thus, *progressive conjunctions* is another name for existential propositions.

In order to explain the situation in detail, let us rewrite the existence and conjunction in type theory as

$\exists$ rules:

$$\frac{(A:\mathit{Type}) \quad \begin{array}{c}[x:A]\\ \vdots\\ B(x):\mathit{prop}\end{array}}{(\exists x:A)B(x):\mathit{Prop}}\exists\text{Formation} \qquad \frac{a:A \quad b:B(a)}{(a,b):(\exists x:A)B(x)}\exists\text{Introduction}$$

$$\frac{c:(\exists x:A)B(x)}{p(c):A \quad q(c):B(p(c))}\exists\text{Elim} \qquad \frac{(a,b):(\exists x:A)B(x)}{p((a,b))=a:A \;\; q((a,b))=b:B(a)}\exists\text{Equal}$$

p,q are projection functions that select the first and second elements of a pair, respectively.

Conjunction rules:

$$\frac{A:\mathit{Prop} \quad B:\mathit{Prop}}{A\wedge B:\mathit{Prop}}\wedge\text{ Formation} \qquad \frac{a:A \quad b:B}{(a,b):(A\wedge B)}\wedge\text{ Introduction.}$$

$$\frac{(a,b):(A\wedge B)}{p((a,b)):A \quad q((a,b)):B}\wedge\text{Elim} \qquad \frac{(a,b):(A\wedge B)}{p((a,b))=a:A \;\; q((a,b))=b:B}\wedge\text{Equal}$$

The difference between the $\exists$ type and the conjunction types is that the second part of the $\exists$ type depends on the first element of the first part. If we let $B : A \to \mathit{Prop}$, $A : \mathit{Prop}$ as $A : \mathit{Type}$ (by the *propositions as types* principle) in the conjunction rule, then it becomes the $\exists$ rule. Thus, the $\exists$

type is interpreted as a conjunction type when the second proposition is based on the first one. Thus, the donkey sentence can be formalized in type theory as

$$(\Pi(\mathit{proof} : (\Sigma \mathit{Pedro} : \mathit{Ind})(\Sigma d : \mathit{Donkey})\mathit{own}(\mathit{Pedro}, d) : \mathit{Prop})\mathit{beat}(p(\mathit{proof}), q(\mathit{proof})) : \mathit{Prop})$$

where the conditional sentence is interpreted as a Π type. For further discussion, recent research [142] replaces the ∃ rules by the ϵ-calculus. This leads to the interpretation that the derivation of the formalization of a sentence in type theory is the anaphora resolution by [143].

2.1.3.5 Dependent types

The original term, *dependent types*, over intuitionistic type theory by [99] is credited to [130]. However, Dybjer introduced this notion in computer science. For a clear application in linguistics, we follow the research in [114] and [112]. As described above, new types, in intuitionistic type theory, are defined in formation rules based on previous rules. Thus, the expressive power of types stays at the general description. In the propositions as types principles, the propositions are diverse. Thus, type theories need to be developed to express the correspondence. If we want types to have more roles rather than just labeling, hence improving their expressive power, Martin-Löf's type theory reaches its limits. One improvement on Martin-Löf's type theory is creating new types that can be defined over previous terms rather than types, thus, creating a layer of interactions between types and terms.

The new type theory is called dependent types. It improves Martin-Löf's type theory by having more precise descriptions of types. Thus, instead of requiring precise descriptions of terms, we can express that on the description of types. Let us illustrate a relation between types and terms by an example of a type A which is a n-tuple vector Vec^n and A depends on n. n must be given explicitly in the definition of A. In intuitionistic type theory, n is left in the assumptions. This leads to n being implicit, and requires another variable in the assumptions for the declaration of n. On the other hand, dependent type automates the declaration of an implicit argument as a new type formation that depends on terms. In the example, the dependent typed version of A is A *: (n : Nat)* Vec^n, i.e. the declaration that the type A depends on the term n.

We follow [114, 144] to introduce the dependent Π and Σ types. The generalized dependent Π type has the following formula:

$$\Pi(x : \alpha)\beta$$

Where α is a type, x is a term, and β is a type which depends on x. The application rule of the product type is

$$\frac{f : (x : \alpha)\beta \quad a : \alpha}{f(a) : \beta(x = a)}$$

Formation and introduction rules are

$$\frac{\alpha : \mathit{Type} \quad \begin{array}{c}(a : \alpha)\\ \\ f(a) : \mathit{Type}\end{array}}{(\Pi a : \alpha) f(a) : \mathit{Type}} \qquad \frac{\begin{array}{c}(a : \alpha)\\ \\ f(a) : F(a)\end{array}}{\lambda a.f(a) : (\Pi a : \alpha) F(a)}$$

The application rule means that: given f, as a dependent type $(x : \alpha)\beta$, and a term a with a type α, $f(a)$ has a type $\beta(a)$ which is a substitution of a for x in β. In cases where β does not depend on α, we have a normal Π type. For simplicity, we write $f : \alpha \rightarrow \beta$.

There is another example of dependent types as the Σ type [144]

$$\Sigma(\textsc{a}, \textsc{b}) \text{ or } \Sigma(x : \textsc{a})\textsc{b}$$

Its terms are a pair (a, b) where a has a type $\textsc{a}$ and b has a type $\textsc{b}(a)$. If $\textsc{b}$ does not depend on $\textsc{a}$, we have a normal Cartesian product type $\textsc{a} \times \textsc{b}$. We extract a and b from (a, b) by projection functions p and q: $p(a, b) = a$ and $q(a, b) = b$.

In typed theoretical semantics, we use the Σ type to interpret modified common nouns by [144] where common nouns are interpreted as types. The idea of interpreting common nouns as types is similar to the idea of interpreting clauses as type [145]. Now, let us give Luo's example of a modified common noun,

$$\Sigma([\![man]\!], [\![handsome]\!])$$

This is a typed interpretation of the modified common noun, *handsome men* of the common noun, *men*. Where $[\![a]\!]$ is the typed semantics of a word a, *handsome* is an adjective with a type $handsome : Man \to Prop$.

In a proof theory, or typed theoretical semantic in particular, the semantic value of a sentence is asserted by providing a term for a given type. For example, the semantic value of a sentence *Joe is a handsome man*, according to [146] , is the proof of an assertion $Joe : \Sigma([\![man]\!], [\![handsome]\!])$. Practically, it is constructed by providing the following assertions

$Joe\ :\ [\![man]\!]$,

Handsome

Proof : $handsome(Joe)$

Where the individual *Joe* is a term of type *man*, and the *handsome Proof* is being found in cognitive assumptions or derived from a mechanizing process.

An application of dependent type in linguistics is the interpretation of syntactic calculus in dependent type [114]. If we interpret the calculus in typed logical grammar in the sense of [18, 113], we have a towering notion in [12, 17].

2.2 Ambiguity in natural languages

There are persistent ambiguities in interpreting the meaning of natural languages despite serious past attempts, all the way from Aristotle to Hilbert's program. Philosophically, it is challenging to formalize the abstract physical definitions such as *center of the universe* or intentionalities such as *belief*. In the strict grammars required in programs such as compilers, [26][p. 5] states that grammars are either incomplete or overgenerating. This claim is supported by claimed technical difficulties in dealing with English ambiguity in [147]. Among the major reasons for the difficulty of parsing natural languages text is their complexity, ambiguity, and specification by collections of examples rather than by complete formal rules for grammars. Another reason is that punctuation is used more sparingly.

From the perspective of what compilers need, the definition of ambiguity in [85][p. 63] is:

> A sentence from a grammar can easily have more than one pro-

> duction tree, i.e., there can easily be more than one way to produce the sentence. From a formal point of view this is a non-issue (a set does not count how many times it contains an element), but as soon as we are interested in the semantics, the difference becomes significant. Not surprisingly, a sentence with more than one production tree is called ambiguous, but we must immediately distinguish between essential ambiguity and spurious ambiguity. The difference comes from the fact that we are not interested in the production trees per se, but rather in the semantics they describe. An ambiguous sentence is spuriously ambiguous if all its production trees describe the same semantics; if some of them differ in their semantics, the ambiguity is essential.

The characterization of ambiguity into essential and spurious in linguistics has not been researched, to the author's knowledge. The present state of research on the spurious ambiguity is given in categorial grammar by [148]. However, instead of that, the disambiguition of various forms of sentence parsing in compiler technique has been used to solved linguistic semantic ambiguity in [149], or scope-taking in [12].

Philosophically, in the author's opinion, the main source of ambiguity in the semantics of natural language is an implicit use of the context, technically termed 'context sensitivity' in programming languages such as in chapter 1 of [6]. This idea is related to the research in [150]. It, for example, includes the common sense in [151]. In the author's opinion, basically, it relates to the problem of requiring context-appropriate relevance of information in order to interpret a formula. For example, if a sentence, *John loves Mary* is evaluated on an utterance, it requires the speakers and listener to have common concerns and intuitions about subjects, viz. *John*, *loves*, and *Mary*, rather than other contextual information of the world such as whether the *king of France* exists or not.[4]

In the author's opinion, the context sensitive in programming languages is similar to the definition of pragmatics in [97]:

> **Semantics** deals with the literal meaning of words and the meaning of the way they are combined, which taken together form the core of meaning, or the starting point from which the whole meaning of a particular utterance is constructed. **Pragmatics** deals with all the ways in which literal meaning must be refined, enriched, or extended to arrive at an understanding of what a speaker meant in uttering a particular expression.

For example, in Kearns' sentence

I forgot the paper.

[4] This idea, in the author's opinion, can be linked to independent logic by [152, 153], which is also proposed as a foundation of mathematics. It is very expressive since dependent logic is not proposed as a foundation of mathematics, to the author's knowledge

semanticists interpret an indexical “I”, predicate “forget” with the past tense, a definite description *the* and an object *paper*. From these interpretations, we can construct the meaning of a sentence as a list of indicative sentences: *There is a person who is speaking. There is a time t in the past that is referred to, and he/she forgot an object which is a paper at that time.*

There is still vagueness in the semantic interpretation due to the pragmatic approach requiring role-play in order to have an accurate sentential meaning. It answers questions such as who is “I”? What time is the utterance? What is its scenario? Or what is the paper? We can only answer these questions by a particular context of an utterance. Thus, in the author’s opinion, we have pragmatic issues by a practical usage of natural languages.

Linguistic ambiguity is far more complex than the compiler’s scope for ambiguity. It can be seen as a sense and reference of a linguistic term in by [62]. In the author’s opinion, sense is an ontology and reference is an epistemology. An example of the reference is the coreference problem between the *morning star* and the *evening star* in [62]. The vagueness problem, for example, discussed in [154], is normally associated with a sense in a term of cognition, and an ambiguity is associated with a non-transparent reference. Recent research by [72] states that the reference problem is the most generic problem in natural languages.

Analogically we can think of the reference problem as providing semantics to a pointer, or to a *goto* statement in computing. However, unlike computing, references in natural languages are hard to grab because references have a complex syntax in languages all over the world, on top of the semantics. An example is the syntax of quantifications such as $\forall, \exists$ or the modality *may, might* in the cross-disciplinary approach of [155] .

Mathematically, according to [94], the type theories developed in [92] avoid the ambiguity. Recent research by [69] also shows the perspective that disambiguity is a coercion of types. However, ambiguity still persists in modern mathematics in linguistics-related or conventional definitions such as universes in category theories.[5]

A list of linguistic ambiguities is studied in chapter 4 of [156]. Furthermore, a study of ambiguity in mathematical text is conducted by [95], where he stated that the major challenges are

- Introduction and use of variables in one sentence.
- Interpretation of symbolic mathematics.
- Linguistic text in mathematics. For example, a sentence such as

  some natural number is prime

  is ambiguous because the adjective *prime* has many senses.
- The interaction between symbolic and linguistic terms in mathematical texts such as *e is prime* and *$G_{n,e}(\mathbb{C})$ is prime.*
- The combination of the above points in sentential analyses.

[5] the universes is used in the proof of Fermat's last theorem.

From above reasoning, a complete study of ambiguity in linguistics is out of the scope of this dissertation. Firstly, unlike computers' language, the terminologies in natural languages, such as *the centre of the universe* is vague. Hence, in the author's opinion, the process of natural languages' disambiguation in accordance with parsing is analogous to providing a semantics for them. Up to the state of the art, it depends on the linguistic phenomena and theoretical backgrounds. Subsequent chapters explain the details of already-studied linguistic phenomena in the background of the category theory; hence the rest of this section provides a complement for these studies.

2.2.1 Quantifications

Quantifiers are a multi-disciplinary research topic. It is an important topic; the author cannot survey all in this dissertation. However, [157, 92] and Henkin's quantifier in [86] are classical mathematical studies. Linguistically, [108] studied and called them as the generalized quantifiers with an English examples as *few, some, most*. These are regarded as a further discussion of the generalized quantifiers in the section on λ-calculus. However, the source for this section is limited to the recent research in [158] and [159, chapter 1]. Quantifiers have been studied since [92], with the first-order logical interpretation of Westerståhl's sentences

Some professors smoke.

The king of France is bald.

as

$$\exists x.\mathbf{professor}(x) \land \mathbf{smoke}(x)$$

and

$$\exists x.\forall y.(\mathbf{king\_\ of\_France}(y) \leftrightarrow y = x) \land \mathbf{bald}(x)$$

The problem with this interpretation is that it is not in the composition as discussed in [159][p. 9]. [4] provides a compositional interpretation in *simple type theory*:

$$(\lambda P.\lambda Q.(\exists x.(P(x) \land Q(x)))(\mathbf{professor}))(\mathbf{smoke})$$

$$((\lambda P.\lambda Q.\exists x.\forall y.P(y) \leftrightarrow y = x \land Q(x))(\mathbf{king\_of\_France}))(\mathbf{bald})$$

Hence, the theoretical background frameworks, such as model theory in previous attempts, plays a pivotal role for the interpretation of these sentences. Besides, the linguistic definition, in conjunction with its analysis, would extend the research boundary of the problem. The definition of the quantifiers is given in [160][p. 445] and [159][p. 10–11] as a functor Q, which is described in the introduction to monads chapter,[6] assigning each set E

[6] This idea is also expressed as a broader perspective, i.e. a desired formalized frame-

a binary relation Q_E between subsets of E. Though, not all functors are quantifiers, they should satisfy the condition

$$\text{for all sets } E \text{ and all permutations}^{7}\ F \text{ of } E \text{ and all } A, B \subseteq E,\ Q_E AB \text{ iff } Q_E F(A)F(B)$$

Linguistically, A denotes a NP and B denotes a VP, so we can have a sentence such as *everyone runs* with a quantifier Q of *every*. In addition, [159][p. 14-16] discussed the properties of the quantifiers. They have the characteristic of universality, which means that the meaning is the same in every applied universe. Thus, *at most ten* has the same meaning in *at most ten men* as it does in *at most ten women*. This phenomenon is a domain restriction or conservativity, i.e.

$$\text{for all sets } E \text{ with } A, B \subseteq E,\ Q_E AB \text{ iff } Q_E A(B \bigcap A)\ ^{8}$$

Thus, the meaning of

Several boys like Sue.

has an inference that

work, of generalized quantifiers in [161][p. 460]. However, **(author?)** linked to category grammar rather than a general category theory.

[7] bijections

[8] This, roughly, means a subset relation.

Several boys are boys who like Sue.

The quantifiers also have the Boolean operators on them so we can construct Westerståhl's sentences such as

Two students and a few professors left the party.

Mary and a few professors left the party.

For the linguistic challenges posed by these properties, readers are referred to chapter 5 of [158]. In addition, other useful properties of quantifiers in linguistics are symmetry and monotonicity in [159][p. 18-21]. The symmetry quantifiers mean

$$Q(A,B) \Rightarrow Q(B,A)$$

Thus, *some, even number of* are symmetry quantifiers while *every, most* are not. Since Q has two arguments, the monotonicity quantifiers can be either left or right increase or decrease arguments as follows

$MON\uparrow: Q(A,B) \& B \subseteq B^{'} \Rightarrow Q(A,B^{'})$

$MON \downarrow: Q(A,B) \& B \supseteq B' \Rightarrow Q(A,B')$

$\uparrow MON: Q(A,B) \& A \subseteq A' \Rightarrow Q(A',B)$

$\downarrow MON: Q(A,B) \& A \supseteq A' \Rightarrow Q(A',B)$

This is the basic view of generalized quantifiers. However, the quantifiers may not have a monotonicity property and is called non-monotonicity; or they may have additional properties such as continuity in [160]. Combining the logical operators and the above properties also yields a comprehensive perspective of quantifiers.

A linguistic view point of this topic can be found in [158, chapter 4]. In addition, pragmatic properties of quantifiers, especially indefinites, such as the scope, are argued as being not uniform in [158, chapter 6]. Finally, other forms of quantification such as distributive quantifiers, bare and modified numeral quantification are discussed in [158, chapters 8–10].

2.2.2 Dynamic semantics

This subsection can not cover all aspects of the topic. For a basic literature on the topic, see discourse representation theory (DRT) in [56] or file-change semantics in [111]. Contemporary research includes [109, 110, 27, 44, 162]. Dynamic semantics is also a framework to explain linguistic phenomena in

[46, 17, 95, 10, 42, 163]. In addition, a recent development is summarised in [19, 164, 16]. Notably, [25][p. 305] also argues that the classic DRT is another form of continuation semantics.

According to [162][p. 8], DRT in accordance with file-change semantics, is a formalized system on model theory of discourse referents which is based on the logico-philosophical research of Geach and Karttunen. Thus, it has a strong connection with (first-order) logics. However, they are not the same, as pointed out by [25][p. 304], since DRT can visualize the reference resolution process. Since the author has already introduced both λ-calculus and type theories, the author skips the details of the framework and introduce the basic idea below. A particular framework of DRT, called the cDRT, is analysed in the Chapter 7.

Formally, the discourse representation structure (DRS) of DRT K is a pair $\langle I, C\rangle$ where I is the universe of a list of discourse referents, and C is a list of DRS conditions. The DRS conditions can be described as

- an atomic condition $Px_1 \cdots x_n$ where P is a n-ary relation and $x_1, \cdots, x_n$ are discourse referents
- $\neg K$ where K is a DRS
- $K_1 \vee K_2$ where K_1, K_2 are DRSs
- $K_1 \Rightarrow K_2$ where K_1, K_2 are DRSs

A DRS $\langle\{x_1,\cdots,x_n\},\{c_1,\cdots,c_m\}\rangle$ is usually called a box, and is represented as

$$\begin{array}{|c|}\hline x_1,\cdots,x_n \\ \hline c_1 \\ \vdots \\ c_m \\ \hline\end{array}$$

A box is a primitive object for DRT in [56], meaning that the context, or information state, is the primitive in DRT. DRT is thus distinguished from the traditional classical logics, where the truth condition is primitive. We can merge boxes, construct a complex discourse from sentences, update it, translate to first-order logic, or provide a semantic model. Hence, boxes can be used to formalize linguistic phenomena, and have become an influential framework in the literature[9]. Its strength lies in its rigorous interpretation of the donkey anaphora. However, it is usually criticized for not being a composition. For example, a translation of the sentence, *John owns a Porsche* into a box is

[9] Other influences on dynamic semantics are DPL, ϵ-calculus, file-change semantics, etc

$$\begin{array}{|c|}\hline x, y \\ \hline \mathbf{Jones}x \\ \mathbf{Porsche}y \\ \mathbf{own}xy \\ \hline \end{array}$$

A discourse such as, *John owns a Porsche. It fascinates him.* is complex to parse as a box because the second sentence includes pronouns. If we skip the pronoun resolution, the two boxes from these sentences can be **merged** using the $\bullet$ operator in [25][p. 304-308], to

$$\begin{array}{|c|}\hline x, y \\ \hline \mathbf{Jones}x \\ \mathbf{Porsche}y \\ \mathbf{own}xy \\ \mathbf{fascinate}yx \\ \hline \end{array}$$

Hence, the above box, can be translated into first-order logic as

$$\exists x, y.\mathbf{Jones}x \wedge \mathbf{Porsche}y \wedge \mathbf{own}xy \wedge \mathbf{fascinate}yx.$$

2.2.3 Scope-taking

The basic literature for scope-taking in linguistics is referred to by [12]. Related research are [165, 166, 75, 6, 17]. A recent research is conducted by [16] with the study of the scope-taking of presuppositions in the graded monads. Section 7.4 of this dissertation also discussed the scope-taking phenomenon. A recent summary of scope-taking research is given by Barker in chapter 2 of [159]. Scope-taking is defined with a syntactic orientation in [159][p. 40] as

> A phrase takes scope over a large expression that contains it when the larger expression serves as the smaller phrase's semantic argument.

Thus, in Barker's example

John said [Mary called [everyone] yesterday] with relief.

The quantification *everyone* takes scope over its nuclear scope of *Mary called everyone yesterday*, and spans over the whole sentence. Hence, it leads to the scope ambiguity in accordance with two major syntactic analyses of linear and inverse-scope reading in a complex sentence, including scope island and ellipsis.

In order to provide the semantics for the challenges, various techniques have been used, as described in [159, chapter 2, sections 2–3]. They are quantifying in, quantifiers raising, Cooper storage, Flexible Montague grammar, a logic in category grammar, and continuation.

A characterization of scopes is given in section 4 of the above research. Scope takers cover all of lowering, split scope, existence versus distributive scope, parasitic scope, and recursive scope. The lowering or total-reconstruction scope taker means that the subject is taking a scope under an embedded clause. For example, in Barker's example

some politician$_i$ is likely [p_i to address John's constituency].

means that the subjects of *politician(s)* is also a subject of an embedded clause, i.e.

There is a politician x such that x is likely to address John's constituency.

Thus, its scope is to narrow down.

The split scope and existential versus distributive quantification mean that an expression can have multiple meanings[10]. An example is the German determiner *kein* (no) which can be either negation or an existence quantification. Another one is the wh-phrases, such as *how many*, in English, which can be either a wh-operator or a generalized quantifier. The detailed

[10] You can think of it as a special kind of co-predication

interpretation depends on the focus of the sentence. For example, in this sentence

How many people should I talk to?

The wh-phrase interpretation is

What number n is such that there are n-many people whom I should talk to?

i.e. how many **people** are there, with a property of the requirement for me to talk to them? On the other hand, the generalized quantifier reading is

What number n is such that I should talk to n-many people?

which means the **number** of people to be talked to.

The parasitic scope means a higher-order scope[11]. It is an expression that takes scope over another scope taker such as *same, different*. For example, in the sentence

Every student reads the same textbook.

same takes scope over the other scope taker, *every*. The same process can be used to interpret *average* in Kennedy and Stanley's sentence,

[11] This is analogous to higher-order logics or higher-order plurality; see [167, 168]

The average American has 2.3 kids.

Finally, recursive scope-taking means an expression that can be another scope-taking expression if it is combined with a scope-taking expression. Barker's example is the English expression, *same*; however, with an alternative interpretation. For example, in the sentence,

Ann and Bill know [some of the same people.]

same is combined with *people* to become another scope-taking phrase, *same people*.

The indefinites and their scope, which is a motivation for dynamic semantics, is analyzed in [159, chapter 2, section 5]. An indefinite can be a reference or quantifiers, and are related to **Skolem functions** that show the analogy between existential quantifiers and operations over the set of individuals, i.e. the formula $\forall x.\exists y.Px \wedge Qy$ is equivalent to $\forall x.Px \wedge Q(fx)$. The function can be probabilistic, and is called a choice function. The quantifiers can be developed further into branching or Henkin's quantifiers. A recent development is dependence logic [153]. Indefinites also have a cumulatives of plurality reading in the sentence:

Two boys read three books.

or the *de dicto/de re* ambiguity in intensionality. For example, Barker's sentence

Mary wants to buy an inexpensive coat.

has a *de dicto* reading that she wants to save money, while its *de re* reading means that Mary bought a coat but is not concerned that it is inexpensive.[12]

2.2.4 Conventional implicatures

The thorough literature review of the phenomenon with a formal semantics approach is summarized in [78]. A short review is given by [169] with an indication that there is no unified formal treatment of speaker-orientation in Potts' theory. An empirical challenge to the non-interaction principle between the at-issue and the conventional implicature, with the proposed solution in the dynamic predicate logics, is given by [80]. An alternative solution is given in section 9.3 of this dissertation. In the author's opinion, the interaction between textual and symbolic meaning in mathematical text in [95] can be viewed as an interaction between at-issue and conventional implicature dimensions. Hence, this viewpoint supports the hypothesis in [80].

General speaking, the conventional implicature (CI) phenomenon is the multidimensional analysis of a sentence. Sections 2 and 3 of [169] provides the theoretical framework and empirical explanation. The detailed discussion of the phenomenon is given in sections 4.4 and 9.3. The definition of the phenomenon, according to [169][p. 710] and [78], is the meaning triggers which:

[12] For the state of the art of other readings, besides the two given, see [159][p. 69]

- Are constituent to the meaning of an utterance in the conventional way, and is non-cancellable.

- Not at the central constituent, or the *at-issue* part, and independent from this part; however, CI can take *at-issue* as an argument.

- Are *scopeless*, or scope-free.

- Are speaker-oriented, except in direct quotations.

Under pragmatic characterization, Potts divided the CI into two main groups of supplements (e.g appositives and parentheticals), and expressive, [78, chapters 4 and 5]. Examples in [169][p. 710–711] are

Supplements

As-parentheticals: Ames was, as the press reported, a successful spy.

Supplementary relatives: Ames, who stole from the FBI, is now behind bars.

Nominal appositives: Sheila believes that Chuck, a psychopath, is fit to watch the kids.

Topic-oriented adverbs: ‘Physically, the keyboard is smaller than I expected, and extremely well built—there’s no creaking or flexing. The keys look as if they will last well—including their paint. Thoughtfully, there is a clip-on cover for the connector while not in use.’

Speaker-oriented adverbs: Motorola said that, amazingly, it has no spare modems.

Utterance-level modifiers: Frankly (speaking), Ed fled.

Expressives

Expressive attributive adjectives: Sue’s dog is really bloody mean.

Epithets: Every Democrat advocating [a proposal for reform]$_1$ says [the stupid thing]$_1$ is worthwhile.

Honorifics (Japanese):

Ame	ga	furi-mashi-ta
rain	SUBJ	fall (HON-PAST)
it rained	(performative honorific)	

or

Yamada sensei-ga o-warai-ni nat-ta

Yamada teacher-NOM HON-laugh-DAT be-PERF

Professor Yamada laughed.

German **Konjunktiv I** which implies a lack of speaker's commitment to an embedded clause:

Sheila	behauptet,	dass	sie	krank	sei.
Sheila	maintains	that	she	sick	be.
KONJ					
Sheila	maintains	that	she	is	sick.

Finally, a further discussion of this phenomenon is provided in section 9.3.1.

2.3 Discussion

This chapter provided a background for this dissertation, introduced the parsing of natural languages and how to address ambiguity. Natural language parsing is approached under the logical perspective with the hypothesis of parsing as deduction. The author also discussed λ-calculus and type theories as parsed frameworks for linguistic phenomena. Besides the above-listed phenomena, also worth investigating is the copredication phenomenon, recently studied in [154, 170, 171, 172, 173]. Finally, a further study of characterization of ambiguity in natural languages, such as essential and spurious ambiguity, in compilers is suggested as a prominent research direction in section 2.2.

Chapter 3

The definition of Monads

According to [157][p. 1], there is a duality between formal and conceptual definitions in mathematics. In a rough sense, formal means logical deductions or axiomatic studies, while conceptual means inside properties of a studied object. For example, the formal approach concerns the deduction of theorems from axioms in group theory, while the conceptual approach considers classes of actual groups to which 'group' refers.

Furthermore, [157] also states that studying the foundation of mathematics means studying the universal mathematical characteristics. Hence, if we focus on the formal aspect, we have logics, and we have a category theory if we concentrate on the conceptual aspect. This idea is developed further to the triangle equality between logics, types, and category theories in [30]. That relationship is called the Curry–Howard–Lambek correspondence.

If we keep the correspondence, an analogy between type and category theories, which is also studied by [174], is summarized and extended by recent studies as

Type theory	≈	Category theory	annotation
Σ type	≈	binary product	$\exists quantifier,$pair of expressions
Π type with intro and elim rules	≈	Exponential with curry and eval	$\forall quantifier,$application and abstraction of an λterm
metavariable in the sense of[121]or hypothesis reasoning in the sense of[28]and[123]	≈	*monad*	express questions or modality
Dependent type theory	≈	locally cartesian closed categories	[175, 176]
dependent linear type theory	≈	indexed monoidal category	quantum computation by[177, 178]

Table 3.1: duality between category and type theories

The introduction of category theory or a comprehensive survey of the equivalent between category and type theories is broad an under contemporary research. Thus, this dissertation is going to introduce basic definitions in category theory which associate with monads as a background knowledge. Let us start by giving a definition of a category, as in [30, 29].

3.1 Basic definitions

Definition 1: A ***category*** $\mathcal{C}$ is defined by:

1. a collection of **objects**, **Obj**($\mathcal{C}$). An object is called A, B, C, etc.

2. a collection of **arrows** (often called a **morphism**), **Arr**($\mathcal{C}$). An arrow is called f, g, h, etc.

3. operations assigning each arrow f to its **domain**—an object *dom f*, and its **codomain**—an object *cod f*. We usually write these traditionally

as

$f : A \to B$ or, graphically, $A \underset{\to}{f} B$.

A collection of all arrows with domain A and codomain B is $\mathbf{C}(A, B)$.

4. If f, g are arrows and $cod\, f = dom\, g$, a **composite** arrow $g \circ f : dom\, f \to cod\, g$ is a *composition* operator with an additional *associative law*:

   for any arrows $f : A \to B, g : B \to C, h : C \to D$,

   $h \circ (g \circ f) = (h \circ g) \circ f$.

5. For each object A, an **identity** arrow $\underset{A}{id} : A \to A$ with an additional law $f \circ id_A = f = id_A \circ f$.

   Intuitively, an object is a *type* or set in type theory or computing, and an arrow represents a judgement or function over the type or set. However, in category theory, the priority is the relation between arrows rather than defining the actual elements in a set. Notably, an associative law is able to be regarded as the substitution rule. An example of a category is the **Set** category by setting:

   (a) A collection of sets as objects (if we do not concern about cardinality and hierarchy).

   (b) Total functions between sets as arrows.

(c) Identity arrows as identity functions in set theory.

To verify that **Set** is a category, we check that the five elements of the above definition all hold. Further categories with definition of objects and arrows, such as *partial order sets* with *monotone function*, *monoids* with *monoid homomorphisms*, *vector spaces* with *linear transforms*, *topological spaces* with *continuous functions*, are illustrated in [29]. These examples substance the coverage of the category theory and show its potential applications.

Linguistically, we can view situations or information states in situation theory in [47, 179] as objects, and arrows as assignment functions or transitions between states. If we restrict the assignments[1] to the associative and identity laws by interpreting them in Kripke's concrete possible-worlds semantics (set theoretic semantics [180]), or transitions to team semantics [181], then we have a categorical interpretation of situation or dynamic semantics.

Another linguistics-oriented example is Church's simply typed λ-calculus [3], interpreted as a closed Cartesian category [182]. It will be discussed in subsequent sections of this chapter. It implies that we can have a categorical interpretation of Montagovian semantics [4], of which sample illustrations can be found in [14] and [25][p. 173-175].

[1] In Carnap's approach to the philosophy of languages [105], an arrow denotes Tarski's assignment function that assigns individuals to variables.

This approach would answer to criticisms that Coecke's research [32] does not cover Montague's semantics.

Definition 2: An object $\mathbf{0}$ is an **initial object** if, $\forall A \in \mathbf{Obj}(\mathcal{C})$, there exists an unique arrow from $\mathbf{0}$ to A.

Definition 3: An object $\mathbf{1}$ is a **terminal object** if, $\forall A \in \mathbf{Obj}(\mathcal{C})$, there exists an unique arrow from A to $\mathbf{1}$.

For example, in the category **set**, the empty set $\{\}$ is the initial object since, for each set $S = \{\overline{a}\}$ in $\mathbf{Obj}(\mathcal{C})$, the only arrow from $\{\}$ to $S = \{\overline{a}\}$ is a map from $\varnothing$ to $\overline{a}$. Each one-element set $\{a\}$ is a terminal object because, for each set S, the arrow from S to $\{a\}$ is a constant function that maps each element of S to a.

Definition 4: A **product** of two objects A and B is an object $A \times B$ with two projections: $\pi_1 : A \times B \to A$, $\pi_2 : A \times B \to B$, such that, for every triple $A \xleftarrow{f} C \xrightarrow{g} B$, there is a unique arrow $\langle f, g \rangle : C \longrightarrow A \times B$ such that

$\pi_1 \circ \langle f, g \rangle = f$ and $\pi_2 \circ \langle f, g \rangle = g$.

The product definition is a representation of the Cartesian product $A \times B = \{(a, b) : a \in A \wedge b \in B\}$ in set theory. As part of our abstraction away from set theory, we do not have the $\in$ notion. Thus, the above definition replaces the Cartesian product in set theory.

Similar to disjoint unions in set theory is the definition of coproduct in category theory:

Definition 5: a **coproduct** of two objects A and B is an object $A+B$ with two injection arrows $\iota_1 : A \to A+B$ and $\iota_2 : B \to A+B$ such that, for any triple $A \xleftarrow{f} C \xrightarrow{g} B$, there is a unique arrow $[f,g] : A+B \to C$ such that

$[f,g] \circ \iota_1 = f$ and $[f,g] \circ \iota_2 = g$.

In set theory, the disjoint union of two sets X, Y is $X+Y = \{1\} \times X \cup \{2\} \times Y$. In the category **set**, we illustrate the objects as below

$$X \xleftarrow{\iota_1} X+Y \xrightarrow{\iota_2} Y$$

$$\iota_1(x) = (1,x), \iota_2(y) = (2,y)$$

The new formed arrows are given as follows. Suppose that $f : X \to Z, g : Y \to Z,\ [f,g] : X+Y \to Z$.

$[f,g](1,x) = f(x)$, and $[f,g](2,y) = g(y)$.

From the definition of product and coproduct, we can generalize to the indexed family of products.

Definition 6: Let $\mathcal{C}$ be a category with a product, and A, B be objects in $\mathcal{C}$. An object B^A is an **exponential object** if there exists an arrow $\underset{AB}{eval} : (B^A \times A) \to B$ such that, given an object C and arrow $g : (C \times A) \to B$, there is exactly one arrow $curry(g) : C \to B^A$ such that

$$\underset{AB}{eval} \circ (curry(g) \times id_A) = g.$$

The exponential object is the representation of the functional space, or a collection, from A to B in set theory, i.e. $B^A = \{f : A \to B\}$ (or Π type in a type theory).

Definition 7: A **Cartesian closed category** (CCC) is a category with a terminal object, a product, and exponentiation.

These are the basic definitions, in category theory, for understanding monads. For other definitions, such as **limit, colimit, equalizer, pullback, pushout**, see [29].

The next section introduces the interpretation of simply typed λ-calculus in category theory.

3.1.1 Simply typed λ-calculus in category theory

Recent research in formal semantics, especially Montagovian grammar, is based on the λ-calculus in section 2.1.1. This section will show how we interpret the calculus in a Cartesian closed category, in order to establish a starting point for further research. Basically, the interpretation is as given in [29][p. 53-57] or [30, 29, 182]. Its potential further research, for example, is the study of Kripke's semantics in the sense of a linguistic model in category theory. [183] shows the translation between Kripke's semantics and the Cartesian closed subcategories of presheaves over a poset while the majority of formal semantics, which includes λ-calculus, is based on Kripke's possible-world semantics [180].

Basically, according to [29] the syntax for typed λ-calculus is given, by [3], as below

$$\boxed{M := unit|c|x|\lambda x : A.M|(M\ M)|(M, M)|\pi_1\ M|\pi_2\ M}$$

Table 3.2: Syntax of the typed λ-calculus

Where x is a variable that ranges over a set of variables, and c is a metavariable that ranges over a set of constants. The λ notion is the functional abstraction. $M\ M^{'}$ is the functional application. $(M, M^{'})$ is a pair with π_1 and π_2 being projection functions.

Besides the basic syntax, the treatment of free variables is given an additional step. The variable x is free in the formula x, and the free variable in the formula $\lambda x : A$. M is the set of free variables in M excluding x. Free variables in other formulae are the concatenation of free variables in their sub-formulae. The typing rules for the syntax are

$$\boxed{\begin{array}{c}
\Gamma \vdash unit : Unit \\[1em]
\Gamma \vdash c : B_c \\[1em]
\Gamma ; x : A \vdash x : A \\
\dfrac{\Gamma \vdash x' : A'}{\Gamma ; x : A \vdash x' : A'} \\
\dfrac{\Gamma ; x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \to B} \\
\dfrac{\Gamma \vdash M : C \to B \quad \Gamma \vdash M' : C}{\Gamma \vdash (M\ M') : B} \\
\dfrac{\Gamma \vdash M : B \quad \Gamma \vdash M' : B'}{\Gamma \vdash (M, M') : B \times B'} \\
\dfrac{\Gamma \vdash M : B \times B'}{\Gamma \vdash \pi_1 M : B} \\
\dfrac{\Gamma \vdash M : B \times B'}{\Gamma \vdash \pi_2 M : B'}
\end{array}}$$

Table 3.3: Recapturing typed λ-calculus in the type theory by [29].

Besides typing rules, there are additional equivalent rules to provide the reasoning in the languages. notable are β and η rules as

$$(\beta)(\lambda x : A.M)N = [N/x]M : B$$

$(\eta)(\lambda x : A.M\ x) = M : A \to B$

The β rule means that substituting x to N in M is equivalent to applying N to an abstract over M for a free variable x in M. Normally, the rule is associated with the **let** construction in programming languages. The η rule means that each element in a function type is equivalent to a λ abstraction.

The translation of the above type system to a Cartesian closed category $\mathcal{C}$ are given below. The main difficulty is to translate the substitution to $\mathcal{C}$. Consequently, the difficulty of the inverse direction is to define the λ abstraction. The translation is basic in the sense that we can complicate and equip the about type system with additional equality rules.

The typing translation is

$$\boxed{\begin{gathered}
\llbracket A \rrbracket = A_{\mathcal{C}}(A \in K) \\
\llbracket Unit \rrbracket = 1 \\
\llbracket A \times B \rrbracket = \llbracket A \rrbracket \times \llbracket B \rrbracket \\
\llbracket A \to B \rrbracket = \llbracket B \rrbracket^{\llbracket A \rrbracket}
\end{gathered}}$$

The context is translated into

$$\llbracket \varnothing \rrbracket = 1$$

$$\llbracket \Gamma; x : A \rrbracket = \llbracket \Gamma \rrbracket \times \llbracket A \rrbracket$$

The syntax is translated as follows

$$\llbracket \Gamma \vdash unit : Unit \rrbracket =! : \llbracket \Gamma \rrbracket \to \llbracket Unit \rrbracket$$

$$\llbracket \Gamma \vdash c : B_c \rrbracket = c \circ ! : \llbracket \Gamma \rrbracket \to \llbracket B_c \rrbracket$$

$$\llbracket \Gamma; x : A \vdash x : A \rrbracket = \pi_2 : (\llbracket \Gamma \rrbracket \times \llbracket A \rrbracket) \to \llbracket A \rrbracket$$

$$\llbracket \Gamma; x : A \vdash x^{'} : A^{'} \rrbracket = \llbracket \Gamma \vdash x^{'} : A^{'} \rrbracket \circ \pi_1 : (\llbracket \rrbracket \times \llbracket A \rrbracket) \to \llbracket A^{'} \rrbracket$$

$$\llbracket \Gamma \vdash \lambda x : A.M : A \to B \rrbracket = curry(\llbracket \Gamma, x : A \vdash M : B \rrbracket) : \llbracket \Gamma \rrbracket \to (\llbracket B \rrbracket^{\llbracket A \rrbracket)}$$

$$\llbracket \Gamma \vdash (M\ M^{'}) : B \rrbracket = eval_{CB} \circ \langle \Gamma \vdash M : C \to B, \llbracket \Gamma \vdash M^{'} : C \rrbracket \rangle : \llbracket \Gamma \rrbracket \to \llbracket B \rrbracket$$

$$\llbracket \Gamma \vdash (M, M^{'}) : B \times B^{'} \rrbracket = \langle \llbracket \Gamma \vdash M : B \rrbracket, \llbracket \Gamma \vdash M^{'} : B^{'} \rrbracket \rangle : \llbracket \Gamma \rrbracket \to \llbracket B \times B^{'} \rrbracket$$

$$\llbracket \Gamma \vdash \pi_1 M : B \rrbracket = \pi_1 \circ \llbracket \Gamma \vdash M : B \times B^{'} \rrbracket : \llbracket \Gamma \rrbracket \to \llbracket B \rrbracket$$

$$\llbracket \Gamma \vdash \pi_2 M : B \rrbracket = \pi_2 \circ \llbracket \Gamma \vdash M : B \times B^{'} \rrbracket : \llbracket \Gamma \rrbracket \to \llbracket B \rrbracket$$

Table 3.4: Typed λ-calculus in category theory.

[30] provided a better illustration, as below, by showing the analogy between logic and category theory.

Axiom	$\dfrac{}{\Gamma, A \vdash A}Id$	$\dfrac{}{\pi_2 : \Gamma \times A \longrightarrow A}$
Conjunction	$\dfrac{\Gamma \vdash A \, \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge I$ $\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge E_1$ $\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge E_2$	$\dfrac{f : \Gamma \longrightarrow A g : \Gamma \longrightarrow B}{\langle f, g \rangle : \Gamma \longrightarrow A \times B}$ $\dfrac{f : \Gamma \longrightarrow A \times B}{\pi_1 \circ f : \Gamma \longrightarrow A}$ $\dfrac{f : \Gamma \longrightarrow A \times B}{\pi_2 \circ f : \Gamma \longrightarrow B}$
Implication	$\dfrac{\Gamma; A \vdash B}{\Gamma \vdash A \supset B} \supset I$ $\dfrac{\Gamma \vdash A \supset B \, \Gamma \vdash A}{\Gamma \vdash B} \supset E$	$\dfrac{f : \Gamma \times A \longrightarrow B}{} \Lambda(f) : \Gamma \longrightarrow (A \Longrightarrow B)$ $\dfrac{f : \Gamma \longrightarrow (A \Longrightarrow B) \ g : \Gamma \longrightarrow A}{ev_{A,B} \circ \langle f, g \rangle : \Gamma \longrightarrow B}$

Table 3.5: Illustrated duality between logics and category theory.

The Λ notion in implication rules means the abstract type, which is an abstract for type, whereas an abstract for terms is the normal λ notion.

We can extend this basic interpretation further. [8, 184], for example, showed that monads (to be discussed later in section 6) in category theory can capture an extension of typed λ-calculus with effects. An extension of monads is given in Chapter 5. The intuitive idea in the logical sense, in the author's opinion, is the translation between the propositional universe and the hypothetical universe. The propositional universe accommodates pure calculations in functional programming. The hypothetical universe, in contrast, includes modality, conjectures, questions, or effects. The monadic laws tie up the interaction between the two universes, particularly via Kleisli's triple. This view is also taken in, for example, logic-enriched type theory [131].

3.2 The definition of functors

Definition 7: Let $\mathcal{C}$, $\mathcal{D}$ be categories. A **Functor**[2] $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ is defined as follows:

1) if A is an object of $\mathcal{C}$, $\mathcal{F}(A)$ is an object of $\mathcal{D}$.

2) if f is a $\mathcal{C}$ arrow with $f : A \to B$ then $\mathcal{F}(f) : \mathcal{F}(A) \to \mathcal{F}(B)$ with additional properties. For all A in $\mathbf{Obj}(\mathcal{C})$, and composable arrows f, g in $\mathbf{Arr}(\mathcal{C})$,

- $\mathcal{F}(id_A) = id_{\mathcal{F}(A)}$
- $\mathcal{F}(g \circ f) = \mathcal{F}(g) \circ \mathcal{F}(f)$

If $\mathcal{C}$, $\mathcal{D}$ is the same, $\mathcal{F}$ is called an *endofunctor*. There are various examples of functors; [30][p. 26–27] give examples of $\mathcal{F} : G \to \textbf{set}$, where G is a group and this functor denotes an action of G on a set. If P is a poset which represent time, a functor $\mathcal{F} : P \to \textbf{Set}$ can be used to describe the *Kripke*'s semantics. Another example is a functor, $list : \textbf{set} \to \textbf{set}$. If A is an object in **set**, $list(A)$ is a collection of finite elements in A. If an arrow $f : A \to B$ is a function which operate on the set A (such as sorting) then

$list(f) : list(A) \to list(B)$ and

$list(f)[x_1, \cdots, x_n] = [f(x_1), \cdots, f(x_n)]$.

[2] According to [29], the name is taken from [105].

3.3 The definition of a natural transformation

Functors, in the previous section, show relations between categories. Natural transformation is the relation between functors. Intuitively, from the logical point of view, a functor is the higher-order logic, i.e. reasoning with arrows of a category. An example of a functor in linguistic is the quantifier $\forall$. Thus, we can use it to express an axiom: $\forall x : A.f(x)$ for A and f are an object and arrow, respectively, in $\mathcal{C}$.

Hence, a natural transformation is used to write an inference statement such as $\forall x : A.f(x) \Rightarrow \forall y : B.g(y)$ for x, y, f, g being objects and arrows in $\mathcal{C}$. Formally, the transformation is defined as below.

Definition 8: let $\mathcal{C}, \mathcal{D}$ be categories. F,G are functors from $\mathcal{C}$ to $\mathcal{D}$. A natural transformation η from $\mathcal{F}$ to $\mathcal{G}$ ($\eta : \mathcal{F} \to \mathcal{G}$) is a function: if A is the object of $\mathcal{C}$, there exists a $\mathcal{D}$ arrow $\eta_A : \mathcal{F}(A) \to \mathcal{G}(A)$ such that, for any $\mathcal{C}$ arrow $f : A \to B$, the following diagram commutes in $\mathcal{D}$

$$\begin{tikzcd}
\mathcal{F}(A) \arrow[r, "\eta_A"] \arrow[d, "\mathcal{F}(f)"'] & \mathcal{G}(A) \arrow[d, "\mathcal{G}(f)"] \\
\mathcal{F}(B) \arrow[r, "\eta_B"] & \mathcal{G}(B)
\end{tikzcd}$$

for example, for any functor F, the identity natural transformation $\iota : F \to F$ are identity arrows, i.e. $\iota_A = id_{F(A)}$.

Another example is the natural transformation reverse of a functor $List : \mathbf{Set} \to \mathbf{Set}$, which is defined as:

$$reverse : List \to List$$

$$reverse_X : List(X) \to List(X) := [x_1, \cdots, x_n] \Rightarrow [x_n, \cdots, x_1]$$

3.4 An introduction to monads

The mathematical definition of monads is given below.

Definition 9: A **monad** over a category C is a triple (T, η, μ) where T is an endofunctor, and

$$\begin{cases} \eta : Id_C \to T \\ \mu : T^2 \to T \end{cases}$$

are natural transformations, and the diagrams below commute:

$$\begin{tikzcd}
T \arrow[r, "\eta_T"] \arrow[dr, "id_T"'] & T^2 \arrow[d, "\mu"] & T \arrow[l, "T(\eta)"'] \arrow[dl, "id_T"] \\
 & T &
\end{tikzcd}$$

η, μ are called the unit and multiplication of the monad, respectively. Assuming familiarity with functional programming (such as [185]), we can follow the standard practice of using the Kleisli triple [33][p. 45] to interpret the above monads. We follow [20] to provide a simpler interpretation of monads. A monad is a monoid. Monoids meet these two criteria:

1. Monoids combine with other objects of the same kind to return an object of the same kind. This principle is quite useful for modelling the compositional rules in linguistic semantics.
2. There exists a unit object so that, if we combine an object α with the unit object, we will still have α. For example, in basic arithmetic, we have a unit object 0 for the $+$ operator and a unit object 1 for the $\times$ operator.

In addition, the monads require:

1. The functor that will create or associate new types from original types and the connections between new types that correspond to the old relations.
2. A unit operation that lifts the actual values of the original types to their images under the functor operator.
3. A bind operation that performs the composition rules.

Thus, we write the monad as a Kleisli triple $\langle \mathbb{M}, \eta, \star \rangle$ where $\mathbb{M}$ is the functor, η is the unit and $\star$ is the binding operator.[3]

η has a formation $\forall a.a \rightarrow \mathbb{M}(a)$.

[3] In [17], the $\star$ operator is represented by the $\multimap$ operator.

$\star$ has a formation $\forall a.\forall b.\mathbb{M}(a) \to (a \to \mathbb{M}(b)) \to \mathbb{M}(b)$.

In addition, further rules for a proper monad, which is equivalent to the commute diagrams, are

$\eta(x) \star f = f(x)$

$m \star \eta = m$

$(m \star f) \star g = m \star (\lambda x.f(x) \star g)$

The third monad requirement above introduces the binding operator, which needs an associative property in $\star$. The second requirement makes sure that η has no interface to functions, and the first requirement lifts up functional application from old functions to a new binding functor.

3.5 The distinction between mathematical definitions and an implementation of monads

The above definitions come from category theories in mathematics. In computing perspective, the original application idea comes from [8], who realized that monads can be used to capture effects in functional programming. Hence, various effects have been studied by researchers such as [186, 9] and [187]. In addition, the contemporary overview of monads and their applications in programming languages and industries is given in [188].

In the author's opinion, the computational implementation of monads yields merit and further insights beyond their theoretical studies. For example, [6] shows that an evaluation order provides a further insight into the analysis of quantifications in monads beyond that obtained through traditional static analysis of logics in typed logical grammar. According to [189], other advantages of implementing monads outside evaluation order that are worth exploring are sugar syntax, reuseables, and encoding more non-standard computations.

This idea is also found in [8], distinguishing between a program and a proof. A proof is a mathematical construction and a program is a computer construction. By the Curry–Howard correspondence, they are equivalent. However, as Moggi noted, they are different in practice. This idea and its potential implications, for example, is recently confirmed by research in neuroscience by [190].

3.6 Notable monads in computing

In functional programming, we concretise the ideas of objects and arrows in category theories as data types and functions over these data types. Hence, we can construct complex operators and real world applications from those data types, functions, and operators. Furthermore, the monads are interpreted as abstract datatypes. A general study of functional programming language is provided in [185], a particular computer science application of category theory to natural languages can be found in [25, 26].

Despite the fact that [8] is the first author to introduce the notion of monads in computing, [186] is the first author to substantiate the mathematical notion into an implementation with substantial examples with impure properties in the design of programming languages. Let us illustrate the basic examples of monads by following [33, 186, 50].

3.6.1 The maybe monad

The **partiality** or **maybe monad** is defined as

M A = A $+\{\perp\}(i.eA_{\perp})$

$$\begin{cases} \eta_A = A \xrightarrow{\iota_1} A + \perp \\ \text{If } f : A \to \mathbb{M}B, f \star \perp = \perp, f \star a = f(a)\text{for} a \in A \end{cases}$$

3.6.2 The nondeterminism monad

$\mathbb{M}A = \mathcal{P}_{fin}(A)$, i.e the finite power set of A.

$$\begin{cases} \eta A = a \Rightarrow \{a\}, \text{i.e a map from an element to a set of that element.} \\ \text{if } f : A \to \mathbb{M}B, c \in \mathbb{M}A, f \star c = \cup\{f(x)|x \in c\} \end{cases}$$

3.6.3 side-effects or the state monad

$\mathbb{M}A = (A \times S)^S$, for s is a stack or state of a computer.

$$\begin{cases} \eta_A = a \Rightarrow \lambda s : S.(a, s) \\ \text{if } f : A \to \mathbb{M}B, c \in \mathbb{M}A, f \star c = \lambda s : S.fa(\pi_2(cs)) \end{cases} \quad \text{(i.e}$$

$do(a, s^{'}) \leftarrow cs$

$f\ a\ s^{'})$

The **Writer** and **Reader** monads are special cases of the **State** monad. A further discussion is given by [186, 9, 191]

3.6.4 The exception monad

$\mathbb{M}A = A + E.$

$$\begin{cases} \eta_A = \eta_A = A \xrightarrow{\iota_1} A + E \\ \text{if } f : A \to \mathbb{M}B, f \star (\iota_2 e) = \iota_2 e, (e \in E), f \star (\iota_1 a) = f(a), (a \in A) \end{cases}$$

3.6.5 The continuation monad

$\mathbb{M}A = \Omega^{\Omega^A}$

$$\begin{cases} \eta_A a = \lambda k : \Omega^A.k\ a \\ \text{if } f : A \to \mathbb{M}B, c \in \mathbb{M}A, f \star c = \lambda k : \Omega^B.c(\lambda a : A.f\ a\ k) \end{cases}$$

3.6.6 IO monads

For the operational semantics of IO monads, see [187, 192]. The denotational semantics is given below, from [33]. The underlying idea is to use the recursive functions (denoted by the fixed point operator μ), represented by trees, to model the sequent of an input monad.

Input monad $\mathbb{M}A = \mu X.A + X^{\mathbb{U}}$[4]

$\eta_A : a$ to a tree with only one leaf labelled with a.

if $f : A \to \mathbb{M}B, c \in \mathbb{M}A, f \star c$ is the tree which replaces leaves of c labelling by a to f a.

Output monad $\mathbb{M}A = \mu X.A + (\mathbb{U} \times X)$

η_A is a map $a \to (\epsilon, a)$

if$f : A \to \mathbb{M}B, f \star (s, a) = (s; s^{'}, b)$ where $fa = (s^{'}, b)$ and $s; s^{'}$ is a concatenation of s and $s^{'}$

In functional programming, the input–output (IO) monad is implemented as by [187, 38] in the programming language, Haskell. IO provides an additional dimension to capture the non-mathematical computations of users' input and output from computers. It separates the class of computation from a pure or mathematical computation, and from impure or physical information such as from users' input and output devices. According to [193], IO is designed to capture the imperative function from imperative programming languages, like C, for functional programming.

[4] U is a universe

An *IO a* is a *computation* that may perform in an *IO* functor, and return a result of type a. In Haskell, the η function is replaced by a *return* notation and $\star$ is replaced by a $\triangleright$ notation.

$return :: a \rightarrow IO\, a$ (i.e an η operator)

$\triangleright$ :: $IO\, a\rightarrow$ (a → IO b) → IOb (i.e an $\star$ operator)

the *return* function lifts a value into an *IO* monad environment and the $\triangleright$ operator performs the computation which is related to its first argument and passes the results to its second argument. We can define other functions from these two primitive functions in Haskell.

For example, the *main* function, a main function that relates each Haskell program to a computer's hardware, is defined as $IO()$ which means a computation that returns nothing.

$\gg$ :: *IO* a → *IO* b → *IO* b

p $\gg$ q = p $\triangleright$ $\lambda x.q$

The first of these lines states that the function $\gg$ takes two arguments. The first argument has a type *IO* a, and the second has a type *IO* b, and the computation returns a value of type *IO* b. Implicitly, if two arguments p and q are given in a computation, i.e. $p \gg q$, then we use the $\triangleright$ computation to calculate it. The left hand of $\triangleright$, i.e. p, is given in the computation (as equal to IO a), and the right hand of $\triangleright$ specify how we compute it (as a → IO b). $\lambda x.q$ means a function that for any given value x, we still get q, i.e the value of type $IO\,b$. Therefore, the $\gg$ operator means that we compute only the first argument, and return only the second argument. Though the first argument is computed, we discard the return value of this computation.

Monads are further discussed in [50, 38].

3.6.7 An example of the reader monad

The above sections show the usefulness of the applications of monads from the pure mathematical notion in category theory to model the effects in the functional programming. In the author's opinion, the main advantage of the category theory is its capability of interpret λ calculus as its internal language. The research by [194] shows the application of the reader monad in the linguistic field. The reader monad $\langle R, \eta, \star \rangle$ associates each value, or a piece of data, with a specific location in memory, the location being indexed by an integer. More explicitly, this monad is a function that, given a location i, returns the value a stored in that location. In short, it is a function of a type $\iota \to \alpha$ with additional properties of η and $\star$ operators.

R stands for the reader monad that will read a memory location to get a value or stored data. R lifts every element of a type τ to element of a type

$\iota \to \tau$, i.e. R is a function from indexes ι to objects of type τ. It operates in the same manner as in Montague's semantics, where we lift entities to their intentional interpretations.

For each function f or a relation between two types α, β $f : \alpha \to \beta$, R maps them to $R(f) : R\,\alpha \to R\,\beta$. In our case, $R(f) : (\iota \to \alpha) \to (\iota \to \beta)$

As $f : \alpha \to \beta$, or f's first argument has a type α, we can rewrite R(f) as:

$R(f) = \lambda\theta.f \circ \theta$

As θ is a function from ι to α, or $\theta : \iota \to \alpha$, f is a function from α to β. Thus, $f \circ \theta = \lambda i.f(\theta(i))$. Hence:

$R(f) = \lambda\theta.\lambda i.f(\theta(i))$.

The unit operator in our monad R lifts a value a of type α to a function f of a type $\iota \to \alpha$ that associates every index i of type ι to the fixed value, a. Thus it is a constant function and we can write it as:

$\eta(x) = \lambda i.x$

The above notion means that, if we pass a parameter x to η, and if we are given a further value i, then the function still returns x.

The $\star$ or binding function operates as a functional application:

if $m : R\,\alpha$, $f : \alpha \to R\,\beta$ then

$$m \star f = \lambda i.f(m(i))(i)$$

which is $R\,\beta$. As $m : R\,\alpha$, i.e. $m : \iota \to \alpha$, $m(i) : \alpha$. as $f : \alpha \to R\,\beta$, $f(m(i)) : R\,\beta$. Thus $m \star f : R\,\beta$, or $m \star f : \iota \to \beta$. In other words, if we introduce an additional variable i, it is the function, $f(m(i))$, that returns a value of type β with i as an input.

3.6.8 Other monads

A range of monads have been research besides those listed above, for example, the list monad [186], the probability monad [195, 196] with a pioneering approach to probability in [197], the completion monad [198], the update monad [199]. Its potential applications in linguistics is given in the discussion section 4.5.

3.7 Discussion

This chapter provides the mathematical definitions of monads. The next chapter shows its applications in linguistics and the subsequent chapters extend the monads to the parameterized monads and its linguistic applications.

A further general study of category theory is available in [200, 31, 201]. However, despite the powerful expressive of monads, it should be noted that combining monads is a difficult topic, as discussed in [21, 6, 202].

There are benefits of using category theory in computing which the author also found useful in linguistics. According to [174], category theory provides a pure theory of function, giving a powerful guide to research direction. Dybjer also states that categorical properties are essential if they exist, and that the theory studies objects by characterizing universal properties rather than by description. According to [30], category theory provides composition, abstraction, and independence of representation. Furthermore, the theory opens a new foundation for mathematics and science as an alternative for set theory.

Chapter 4

Monads in linguistics

This chapter provides a summary of applications in linguistics of the previous chapter's monad definitions. In this dissertation, I interpret the effects in the computing sense, as in [6, 33], as contextually related phenomena in linguistics. Hence, monads, in accordance with [21, 17, 16], is a moderate contextual theory in *pragmatic enriched theories* in the sense of [41]. According to [41], there are three contextualism approaches. I characterize existence frameworks as follows.[1] Firstly, the minimal contextualism approach assigns a context to a minimal set of words such as pronouns, demonstratives or indexicals. It is mostly identical with the static or logical semantics approach to natural languages. Theories of this kind include [86, 112] or [203]. It's strengths are in its foundational background and representation analysis. However, in order to keep theoretical analysis precise, researchers are currently working on a small number of linguistic examples.

In the second approach, radical contextualism in [41], one assigns a context

[1] The characterization is based on the framework's strength because it is not quite straightforward: each framework can interpret others based on the development of the underlying theories, or one theory may have many frameworks.

to every word. Notable theories are [56, 47, 111]. Radical contextualism offers strength in explanation, but is weak on foundational issues. For example, situation semantics [47] is based on possible-worlds semantics [180], which is based, in turn, on set theory. However, the set-theoretic foundation is criticised in [28], for example, in the case of the difficulty to use the theory to formalize the plurality in [204].

Finally, there is a moderate contextualism, richer than the minimal approach, but less contextual than the radical approach. It extends the minimal to overcome its rigidity while still preserving representation analysis. Applying monadic frameworks to natural languages is considered to be a moderate contextualism approach. For example, according to [45], their concern is the semantic contexts, or with local context, rather than the whole information states of utterance contexts. The theories of this kind are this framework and others in [27, 21, 10, 28, 12, 16, 11]. Moderate contextualism is also moderate in the sense that we interpret the verb phrases in the dynamic tradition, for example, as tests on the context only. In the research framework in [28], it also means that a proposition doesn't require a contextual analysis. A non-proposition, however, does.

Using monads, as a particular instance of category theory, in linguistics yields observable advantages. Firstly, monads are abstract and expressive enough to formalize and modularize other phenomena under the same algebraic structure. It inherits the abstract approach from category theory; thus, it overcomes the technical misinterpretation that occurs, for example, under set theory. The modularization advantage has been claimed by [6].

In this dissertation, using monads also provides a denotational semantics, which allows further processing for verifying or proving the correctness of linguistic properties, for the interpretation of a linguistic phenomenon. For example, a property is an assertion, e.g. that the definite description **the** does not go after the word **there** [159]. Another example is the coreference problem between the *morning star* and the *everning star* in [62]. In a particular case, the research by [11] shows that monads provide a denotational semantics for intentionality. In addition, monads provide a common framework, beside others, for the interpretation of linguistics phenomena. Grasping phenomena in a single framework helps others to study multiple phenomena more easily. Hence, applying monads can lead to research principles which will be useful for future implementation on computing and linguistic phenomena.

Using monads in linguistics was pioneered in [63] and further developed in [17, 15, 21, 16, 122, 205]. A general summary of previous research on monads is given as below by extending [11] with recent research on monads since 2009.

Monadic types	**Formation**	**Linguistic phenomena**	**Researchers**
State.set monad	Monad morphism	Exceptional scope taking, indefinite	[17]
Writer monad	$\tau \mapsto \langle \tau, 1 \rangle$	Conventional implicature	[20]
Reader monad	Special case of the state monad	Generalize opacity	[194]
Probability monad	probability monad	Conjunction fallacies	[206]
Exponential	$\tau \rightarrow \_$	Variable binding	[4]
Exponential		Intentionality	[4]
Product	$\tau \times \_$	Variable binding	[11]
State	$\tau \rightarrow \langle \tau \times \_ \rangle$	Variable binding	[11]
Power set	$\{X \mid X \subseteq \_\}$	Quantification	[11]
Power set		Interrogatives	[207]
Pointed powerset	$\{\langle x, X \rangle \mid x \in X \subseteq \_\}$	Focus	[208]
Sum	$\tau + \_$	Presupposition	[11]
Continuation	$(\_ \rightarrow \tau) \rightarrow \tau$	Interrogative	[11]

Continuation	Type lifting	Quantification
		
		generalized quantifiers
		donkey anaphora
		grammar, conjoinable coordinate
		(as delimited control)
		crossover, superiority,
		donkey anaphora, evaluation order
		negative polarity licensing
			[4];
			[165];
			[149, 46, 164];
			;
			[166];
			[6]
			[209, 96, 45, 12]

General/not specify		Compositional treatment of anaphora	[70]
typical monad	state monad	Dynamic semantics	[15]
Continuation	Practical perspective	Anaphora resolution	[210]
Internal monad (composable continuation)	A sub-language of the meta lambda calculus	Non-determinism Contextual parameters Delimited continuation Covert movement Focus Inverse scope	[211] as above as above [212] as above as above
effects and handlers	Define the calculus	Deixis Quantification Conventional implicature	[213]
graded monads	monad	presupposition	[16]

Table 4.1: Summary of contemporary research of monads in linguistics

4.1 A basic linguistic example

Let us consider some linguistic examples of how we use binding to perform composition in the monad of propositional logic, using the concepts as introduced into linguistics in, for example, [108, 106, 44, 4]. We use the monad $\mathbb{P}$ for the proposition space which is modelled as a set-theoretic interpretation of possible-world semantics in [180]. Hence, the lifting is a translation from a linguistic expression to its possible-world semantics, i.e. a set. For example, we lift entities such as the set of individuals, *John*, and *Mary* to their logical constants J^*, M^*. The transitive verbs, such as *like*, express the relations between entities and truth values. The binding $\star$ is just the functional application, as is standard in formal semantics.

$\eta(John) : \mathbb{P}\, e$ (for the convenience, we call $\eta(John)$ as J^*),

$\eta(Mary) : \mathbb{P}\, e$ (i.e M^*),

$likes : \mathbb{P}\, e \to (e \to t)$. The type also equals $\lambda e.\mathbb{P}\,(e \to t)$ by the additional monadic associative law.

Therefore, the phrase, *John likes* is lifted to our monad with the binding operator as $J^* \star likes$ with $J^* : \mathbb{P}\, e$ and $likes : e \to \mathbb{P}\,(e \to t)$. Thus $J^* \star likes : \mathbb{P}\,(e \to t)$.

Let us look at the complete sentence, $John\, likes\, Mary$. If we compose the sentence by reading it from left to right as $(John\, likes)\, Mary$, it has the following monadic interpretation
$J^* \star likes \star M^*$.

From the above explanation, this is equal to

$\mathbb{P}\,(e \to t) \star M^*$.

If we assume that we can swap two parameters in the binding without changing the meaning, it also equals

$M^* \star \mathbb{P}\,(e \to t)$

We should note that, from the associative law $\mathbb{P}\,(e \to t)$ is equal to $\lambda e.\mathbb{P}\,t$. Thus, $M^* \star \mathbb{P}\,(e \to t)$ is equal to $M^* \star (\lambda e.\mathbb{P}\,t)$. By our binding definition, this latter returns $\mathbb{P}\,t$. Therefore, the sentence, $John\, likes\, Mary$ is computed to $\mathbb{P}\,t$, i.e. a truth proposition.

In the above example, we assume that the evaluation order we compute is from left to right. Other frameworks, such as Lambek's calculus [114], may explicitly specify what the left and right parameters of the computation are. This specification can be needed because some natural languages do not have any word order, i.e. the words distribute freely from left to right. Languages with free word-order include Russian and Japanese, whereas strict word-order occurs in English. Thus *John likes Mary*, or *Mary likes John*, or *John Mary likes* have the same meaning in Russian or Japanese. They have a distinct meanings, however, in English. More clearly, the former sentence is a correct grammatical sentence while the latter is an incorrect one.

4.2 The continuation for the scope problem and quantifications

4.2.1 The continuation in linguistics

The basic idea of continuation is expressed in Section 3.6 with details in [6, 9], and a composable continuation[2] is indeed a metavariable, as mentioned in the Section 2.2.3. The idea of continuation also appears in other research fields. To begin, let us consider an example of a double-negation rule in classical mathematical logic: $\neg(\neg A) = A$ [214]. This rule is usually taken for granted when constructing a mathematical proof by contradiction (*reductio ad impossibile*). Instead of proving a problem directly, you suppose its conclusion to be false, and you draw from the false premises to a contradiction, hence proving the problem.

In computing, 'the continuation' means the remainder of a program or

[2] The global context is sliding to a list of compositional local contexts, and each local context is called composable continuation in [7].

future computation. Continuation is used to provide semantics for the complex statements which contain a recursive expression or **goto** and **jump** commands. A survey of the discovery of this terminology in computing is given in [215]. In natural language semantics, the continuation is the type-shifting in [4, 216, 149]. In general, the basic idea of continuation is to shift the interpretation of an expression's meaning from its syntactic analysis to its associated final objective such as a true or false proof in mathematics, final state in computing, or truth values in linguistic semantics.

In this dissertation, we use the notion of composable continuation from [217, 9, 75, 6, 23]. For a detailed discussion of continuation, see [218], and, for a characterization, [219]. An intuitive explanation of continuation in linguistics is in [220, 166, 149], with more elaborate explanation in [12, 17].

According to [166, 6], the continuation denotation of an expression consists of two steps

1. continuation: transforms a value to intermediate answer, and
2. continuized denotation: transforms continuation to an intermediate answer.

Let us illustrate a simple Montagovian grammar from [166] (without categorial grammar).

Category	Syntax	Direct semantics	direct denotation	semantic type	logical meaning
S	$NP\,VP$	$[\![VP]\!]([\![NP]\!])$	$[\![S]\!]$	t	Boolean value
VP	$Vt\,NP$	$[\![Vt]\!]([\![NP]\!])$	$[\![VP]\!]$	$\langle e,t\rangle$	property
VP	run	$\lambda x.\mathbf{run(x)}$	$[\![VP]\!]$	$\langle e,t\rangle$	property
NP	Alice	$\mathbf{A}$	$[\![NP]\!]$	e	entity
NP	Bob	$\mathbf{B}$	$[\![NP]\!]$	e	entity
Vt	like	$\lambda x.\lambda y.\mathbf{like(x,y)}$	$[\![NP]\!]$	$\langle e,\langle e,t\rangle\rangle$	relation on entities

The variable, as in traditional logics, are

Variable	Type	Meaning
p, q	t	propositions
x, y, z	e	individual constants
P, Q	$\langle e,t\rangle$	one-place predicates
R, S	$\langle e,\langle e,t\rangle\rangle$	two-places predicates

The sentence has semantic construction in Montagovian grammar as usual. For example, *Alice runs.* or *Alice likes Bob* is parsed as **run j**, **like A B**.

The **continuation** of an expression is a function from the expression to the results of the entire meaning (or computation). The **continuized denotation** is a function from the continuation to the entire meaning. Intuitively, the continuation lifts an expression to a function, and the continuized denotation states the results of applying the continuation function. For example, the above grammatical expression has the following continuation:

Category	Continuation Type	Meaning
C_S	$\langle t, t\rangle$	sentence continuation, discourse meaning
C_{NP}	$\langle e, t\rangle$	NP continuation, result in a sentence truth value
C_{VP}	$\langle e, \langle t, t\rangle\rangle$	VP continuation, results is a sentence truth value
C_{Vt}	$\langle\langle\langle e, \langle e, t\rangle, t\rangle, t\rangle$	transitive verb continuation

In general, a continuation is a function from a category to a meaning of a sentence, i.e. a truth value. Thus, the continuized semantics is a function from a continuation to a meaning of a sentence, too (i.e. how the truth value is changed by applying the continuation). Thus, the above expressions have the continuized grammar below:

Category	Continuatized Type	Meaning
$\overline{S}$	$\langle\langle t, t\rangle, t\rangle$	continuized sentence
$\overline{NP}$	$\langle\langle e, t\rangle, t\rangle$	continuized NP
$\overline{VP}$	$\langle e, \langle e, t\rangle\rangle, t\rangle$	continuized VP
$\overline{Vt}$	$\langle\langle\langle e, \langle e, t\rangle, t\rangle, t\rangle, t\rangle$	continuized transitive verb

In a similar manner, the denotation of the λ-calculus in [3] also has the following continuized denotation in [221, 149]

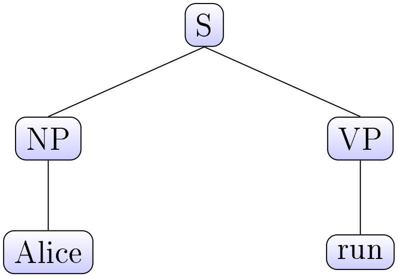
(1)	$\overline{c} = \lambda k.k\ c$
(2)	$\overline{x} = \lambda k.k\ x$
(3)	$\overline{\lambda x.M} = \lambda k.k\ (\lambda x.\overline{M})$
(4)	$\overline{M\ N} = \lambda k.\overline{M}(\lambda m.\overline{N}(\lambda n.m\ n\ k))$

Table 4.2: Continuation semantics for untyped λ-calculus terms.

Applying this transform style to the above grammar, we have the below denotation from [166]

Category	Syntax	Continuized semantincs	rule
S	$NP\,VP$	$\lambda c_S.\overline{VP}(\lambda P.\overline{NP}(\lambda x.c_S(P(x)))$	(4)
NP	Alice	$\lambda c_{NP}(\mathbf{A})$	(1)
VP	$Vt\,NP$	$\lambda c_{VP}.\overline{NP}(\lambda x.\overline{Vt}(\lambda R.c_{VP}(R(x))))$	(4)
NP	Bob	$\lambda c_{NP}.c_{NP}(\mathbf{B})$	(1)
VP	run	$\lambda c_{Vt}.(c_{Vt}.\lambda x.\mathbf{run}(x))$	(3)
Vt	like	$\lambda c_{Vt}.c_{Vt}(\lambda x.\lambda y.\mathbf{like(y,x)})$	(3)

Thus, the simple sentence *Alice runs* is derived grammatically as follows:



with the following semantics as in [166] :

$S = \lambda c_S.\overline{VP}(\lambda P.\overline{NP}(\lambda x.c_S(P(x))))(\text{rule 4})$

$= \lambda c_S.\overline{VP}\lambda P.(\lambda c_{NP}.c_{NP}(\mathbf{A}))(\lambda x.c_S(P(x)))(\text{rule 1})$

$= \lambda c_S.\overline{VP}\lambda P.c_S(P(\lambda c_{NP}.c_{NP}(\mathbf{A})))$

$= \lambda c_S.(\lambda c_{VP}.c_{VP}(\lambda x.\mathbf{run}(x)))\lambda P.c_S(P(\lambda c_{NP}.c_{NP}(\mathbf{A})))(\text{rule 3})$

$= \lambda c_S.c_S((\lambda c_{VP}.c_{VP}(\lambda x.\mathbf{run}(x)))(\lambda c_{NP}.c_{NP}(\mathbf{A})))$

$= \lambda c_S.c_S(\lambda c_{NP}.c_{NP}(\mathbf{A}))(\lambda x.\mathbf{run}x)$

$= \lambda c_S.c_S(\lambda x.\mathbf{run}(x))(\mathbf{A})$

$= \lambda c_S.c_S(\mathbf{run(A)})$

Thus, the above sentence returns a continuation function. To return a value, we usually have to use the evaluation function *eval* as in [9]. Normally, the evaluation function is the application of the continuation function to the identity function, i.e. $\lambda p.p$. Thus

$$
\begin{aligned}
&eval(\lambda c_S.c_S(\mathbf{run(A)})) \\
= \quad & (\lambda c_S.c_S(\mathbf{run(A)}))(\lambda p.p) \\
= \quad & (\lambda p.p)(\mathbf{run(A)}) \\
= \quad & \mathbf{run(A)}
\end{aligned}
$$

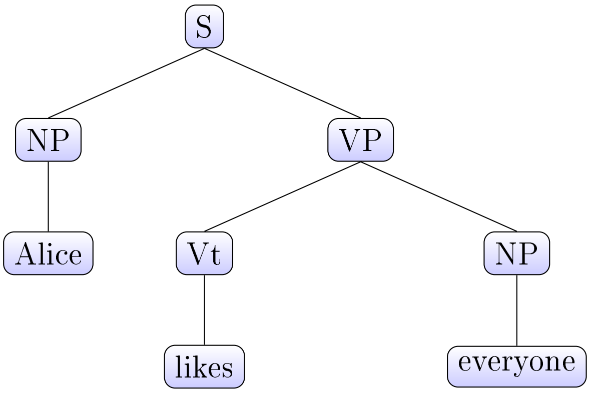
4.2.1.1 Quantifications

A brief overview of quantification is given in section 2.2.1. The continuation approach to quantifications has been studied in [220, 149, 166, 6, 75, 12, 96, 11]. The basic idea is that, in order to analyze, a quantification expression such as **every, some, most, no**, the direct grammatical rules are not enough, so we have to interact with the contexts to explain the expression's meaning. This approach finds support also from cognitive research [222], which interprets quantification as a numeric sense rather than linguistic one. The context, then, is regarded as the end-results of continuation. Thus, [166, 25], for example, explain the quantifications as

Everyone	$\lambda \overline{x}.(\forall c.(\mathbf{person}(c) \rightarrow \overline{x}(c)))$
Someone	$\lambda \overline{x}.(\exists c.(\mathbf{person}(c) \wedge \overline{x}(c)))$
Every	$\lambda \overline{P}\overline{x}.\overline{P}(\lambda P.\forall x.P\ (x) \rightarrow \overline{x}(x))$
A, Some	$\lambda \overline{P}\overline{x}.\overline{P}(\lambda P.\exists x.P(x) \wedge \overline{x}(x))$
Most	$\lambda \overline{P}\overline{x}.\overline{P}(\lambda P.\mathbf{most}(P)(\overline{x}))$
No	$\lambda \overline{P}\overline{x}.\overline{P}(\lambda P.\neg\exists x.P(x) \wedge \overline{x}(x))$
The	$\lambda \overline{P}\overline{x}.\overline{P}(\lambda Q.\iota x.(Q(x) \wedge \overline{x}(x)))$

$\forall, \exists, \rightarrow, \neg$ are basic logical connectives. The quantifications do not have a direct semantic interpretation, i.e. a context-free grammatical interpretation, since that requires an additional observational verification that the property $\overline{x}(c)$ holds for all, or at least that there exists one entity for which the property holds. Particularly, the semantic interpretation can be found in the literature as the type raising in by [4, 216] or the μ-calculus in [149].

We can derive the semantics for the sentence, *Alice likes everyone*, as follows



We start from the VP $\overline{likes}\overline{everyone}$

$= \lambda k.(\overline{everyone}(\lambda n.\overline{like}(\lambda m.(k(m\,n)))))(\text{rule } 4)$

$= \lambda k.(\overline{everyone}(\lambda n.(\lambda k'.k'\ \mathbf{like})(\lambda m.(k(m\,n)))))$

$= \lambda k.\overline{everyone}(\lambda n.(k(m\,n)(\mathbf{like})))$

$= \lambda k.\overline{everyone}(\lambda n.(k\mathbf{like}\,n))$

$= \lambda k.(\lambda k'.\forall x.\mathbf{person}(x) \to k'(x))(\lambda n.(k(\mathbf{like}\,n)))$

$= \lambda k.\forall x.(\mathbf{person}(x) \to k(\mathbf{like}\,x))$

Thus the sentence *Alice likes everyone* has the following denotation:

$\lambda k.\overline{Alice}(\lambda n.\overline{likes\ everyone}(\lambda m.k(m\,n)))$

$= \lambda k.\overline{Alice}(\lambda n.\lambda k'.\forall x.\mathbf{person}(x) \to (k'\mathbf{like}\,x)(\lambda m.(k(m\,n))))$

$= \lambda k.\overline{Alice}(\lambda n.\forall x.(\mathbf{person}(x)) \to k((\mathbf{like}\,x)n))$

$= \lambda k.(\lambda k'.k'\ \mathbf{A})(\lambda n.\forall x.(\mathbf{person}(x)) \to k((\mathbf{like}\,x)n))$

$= \lambda k.\forall x.(\mathbf{Person}(x) \to (k(\mathbf{like}\,x)\mathbf{Alice}))$

4.2.1.2 The continuation for an evaluation order

A general discussion of scope taking is in Section 2.2.2. This section highlights an advantage of the continuation approach in its ability to vary the scope of quantifiers, and hence also the evaluation order, in linguistics. Thus, this technique resolves some aspects of scope ambiguity.[3]. Basically, the technique arises from the Plotkin's rule (4) about the substitution rule in

[3] In computing, the scope ambiguity is usually associated with evaluation strategy by [221]

λ-calculus [149, 166, 6]. The substitution rule can have two reading strategies:

$(*)\overline{M\ N} = \lambda k.\overline{M}(\lambda m.\overline{N}(\lambda n.m\,n\,k))$

or

$(**)\overline{M\ N} = \lambda k.\overline{N}(\lambda n.\overline{M}(\lambda m.n\,m\,k))$

Each reading is associated with one scope interpretation. For example, consider the sentence, *Everyone helps someone* from [25][p. 299]. Applying a similar procedure as in the above sentence produces the denotation,

$\overline{helped}\overline{someone}$

$= \lambda k.\exists x.(\mathbf{Person}(x) \wedge k(\mathbf{Help}(x)))$

Thus,

$\overline{everyone}\ \ \overline{help}\ \ \overline{someone} = \overline{everyone}\ \ (\overline{help}\ \ \overline{someone})$

$= \lambda k.\overline{everyone}(\lambda n.(\overline{help}\ \ \overline{someone}).\lambda m.k(m\ \ n))$

$= \lambda k.(\lambda k^{'}.\forall x.(\mathbf{Person}(x)) \rightarrow k^{'}(x))(\lambda n.(\overline{help}\ \ \overline{someone}).\lambda m.k(m\ \ n))$

$= \lambda k.\forall x.\mathbf{Person}(x) \rightarrow (\lambda n.(\overline{help}\ \ \overline{someone})(\lambda m.k(m\ \ n)))(x)$

$= \lambda k.\forall x.\mathbf{Person}(x) \rightarrow (\overline{help}\ \ \overline{someone})(\lambda m.k(m\ \ x))$

$= \lambda k.\forall x.\mathbf{person}(x) \rightarrow (\lambda k^{'}.\exists y.(\mathbf{Person}(y)) \wedge k^{'}(\mathbf{Help}(y)))(\lambda m.k(m\ \ x))$

$= \lambda k.\forall x.\mathbf{Person}(x) \rightarrow (\exists y.\mathbf{Person}(y) \wedge k(\mathbf{Help}\ \ (y))\ \ (x))$

We have the above denotation with the scope of ∀ being wider than the scope of $\exists$ because the rule $(*)$ is used. If we use the rule $(**)$, i.e, if we interpret $\overline{help}$ $\overline{someone}$ before $\overline{everyone}$, the denotation is still kept with the reversal reading scope for $\forall$ and $\exists$ as

$$\lambda k.\exists x.\mathbf{Person}(x) \wedge \forall y.\mathbf{Person}(y).k(\mathbf{Help}\ \ y\ \ x)$$

The two above readings yield a scope ambiguity as discussed by [149, 166]. Further examples of continuation for mathematical functions can be seen in [46, chapter 3] and [220, chapter 3].

4.2.2 The towering notion

The above section shows us how continuation is applied in linguistics. [45, 12] go further by defining the towering notion to elaborate the interaction of continuation and categorial grammar in [66, 84] with empirical study in syntactic analysis. Informally, $A \bbslash B$ is a category of a continuation asserting that B is the returned answer of the category A. In another words, a category A is surrounded by a category B, or B is an expression that requires a subexpression A to be completed. In the reverse order, $C \sslash D$ means that the expression is C if it is surrounded by D. Thus, the general form of a scope taker (such as a quantifier) expression is $C \bbslash (A \sslash B)$.

From the above explanation of continuation, the above formation implies a continuized denotation of the scope taker. The scope taker expression is problematic in syntactic theories because there is no precise method to identify the evaluation scope or type for the expression. It is even more problematic when the scope taker is analyzed in combination with other phenomena, as discussed in [12]. The scope taker, *everyone*, for example, has a category $S \backslash\!\!\backslash (DP /\!\!/ S)$ in [12], which can be expressed vertically as

$$\frac{S|S}{DP}$$

This notation has a counter-clockwise reading starting from the bottom, moving to the above right, and finishing at the above left. It means that the scope-taking of *everyone* takes a continuation of the form, $DP \backslash\!\!\backslash S$ as an input, and returns a type S as a resultant denotation. An example of an expression with a continuation of the category $DP \backslash\!\!\backslash S$ is the following from [12]

John called [] yesterday.

The above expression has a category $DP \backslash\!\!\backslash S$ because it requires an expression of the category DP to fill in the [] to be a complete sentence of the category S. Thus we combine *everyone* with a continuation *John called [] yesterday* to form a (returned) category S.

John called everyone yesterday.

In order to combine the categorical grammar and Montague's semantics with continuation, [12] derived the tower notion which consists of three elements of category grammar for syntax, lexical, and semantics. These are ordered vertically as

$$\left\{\begin{array}{c}\textbf{Categorial grammar + continuation}\\ \\ \textbf{Lexical}\\ \\ \textbf{Montague's semantics}\end{array}\right\}$$

A basic example of the tower notion without continuation is given below; further discussion can be found in [84]

$$\left\{\begin{array}{cc} DP & DP/S \\ \\ John & left \\ \\ \mathbf{J} & \mathbf{Left}\end{array}\right\} = \begin{array}{c} S \\ \\ \textit{John left} \\ \\ \mathbf{Left}(J)\end{array}$$

The combination is explained by adjacent combination in [66] using AB grammar à la Ajdukiewicz. A continuation denotation for *scope taker* is, as in [12],

$$\left\{\begin{array}{cc} \frac{S|S}{DP} & \frac{S|S}{DP\backslash S} \\ \\ everyone & left \\ \frac{\forall y.\,[]}{y} & \frac{[]}{\mathbf{left}} \end{array}\right\} = \begin{array}{c}\frac{S|S}{S}\\ \\ \textit{everyone left}\\ \frac{\forall y.\,[]}{\mathbf{left}(y)}\end{array}$$

The continuation category has been explained above. The continuation semantics of the form $\lambda k.g[k\ f]$ is written as $\frac{g\,[]}{f}$

Thus $\left\{\begin{array}{c} \frac{S|S}{DP} \\ \\ everyone \\ \frac{\forall y.\,[]}{y}\end{array}\right\}$ is identical to $\left\{\begin{array}{c} S \backslash\!\!\backslash (DP /\!\!/ S) \\ \\ everyone \\ \\ \lambda k.\forall y.k\ \ y\end{array}\right\}$

The compositional principle is expressed as the **combination schema** as follows

$$\left(\begin{array}{cc} \frac{C|D}{B/A} & \frac{D|E}{A} \\ \textit{left expression} & \textit{right expression} \\ \frac{g\,[]}{f} & \frac{h\,[]}{x} \end{array}\right) = \left\{\begin{array}{c} \frac{C|E}{B} \\ \textit{left expression right expression} \\ \frac{g\,[h\,[]]}{f(x)} \end{array}\right\}$$

In the above combination, we don't need to read counter-clockwise for the combination on the grammar. Instead, we combine the elements below and above the horizontal directly, i.e. the category B/A is combined with the category A to yield category B, and the category $C|D$ combines with the category $D|E$ to create the category $C|E$.

The above expressions provides a basic definition of the tower notion. Further studied operators on this notion are given below. The type lifting returns a continuation for an expression, and it is equal to type-shifting [216]. The type lowering is the evaluation function in continuation, and the binding introduces the pronoun into the regime.

4.2.2.1 Operators on the towering notion

The **type lifting** operator is defined as

$$\boxed{\begin{array}{ccc} A & & \frac{B|B}{A} \\ \textit{expression} & \textit{LIFT} & \textit{expression} \\ x & \Rightarrow & \frac{[]}{x} \end{array}}$$

For example, the tower $\left\{\begin{array}{c} DP \\ Alice \\ \mathbf{A} \end{array}\right\}$ is lifted into $\left\{\begin{array}{c} \frac{S|S}{DP} \\ Alice \\ \frac{[]}{\mathbf{A}} \end{array}\right\}$

The **type lowering** is defined as

$$\boxed{\begin{array}{ccc} \dfrac{A|S}{S} & \mathit{Lower} & A \\ \mathit{Expression} & & \mathit{Expression} \\ \dfrac{f\,[]}{x} & \Rightarrow & f[x] \end{array}}$$

For example, we can lower the type of the sentence, *everyone runs* as

$$\left\{\begin{array}{c} \dfrac{S|S}{S} \\ \mathit{everyone\ runs} \\ \dfrac{\forall x.[\,]}{\mathbf{run}(x)} \end{array}\right\} \Rightarrow \left\{\begin{array}{c} S \\ \mathit{everyone\ runs} \\ \forall x.\mathbf{run}(x) \end{array}\right\}$$

Both scope ambiguity and evaluation order appear in the strategy for lowering. They also are in the above continuation evaluation order. Finally, we are going to introduce for the rule of the **binding** operator. Basically, the **binding** operator is used for the pronoun resolution. If A, B are categories, then $A \rhd B$ is a category which takes a pronoun of category A, and returns a category B. In the author's opinion, the operator is the delimited continuation in [6], and equal to the metavariable in [122, 137, 121]. [12] provides an example of the definition as

$$\left(\begin{array}{cc} \dfrac{DP \rhd S|S}{DP} & \dfrac{S|S}{DP/S} \\ he & run \\ \dfrac{\lambda x.[]}{x} & \dfrac{[]}{\mathbf{run}} \end{array}\right) = \begin{array}{c} \dfrac{DP \rhd S|S}{S} \\ he\ runs \\ \dfrac{\lambda x.[]}{\mathbf{run}(x)} \end{array} \Rightarrow \begin{array}{c} DP \rhd S \\ he\ runs \\ \lambda x.\mathbf{run}(x) \end{array}$$

Formally, according to [12], the rule for binding is

$$\boxed{\begin{array}{ccc} \frac{A|B}{DP} & \mathit{Bind} & \frac{A|DP \rhd B}{DP} \\ \mathit{Expression} & & \mathit{Expression} \\ \frac{f[]}{x} & \Rightarrow & \frac{f[[]x]}{x} \end{array}}$$

For example, we can bind a quantifier as follow

$$\begin{array}{ccc} \frac{S|S}{DP} & & \frac{S|DP \rhd S}{DP} \\ \mathit{everyone} & \Rightarrow & \mathit{everyone} \\ \frac{\forall x.[\,]}{x} & & \frac{\forall x.[[\,]x]}{x} \end{array} .$$

Hence, the crossover sentence can be derived as the following grammar by [12]

$$\begin{array}{c} \frac{S|DP \rhd S}{DP} \\ \mathit{everyone} \\ \frac{\forall x.[\,]x}{x} \end{array} \left(\begin{array}{c} \frac{DP \rhd S|DP \rhd S}{(DP/S)\backslash DP} \\ \mathit{love} \\ \frac{[\,]}{\mathbf{love}} \end{array} \left(\begin{array}{cc} \frac{DP \rhd S|S}{DP} & \frac{S|S}{DP/DP} \\ \mathit{his} & \mathit{mother} \\ \frac{\lambda y.[\,]}{y} & \frac{[\,]}{\mathbf{mom}} \end{array} \right) \right)$$

$$\frac{S|S}{S} \qquad\qquad\qquad\qquad S$$

$$= \begin{array}{c} \mathit{everyone\ loves\ his\ mother} \\ \frac{\forall x(\forall y.[\,])x}{\mathbf{loves}(\mathbf{mom}y)x} \end{array} \Rightarrow \begin{array}{c} \mathit{everyone\ loves\ his\ mother} \\ \forall x.(\lambda y.\mathbf{love}(\mathbf{mom}y)x)x \end{array}$$

From the above operators and notion [12] derived the scope analysis of various linguistic phenomena such as crossover, superiority, donkey anaphora, etc. While I am not going into detail, this research plays a pivotal role in the contemporary linguistic semantics in [159].

4.2.3 The continuation in monads

The idea of interpreting continuation in monads can be found in [9]. There is a distinctive notion of delimited continuation and undelimited continuation. The undelimited continuation, with its application in linguistics by

[220, 149, 166], treats a result as a whole while delimited continuation breaks the result into composable results. The latter is also called composable continuation in [7, 6, 219, 23]. In general, the undelimited continuation has less expressive power than delimited continuation; however, delimited continuation has a less concrete theoretical foundation such as a formalized type systems, as discussed in [9, 219].

The continuation monad has been studied in linguistics by [17, 22], where the towering notion is defined by replacing the category grammar by the continuation monads [17][p. 70], in order to study the indefinite and its exceptional scope taking. Continuation monads are expressive enough for that to succeed in combination with monad transformers [202]. The author will use the general notion of monads as parameterized monads [23] as an equivalent of delimited continuation [6, 7] in a comparison by [223] in Chapter 5 of this dissertation.

The rest of this section explains the rule of the continuation monads and gives basic examples. They are not much different from the undelimited continuation.

Let us recall the definition of continuation by [9][p. 43]; the rule definitions for the *continuation monad* of the *answer type* ω are described below. The answer type ω can contain the truth value t, or the truth values in many-valued logics.

$$\begin{cases} (1) & \mathbb{M}\alpha = (\alpha \to \omega) \to \omega, \quad \forall \alpha \\ (2) & \eta(a) = \lambda c.c(a) : \mathbb{M}\alpha, \quad \forall a : \alpha \\ (3) & m \star k = \lambda c.m\ (\lambda a.k\ a\ c) : \mathbb{M}\beta, \quad \forall m : \mathbb{M}\alpha, k : \alpha \to \mathbb{M}\beta \end{cases}$$

Rules 1 and 2 define the model and lifting rules as usual. The $\star$ rule means that we are taking the continuation c of type $\beta \to \omega$ as a parameter. Then, we read m and pass the value as a to the computation $(ka)c$. if we compute $k\ a = x : \beta$, then $\lambda c.(k\ a)c = \lambda c.c(x) : \mathbb{M}\beta$ since it passes a value x of a type β to the continuation c.

Each value of a type $\mathbb{M}\alpha$ turns a continuation (type $\alpha \to \omega$) into an answer (type ω). α is a type of a parameter of the continuation, and ω is the return type of the calculation. Thus, $\mathbb{M}\alpha$ is a space for a contraction or application rule since it reduces the length of its elements. Thus, $\mathbb{M}\alpha$, in our definition, is a collection of functions that each take an argument of type α and return a value of type ω.

The variable c in the above definition is a definition of the *continuation*. It is a function, for example of a type $\alpha \to \omega$. For example, our η function above, if we feed it with an element of type α, returns a value of type ω. More specifically, $c : \alpha \to \omega$, if $a : \alpha$, $c(a) : \omega$. Thus, $\lambda c.c(a) : \mathbb{M}\alpha$ since the notion on the left, by our λ-calculus, means that, if we pass a function which takes an argument of type α as our parameter, it will return a value of type ω.

In our unit η examples, $\eta(John) = \lambda c.c(\mathbf{J})$, i.e. the collection of functions that are applicable to *John*. Thus, $\eta(John) : \mathbb{M}e$.

Similarly, $\eta(smokes) = \lambda c.c(\mathbf{smoke})$. If *smoke* is defined by our semantics as an intransitive verb, then $smoke : e \to t$. Thus $\eta(smokes) = \lambda c.c(\mathbf{smoke}) : \mathbb{M}(e \to t)$.

Finally, at our binding rule, i.e. $\star$, our calculation process is: firstly, we compute kac where $k : \alpha \to ((\beta \to \omega) \to \omega)$. This computation involves a value $a : \alpha$ and our continuation $c : \beta \to \omega$. After that, we pass the result to m. The final result is interpreted as: take a continuation of type $\beta \to \omega$ and return a value of type ω, i.e. $\mathbb{M}\beta$.

A monad has two properties: one of these is a unit lift function; the other handles the underlying function. For our continuation monad, we use functional application notation:

$$A_{\mathbb{M}} : \mathbb{M}(\alpha \to \beta) \to \mathbb{M}\alpha \to \mathbb{M}\beta,$$
$$A_{\mathbb{M}}(f)(x) = f \star (\lambda a.x \star (\lambda b.\eta(a(b)))) : \mathbb{M}\beta, \qquad \forall f : \mathbb{M}(\alpha \to \beta), x : \mathbb{M}\alpha.$$

The above rule can be read as: we read f, pass the result as a value a, read x, pass the result as b, and return the value of the functional application of

a to b, i.e. we return $\eta(a(b))$.

By applying our $\star$ operation as per the above definition, we have a concrete λ-calculus formulation

$A_{\mathbb{M}}(f)(x) = \lambda c.f\ (\lambda g.x\ (\lambda y.c(g(y)))), \forall f : \mathbb{M}(\alpha \to \beta), x : \mathbb{M}\alpha.$

This means: we read f and we pass the value as g. Then we read x, we pass the result as y. Finally, we perform the calculation $g(y)$ and pass it as a value to the continuation c.

Let us demonstrate our composition rule with $f = \eta(smoke)$ and $x = \eta(John)$

$[\![John]\!]\ [\![smokes]\!] = A_{\mathbb{M}}[\![smoke]\!]\ \ [\![John]\!]$

$= \lambda c.\eta(\mathbf{smoke})(\lambda g.\eta(\mathbf{John})(\lambda y.c(g(y))))$

$= \lambda c.\eta(\mathbf{J})(\lambda y.c(\mathbf{smoke}(y)))$

$= \lambda c.c(\mathbf{smoke(J)}) : \mathbb{M}\, t.$

In the second line, we replace f, x by $\eta(smoke)$ and $\eta(John)$, respectively. Thus, we take *smoke* as a result of our first computation, i.e. g, and that gives the third-line equation. In the third line, we take $\lambda y.c(smoke(y))$ as the continuation for $\eta(John)$. We thus gain a check on whether *John smokes*, after which the result will pass to the context c that contains the clause, *John smokes.* If *John smokes* is a sentence, then c is just an identity function. To evaluate the final result, i.e. $\lambda c.c(smoke(John))$ to an ω value (in our case, $\omega = t$), we define the *eval* function following [9]:

$eval : \mathbb{M}\,\omega \rightarrow \omega$

$eval\; m = m\; id$ where $id = \lambda v.v$, i.e an identity function.

Thus $eval\;(\lambda c.c(\mathbf{smoke(J)})) = \mathbf{smoke(J)} : t$ with an implicit declaration $c = \lambda v.v$. We should note that we transfer back a type from $\mathbb{M}\,t$ to t in our *eval* function.

Barker [96] interprets quantifier scoping as *evaluation order*. For example, in the interpretation of $A_{\mathbb{M}}$, we interpret f before x. Thus, we have an inverse scope-reading of *someone loves everyone*, i.e.:

$\forall x.\exists y.\mathbf{love}(x, y)$.

Whereas our correct reading is that

$\exists x.\forall y.\mathbf{love}(x, y)$.

To obtain this correct reading, we must evaluate x before the function f

$x \star (\lambda b.f \star (\lambda a.\eta(a(b))))$.

This continuation monad has a disadvantage of being *non-commutative* in that it must follow a strict evaluation order. It is, however, *compositional* and the underlying grammar is still kept (via our $A_{\mathbb{M}}$ rule).

4.2.3.1 Continuation monads in analysing quantifiers

Following the above analysis for quantifiers, we specify the meaning of a quantification such as *everyone* as

$everyone = \lambda c.\forall x.c(x) : \mathbb{M}e.(*)$

We do not use the unit functor η to lift up our quantifier *everyone* as a usual continuation object, because it manipulates the continuation nontrivially. As [222] stated, the sense of quantification is numeric rather than linguistic. Thus, we still need the logical quantifier ∀ in our interpretation. For example, consider the clause *everyone smokes*:

$[\![everyone]\!]\ [\![smokes]\!] = \lambda c.\eta(smoke)(\lambda g.everyone\ (\lambda y.c(g(y))))$.

$= \lambda c.[\![everyone]\!]\ (\lambda y.c(\mathbf{smoke}(y)))$

$= \lambda c.\forall x.c(\mathbf{smoke}(x)) : \mathbb{M}t$, we pass $\lambda y.c(\mathbf{smoke}(y))$ as a continuation for an interpretation of everyone in (*).

To give a semantics of *everyone smokes*, we use the *eval* function to transfer from the monadic value $\mathbb{M}t$ to a t value:

$eval\ (\lambda c.\forall x.c(\mathbf{smoke}(x))) = \forall x.\mathbf{smoke}(x) : t.$

The semantics is true if we observe that, for all x, then x smokes. As we introduce the new notion of variable x, we should be concerned about its scope or x's bounding space. Incorrect manipulation of variable scopes would lead to ambiguity or to multiple interpretations of words.

4.2.4 The recent development of the continuation in linguistics

Contemporary research on the continuation technique in linguistics includes [12, 46, 10, 164]. [46, 164] derive the type-theoretic framework for continuation. The type theoretic framework for continuation is also studied by [220] which uses a well-ordering tree to interpret continuation in Martin-Löf's type theory.

Continuation has strong applications in compiled programming languages. The main idea is the dualist view that a program, or an algorithm, is a function that changes a computer's state to another state by [220]. Thus, instead of looking at the reasoning in the programs, we can conceptualise reasoning in the changing of the memory.

4.3 The state monad for dynamic semantics

An introduction to dynamic semantics is given in section 2.2.3. The substantial study of viewing the state monad as dynamic semantics is credited to Unger [15], with previous research in [11, 122]. Unger's innovation is to stay at the interpretation of meanings as computation in monads which yield advantages over other frameworks. Namely, the state monad provides the rich operators on the states, and clear separation between the dynamic and static interpretation, while keeping the ability for full composition. The state monad also distinguishes between context updating and context accessing, and provides clear referent-handling by state manipulations such as through global and local contexts. Thus, the quantifiers and the indefinite are interpreted in a local context, and deleted after exiting the context. Unger also distinguishes quantifiers from indefinites by the sense that the latter introduce the discourse referent, whereas other quantifiers does not.

Technically, the main supporting hypothesis for the interpretation is the view that the DRS structure is a λ term on the state, i.e. $[x_1, \cdots, x_n | C_1, \cdots, C_m] = \lambda c.\langle C_1 \wedge C_2 \cdots C_m, \langle c; x_n, \cdots; x_1 \rangle\rangle$ where $x_1, \cdots, x_n$ are discourse referents and $C_1, \cdots, C_m$ are conditions. Hence, she interpreted linguistic phenomena in state monads as following.

Let us recall that the state monad is defined in the Chapter 3 as

$$\mathbb{M}\alpha = State \rightarrow (\alpha \times State)$$

i.e. the monad is a function which takes a state as an argument, and returns a pair of a values and a new state. The lifting and binding operators, η and $\star$, are

$$\begin{cases} \texttt{unit}\ \ x = \lambda c.\langle x, c\rangle \\ v \star k = \lambda c.k\pi_1(v\ c)\pi_2(v\ c) \end{cases}$$

Where π_1 and π_2 are the projection functions of the 1st and 2nd element in a tuple $\langle x, y\rangle$. In other words, $\pi_1\langle x, y\rangle = x$ and $\pi_2\langle x, y\rangle = y$. The application function is restated as a compositional semantics of two state monads

$$A_{\mathbb{M}} : \mathbb{M}(\alpha \rightarrow \beta) \rightarrow \mathbb{M}\alpha \rightarrow \mathbb{M}\beta$$
$$A_{\mathbb{M}}\ \ k\ \ v = k \star \lambda f.(v \star \lambda x.\texttt{unit}(f\ x))$$

4.3.1 Lifting linguistic expressions into state monads

The basic syntactic categories—proper names, nouns, pronouns, verbs—are translated into the state monad by the lifting operator unit as follows [15]. Unger interpreted a common noun, such as *unicorn*, as a function from a set of entities to a truth value, as traditional Montagovian semantics in by [14].

$$[\![\texttt{unicorn}]\!] = \texttt{unit}(\lambda x.unicorn\ x) = \lambda c.\langle \lambda x.unicorn\ x, c\rangle$$

Unger also interpreted verbs as predicates, and classified by the number of arguments over entities. She used continuation to lift a verb and proper names. Examples of a proper name, a one-predicate verb *whistles*, and a two-predicate verb *admire* are lifted as below

$\llbracket \texttt{whistles} \rrbracket = \texttt{unit}(\lambda \mathcal{P}.\mathcal{P}(\lambda x.\mathbf{whistle}\ x))$

$\llbracket \mathit{admires} \rrbracket = \texttt{unit}(\lambda \mathcal{P} \lambda \mathcal{Q}.\mathcal{P}(\lambda x.\mathcal{Q}(\lambda y.\mathbf{admire}\ x\ y)))$

$\llbracket \mathit{Alice} \rrbracket = \texttt{unit}(\lambda P.P(\mathbf{A}))$

Now, the compositional principle is applied as above

$A_{\mathbb{M}} \llbracket \mathit{whistles} \rrbracket \llbracket \mathit{Alice} \rrbracket$

$= \lambda c.\langle \lambda \mathcal{P}.\mathcal{P}(\lambda x.\mathbf{whistle}\ x), c\rangle \star \lambda f.(\lambda c.\langle \lambda P.P\ \mathbf{A}, c\rangle \star \lambda x.\texttt{unit}(f\ x))$

$= \lambda c.\langle \mathbf{whistle}\ \mathbf{A}, c\rangle$

4.3.2 State-changing operators

[15, 11] interpret the states as the discourse entities which vary during a linguistics expression in dynamic semantics. Unger's state-changing operators are

- $\hat{}: e \rightarrow \mathit{state} \rightarrow \mathit{state}$. This operator is a function to add entity x to a context or state c.
- $\mathit{sel} : \mathit{state} \rightarrow e$ to select an entity from a context.

Thus, we can have following auxiliary functions:*new*,*get* as following

$$\begin{cases} new : e \to \mathbb{M}\,\top (\text{add an entity to the context and return an unit Boolean truth values} \top). \\ new\ x = \lambda c.\langle \odot, c \hat{}\, x \rangle \end{cases}$$

$$\begin{cases} get : (e \to \mathbb{M}(e \to t)) \to \mathbb{M}(e \to t) \\ get\ m = \lambda c.m(sel\ c)c \end{cases}$$

Thus, we have an example of an interpretation of a proper name

$[\![Alice]\!] = new\ a \gg (\texttt{unit}\lambda P.P\ a) = \lambda c.\langle \lambda P.P\ a, c\hat{}\,a\rangle$

or an interpretation of a general pronoun

$[\![her]\!] = \texttt{get} \gg (\lambda x.\texttt{unit}\lambda P.P\ x) = \lambda c.\langle \lambda P.P(sel\ c), c\rangle$

4.3.3 The discourse representation

In order to combine two sentences, we need an operator that merges two states. To solve this issue, Unger uses the discourse-concatenating operator $\oplus : \mathbb{M}\,t \to \mathbb{M}\,t \to \mathbb{M}\,t$ to combine discourse from two sentences.

$s_1 \oplus s_2 = s_1 \star \lambda p.(s_2 \star \lambda q.\texttt{unit}(p \wedge q))$

This operator means: we read a sentence s_1 and pass the resulting state as p. Then we read the sentence s_2 and pass that resulting state as q. Finally, we return the monadic value as the combination of entities in p and q. For example, the context is added during the computation with two sentences of **Alice whistles** and **Bob admires her** by Unger [15] as

Alice whistles $\oplus$ **Bob admires her**

$= \lambda c.\langle \mathbf{whistle\ a}, \hat{c}\mathbf{a}\rangle \oplus \lambda c.\langle \mathbf{admire}(sel\ c)\ \ \mathbf{b}, c\hat{}\,\mathbf{b}\rangle$

$= \lambda c.\langle (\mathbf{whistle\ a}) \wedge (\mathbf{admire}(sel\ c\hat{}\,\mathbf{a})\mathbf{b}), c\hat{}\,\mathbf{a}\hat{}\,\mathbf{b}\rangle$

4.3.4 Structures adding to states

Unger [15] shows that the complex scope-reading of quantifiers making a single list of entities for state is not suitable for analyzing a sentence discourse. Her example is the discourse, **Every unicorn is eating Bob's flowers. He adores it.** which raises the problem that the pronoun **it** cannot be exactly associated to the correct referee in the previous sentence, i.e. **unicorns**. The underlying cause is that the discourse reference for **unicorns** must be deleted after finishing the first sentence. However, we cannot delete all previous references, such as the global name **Bob**.

Unger's proposal is to add more structure to the state by dividing the state into its global and local contexts. Local contexts exist, and can be deleted in a sentence. Hence, quantifiers can introduce new entities in the local contexts. Since we may have multiple quantifiers in a sentence, the local contexts should be a list of entities. Thus, her proposed state consists of a pair of two sub-states
$state = [\{e\}] \times \{e\}$

Given a state c, Unger divides the `new` operator into $\texttt{new}_{global}$ and $\texttt{new}_{local}$ operators as

$new_{global} x = \langle \odot, (\pi_2\, c)\hat{}\, x \rangle$

$new_{local} x = \langle \odot, add\, x(top(push(\pi_1\, c))) \rangle$

Where the local contexts operators implement a stack:

- $top : [\alpha] \to \alpha$ is a function that reads the top element of the stack.
- $push : [\alpha] \to [\alpha]$ pushes an empty context onto a stack, assigning stack space for new variable.
- $pop : [\alpha] \to [\alpha]$ removes the top element of the stack.
- $add : \alpha \to [\alpha] \to [\alpha]$ adds an element to a set. (for both the global and local context)

We now illustrate these operators by example, with an interpretation that a proper name is a global variable, and that quantifiers introduce local variables.

$[\![Alice]\!] = (new_{global}) \gg (\mathtt{unit} \lambda P.P\, a)$

$[\![she]\!] = get \gg (\lambda x.\mathtt{unit} \lambda P.P\, x)$

4.3.5 Quantifiers and the indefinite

Quantifiers introduce the reference in local contexts and remove it after exiting the scope. Unger defined the function **clear** for this purpose:

$\mathbf{clear} : t \to \mathbb{M}\ t$

$\mathbf{clear} = \lambda x.\lambda c.\langle x, pop\, c \rangle$

This definition enables Unger to derive the denotation for a quantifier such as **every** as

$$\lambda\overline{P}.\overline{Q}.(A_{\mathbb{M}}((\mathbf{new}_{local}x) \rhd (unit\ \ \lambda P.\lambda Q.\forall x.P(x) \to Q(x)))\ \ \overline{P}\ \ \overline{Q}) \star \mathbf{clear}$$

The quantifier is thus cleared after being evaluated on the scope analysis by $\overline{P}, \overline{Q}$ with the generalized quantifier denotation being $\mathbf{new}_{local}x) \rhd (unit\ \ \lambda P.\lambda Q.\forall x.P(x) \to Q(x)$.

The compositional principle is exemplified in analysing, for example, the sentence *every man thinks he is right* as follows, with denotations:

$[\![man]\!] = unit\ \ \lambda x.\mathbf{man}(x)$

$[\![thinks]\!] = unit\ \ \lambda a\lambda b.\mathbf{thinks}\ \ (a)\ \ (b)$

$[\![he]\!] = get \rhd \lambda x.unit\ \ \lambda P.P(x)$

$[\![is\ \ right]\!] = unit\ \ \lambda x.\mathbf{is\_right}\ \ (x)$

The VP *thinks he is right* has the following interpretation:

$$A_{\mathbb{M}}[\![thinks]\!]\ \ [\![he]\!]\ \ [\![is\ \ right]\!] = \lambda c.\langle \lambda x.\mathbf{thinks}(\mathbf{is\_right}(sel\ \ c))\ \ x, c\rangle$$

The NP *Every man* has the interpretation

$$A_{\mathbb{M}}\ \ [\![Every]\!]\ \ [\![man]\!] = A_{\mathbb{M}}\ \ \lambda\overline{Q}.(\lambda c.\langle(\lambda Q.\forall x.\mathbf{man}\ \ x \to Q\ \ (x)), c\hat{\{x\}}\rangle\ \ (\overline{Q})) \star \mathbf{clear}$$

Now, the whole sentence is interpreted by applying the NP to the VP
$\lambda c.\langle \forall x.\mathbf{man}(x) \to \mathbf{think}(\mathbf{is\_right}(sel\ \ c\hat{\{}x\})), c\rangle$

Unger's quantifier-free analysis of the indefinite is more complex than quantifiers because the indefinite's scope is not fixed. Unger observed that the scope of the indefinite depends on the appearance of the quantifiers. Hence, we do not know which of the local or global contexts applies to the indefinite. For example, in her sentence

Alice saw a unicorn in her garden. It was eating the flowers.
an indefinite **a unicorn** acts globally, while in the sentence below, the unicorn acts only locally.
Every formal semanticist saw a unicorn in his garden. $\sharp$ **It was eating the flowers.**

In order to resolve the issue, she proposes the new operator new_{choice} for the selecting the context for an indefinite. This operator chooses to update the local contexts if there is a quantifier. Otherwise, it acts globally. Formally, it is written as

$new_{choice} x = \lambda c.\langle \odot, c + x\rangle$

where $c + x = \begin{cases} add\ \ x\ \ (top(\pi_1\ \ c)) & \text{if there is a local context in c} \\ add\ \ x\ \ (\pi_2\ \ c) & \text{otherwise} \end{cases}$

The non-quantification reading of an indefinite is

$$(new_{choice}x) \rhd unit\ \ \lambda P.\lambda Q.P\ \ (x) \wedge Q\ \ (x)$$

Thus, the sentence **A unicorn barks at Alice. It is afraid**, for example, has the following denotation

$$\lambda c \langle \mathbf{unicorn}(x) \wedge \mathbf{barkAt}\ \ (\mathbf{A})\ \ (x) \wedge \mathbf{afraid}\ \ (sel\ \ \hat{c}(x, \mathbf{A})), \hat{c}(x, \mathbf{A})\rangle.$$

On another hand, the sentence **Every gardener saw a unicorn.** has a reading

$$(\lambda c.\langle \forall x.\mathbf{gardener}x \rightarrow \mathbf{unicorn}\ \ (y) \wedge \mathbf{saw}\ \ (y)\ \ (x), \hat{c}\{x, y\}\rangle) \star \mathbf{clear}$$
$$= \lambda c.\langle \forall x.\mathbf{gardener}(x) \rightarrow \mathbf{unicorn}(y) \wedge \mathbf{saw}\ \ (y)\ \ (x)\rangle$$

4.4 The writer monad for the conventional implicature phenomenon

Following Potts' [78] analysis in section 2.2.4 of this dissertation, [224, 21] argue that there are at least two separate dimensions of the discourse structures for the *at issue* and CI dimensions for conventional implicature

[4]. In order to model the phenomenon in monads, [20] proposes using writer monads. Writer monads, in the sense of [225], are a general write monad with additional stored logging information. Analogically, CI is interpreted as a logging state where additional information is stored at the same time with the main discourse elements in an *at issue*. Intuitively, a writer monad can be thought of as a pair, comprising a value and an additional element of information.

$\mathbb{M}a = a \bullet c$ where $a \bullet c = \langle a, c \rangle$.

The $\bullet$ separates the two dimensions. The first element in the monad is the returned value, and the second element is the logged, or added, information. The first element is the discourse or *at issue* dimension, and the second element is the CI dimension. Thus, the lifting and binding operators between the two dimensions are

$$\begin{cases} 1 \quad \eta(a) : \mathbb{M}\alpha \\ \\ 2 \quad \star : \mathbb{M}\alpha \to (a \to \mathbb{M}\beta) \to \mathbb{M}\beta \\ \quad\ \ a \bullet p \star k = b \bullet (p \wedge q) \text{ where } k\ a = b \bullet q, a : \alpha, b : \beta, k : \alpha \to \mathbb{M}\beta \end{cases}$$

$\top$ means that the logging information is empty. Empty differs from nothing in the sense that an empty value is able to be concatenated later. For example, after the computation $k\ a$, the logging information q is added to

[4] These two dimensions are defined in the section 2.2.4

the previous logging information p, whereas the computing value is b.

Alternatively, we can improve the writer monad to parameterized writer monad (see chapter 5) or the dynamic writer monad in [17]. The reader can skip this part without affecting the interpretation of the phenomenon by [21, 20]. Basically, we are extending the monad with an additional state manipulation, and there is not much change in the writer monad interpretation of the phenomenon.

$\mathbb{M}a : \mathbb{M}\ s_1\ s_2\ \alpha$

$\mathbb{M}a = \lambda s_1.(s_2 = s_1) \wedge \langle a \bullet p \rangle$

with the lifting and binding operators

$$\begin{cases} \eta(a) : \mathbb{M}\ s_1\ s_2\ \alpha \\ \eta(a) = a \bullet \top \text{ (i.e } \lambda s_1.(s_2 = s_1) \wedge \langle a \bullet \top \rangle) \end{cases}$$

$$\begin{cases} \text{m } \star k : \mathbb{M}\ s_1\ s_2\ \alpha \to (\alpha \to \mathbb{M}\ s_2\ s_3\ \beta) \to \mathbb{M}\ s_1\ s_3\ \beta \\ m \star k = (\lambda s_1.s_2 \wedge \langle a \bullet p \rangle) \star k \end{cases}$$

$= \lambda s_1.((k\ a\ s_2) \bullet p)$

$= \lambda s_1.\lambda s_2.(s_3 \wedge \langle b \bullet q \rangle) \bullet p$

$= \lambda s_1.s_3 \wedge \langle b \bullet q \bullet p \rangle$

$= \lambda s_1.s_3 \wedge \langle b \bullet q \wedge p \rangle.$

where $s_2 \wedge \langle a \bullet p \rangle \leftarrow m\ s_1$ and $s_3 \wedge \langle b \bullet q \rangle \leftarrow k\ a\ s_2$.

for convenience, we use the *do* notation

$do\ a \leftarrow m$

which is equal to

$m \star \lambda a$. This process executes the monadic value m, and the result is passed to the next computation as a. Thus

$do\ a \leftarrow m$

$do\ b \leftarrow n$

is a syntactic sugar reading version, i.e an easier to read one, of the complex mathematical notation $m \star \lambda a.n \star \lambda b$. Therefore, the above $\star$ binding is rewritten as

$m \star k =$

$do\ (s_2, a \bullet p) \leftarrow m\ s_1$

$do\ (s_3, b \bullet q) \leftarrow k\ a\ s_2$

$\eta(b \bullet p \wedge q)$

In order to interpret the phenomenon in a compositional manner with both *at issue* and CI dimensions, [20, 21] propose using two implications (applications) on two dimensions. The first implication, $\multimap$ is the normal application in the *at issue* dimension. The second one, $\multimap_*$ is the application rule in the CI dimension. The difference between the two dimensions is what resource to use. $\multimap$ has access to all resource in that dimension, and we must compute it in advance before its use. $\multimap_*$ entails an exchange of resource between two dimensions, and that exchanged resource is reused instead of computed anew.

The defining rules of the two implications are similar, as below

$$\frac{x : A \quad f : A \multimap B}{A(f)(x) : B}\multimap E \qquad\qquad \frac{\begin{array}{c}[x : A]_i\\ \vdots\\ t : B\end{array}}{x \lhd t : A \multimap B}\multimap I_i$$

$$\frac{x : A \quad f : A \multimap_* B}{A_*(f)(x) : B}\multimap_* E \qquad\qquad \frac{\begin{array}{c}[x : A]_i\\ \vdots\\ t : B\end{array}}{x \lhd_* t : A \multimap_* B}\multimap_* I_i$$

Applications A and $A_\star$ combine two expressions, whereas the abstractions $\lhd$ and $\lhd_\star$ create a new function from two variables. The use of $\lhd$ and $\lhd_\star$ is for the type-lifting situation in the sense of Partee in [5], or for continuation in [12]. For example, an expression type NP, can be type-lifted to be $VP \to S$, where $S = NP \to VP$. The examples from [20], below, illustrate the type-lifting of **comma** and **also**.

We use λ notion for computing the resource for the first use, i.e. the A and $\lhd$ definitions. The reused resource is defined by just applying it. Their monadic types are

$$\begin{cases} A(f)(x) : (\mathbb{M}\alpha \to \mathbb{M}\beta) \to \mathbb{M}\alpha \to \mathbb{M}\beta \\ A(f)(x) = f \star a.\ x \star b.\eta(ab) \end{cases}$$

$$\begin{cases} \mathrm{A}_*(f)(x) : (\mathbb{M}\alpha \to \mathbb{M}\beta) \to \mathbb{M}\alpha \to \mathbb{M}\beta \\ A_*(f)(x) = f\ x \end{cases}$$

$$\begin{cases} x \lhd m : \mathbb{M}\alpha \to \mathbb{M}\beta \to \mathbb{M}(\alpha \to \beta) \\ x \lhd m = m \star \lambda n.\eta(\lambda x.n) \end{cases}$$

$$\begin{cases} x \lhd_* m : \mathbb{M}\alpha \to \mathbb{M}\beta \to (\mathbb{M}\alpha \to \mathbb{M}\beta) \\ x \lhd_* m = \lambda x.m \end{cases}$$

Lexical examples from [20] are given below, with an interpretation of comma as a monadic *writer*.

$$\begin{cases} comma : \mathbb{M}\ j \multimap_* \mathbb{M}\ (j \multimap l) \multimap_* \mathbb{M}\ j \\ comma = \lambda j \lambda l.j \star \lambda x.l \star \lambda f.write(f\ x) \star \lambda\_.\eta(x) \end{cases}$$

$$\begin{cases} also : \mathbb{M}\ (d \multimap j \multimap l) \multimap_* \mathbb{M}\ d \multimap_* \mathbb{M}\ j \multimap_* \mathbb{M}\ l \\ also = \lambda v.\lambda o.\lambda s.s \star \lambda x.v \star \lambda f.o \star \lambda y.\mathbf{check}(\exists z.f\ z\ x \wedge z \neq y) \star \lambda\_.\eta(f\ y\ x) \end{cases}$$

$$\begin{cases} John : \mathbb{M}\ \mathbf{J} \\ John = \eta(\mathbf{J}) \end{cases}$$

$$\begin{cases} who : (j \multimap l) \multimap \mathbb{M}\ (j \multimap l) \\ who = \eta(\lambda P.P) \end{cases}$$

$$\begin{cases} likes : c \to j \multimap \mathbb{M}\ l \\ like = \eta(\lambda y.\lambda x.\mathbf{like}(x, y)) \end{cases}$$

$$\begin{cases} cats : \mathbb{M}c \\ cats = \eta(\iota x.\mathbf{cat}^*(x)) \end{cases}$$

$$\begin{cases} likes : d \multimap j \multimap \mathbb{M}\ l \\ likes = \eta(\lambda y \lambda x.\mathbf{like}(x, y)) \end{cases}$$

$$\begin{cases} dogs : \mathbb{M}\, d \\ dogs = \eta(\iota x.\mathbf{dog}^*(x)) \end{cases}$$

The **comma** interpretation can be read as following. Firstly, we read **j** and l (**J** for John, and l for a sentence *who likes cats*). Then we pass **J** and l as x and f, as well as performing $write(f\ x)$ i.e $write(l\ j)$. Therefore, l is lifted in the CI dimension or logging space. Finally, for any results, we return x. We use the wildcard $\lambda\_$ to express the final operation.

The expression **also** is being read as passing s and (f, o) as v and y. If we check from CI that there is z, and $z \neq y$ and $f\ z\ x$, then, for whatever results, we return $f\ y\ x$. **likes** has two type interpretations which are associated with **cats** and **dogs**: $c \to j \to l$ and $d \to j \to l$. Finally, **cats** and **dogs** are the plurals of categories **cat** and **dog**.

The compositional rule for the sentence *John, who likes cats, likes dogs also* is performed as follows. The reading produces several points for future research. For example, there is a polysemy of types in an expression *likes*, and there is no anaphora resolution for *who.* We assume that it is referred to the subject *John.*

$$\dfrac{\dfrac{\dfrac{John}{j} \quad \dfrac{comma}{j \multimap_* (j \rightarrow l) \multimap_* j}}{(j \rightarrow l) \multimap_* j}\multimap_* E \quad \dfrac{\dfrac{who}{(j \multimap l) \multimap (j \multimap l)} \quad \dfrac{\dfrac{likes}{c \multimap j \multimap l} \quad \dfrac{cats}{c}}{j \multimap l}\multimap E}{j \multimap l}\multimap E}{\boxdot j}\multimap_* E$$

$$\dfrac{\dfrac{\dfrac{\dfrac{also}{(d \multimap j \multimap l) \multimap_* d \multimap_* j \multimap_* l} \quad \dfrac{likes}{d \multimap j \multimap l}}{d \multimap_* j \multimap_* l}\multimap_* E \quad \dfrac{dogs}{d}}{j \multimap_* l}\multimap_* E \quad \dfrac{\boxdot}{j}}{l}\multimap_* E$$

Table 4.3: Derivation of an example conventional implicature sentence.

$\boxdot$ is the connection of the sentences for the readable version. From the first part, it returns a type j with the additional CI dimension of *John likes cats*, i.e $\langle j, \mathbf{like}(j, \iota x.\mathbf{cat}^*(x))\rangle$.

The compositional rule is expressed through the typing composition of parts. After the typing composition, the *at issue* dimension is *John also likes dogs* which is of type l, while the CI dimension is *John likes cats.* In order to present the semantics for **also**, we need to do the post-composition analysis using the **check** function. This is done after the composition process since we have to access across both the CI and *at issue* dimensions to draw the verification. The formation semantics of *check* before the post-composition process is

$\langle \mathbf{like}(j, \iota y.\mathbf{dog}^*(y)), \mathbf{like}(j, \iota x.\mathbf{cat}^*(x))\rangle, \mathbf{check}(\exists z, \mathbf{like}(j, z) \quad \wedge \quad z \quad \neq$

$\iota y.\mathbf{dog}^*(y))$.

The post-composition process is performed by using discourse fragments and pointing z as $\iota x.\mathbf{cat}^*(x)$. Hence, we draw $\langle \mathbf{also}(\mathbf{like}, \iota y.\mathbf{dog}^*(y), j) : l, \mathbf{like}(j, \iota x.\mathbf{cat}^*(x))\rangle$ given that $\mathbf{like}(j, \iota y.\mathbf{dog}^*(y)) : l$.

This treatment keeps the CI and *at issue* dimensions apart, and lifts the cross-boundary phenomena to the post-composition procedure, i.e. discourse treatment. I will improve this interpretation by using session type in section 5.3.6 to lift the cross-boundary phenomena during the compositional procedure in section 9.3. We explicitly declare which resource is exchanged in the monadic type declaration. Hence, there is no need to define the implications $\multimap$ and $\multimap_\star$ for the compositional rules, and only the normal application rule $\rightarrow$ is required.

4.5 Discussions

Up to the author's knowledge, there is no thorough summarization of the research of monads in linguistics yet. Additional research on the topic is listed below. The research by [226] also provides a summary of monads in linguistics with additional monads : the list monad for focus, the reader monad for intentionality as reader, and the exceptional monad for presupposition failure. Other detailed linguistic phenomena have been studied in [21, 16, 17, 10]. The focus in Hamblin's semantics is studied through the list monad in [11]. The conjunction fallacy has been studied as a probabilistic monad in [206]. The reader monad, as above, is used to interpret generalized opacity and intentionality in [194]. The indefinite and its related exceptional scope-taking has been studied in [17]. The presupposition phenomenon with projection satisfaction is studied using graded monads in [16].

This chapter summarises previous research on monads in linguistics. [12, 122] provide the continuation approach[5] in linguistics giving insight into the scope analysis problem. [15, 17, 22, 16] focus on the state monad giving insight into dynamic semantics. Finally, [21, 227] give insight into phenomena oriented, such as multidimension, reference opaque, perspective semantics, and hyperintentionality approach in monads. Additional analysis research includes [205, 228].

Monads and computing effects constitute an active research area. In the author's opinion, it opens up fruitful research directions to follow in linguistics. For example, the copredication problem by [172, 173] can be analysed by the probabilistic monads by [196], or we can use update monads [229] to analyse the incremental typed logic by [230]. In addition, the completion monads by [198] have been used to formalize real numbers or [231, 232] to show how we intergrade databases in monads . There are also promising directions for formalizing plurality by previous research by [233, 234, 235] or [65]. Practically,

[5] Mostly using composable continuation monads.

we can interpret dialog as the IO monad by [192, 187] and study a dialog system such as in previous research by [118]. From a theoretical standpoint, it is beneficial to see how the typed predicate logic by [28] is associated with general category theory.

Chapter 5

An introduction to parameterized monads

This chapter presents a theoretical extension of the monads in Chapter 3, which is called parameterized monads. The parameterized monad was originated by [23, 34] under the practical requirement for the generalized model of the computational monads by [8]. Since it is an extension of monads, it preserves the properties of the monadic framework and enriches them with an expressive power while being less generic than an applicative functor framework, such as [37]. Intuitively, according to [53, 236], parameterized monads follow Hoare logic, originated by [51], to extend monads. To explore this, a related study was conducted by [237, 238]. Its strength is able to capture the composable continuation in [7, 6, 75] by following the previous research of [9, 53]. Thus, parameterized monads can encapsulate linguistic theories under the Hoare logic interpretation while preserving the same expressive power to the composable continuation as discussed by [12].

Despite the fact that monads compose a successful mathematical model of computing effects, they have limited expressive power, as observed by [33]. Thus, contemporary research has been attempted to extend the expressive power of monads. For example, a recent attempt was made by [53] and an algebraic effects and handlers approach was taken by [64]. In addition, a summary of the attempted research was given in [239, 240], including notable research by [52, 23, 241]. However, this dissertation was based on a notable study by [23, 35, 242, 243], which created the contemporary popular framework to track the effects of monads.

According to [23, 34], parameterized monads enrich monads by explicitly describing states, or contexts, during the composition of monadic expressions. Adding a state's variation to its monadic expression is difficult from the theoretical perspectives at several points. First, a state's changing is independent from monadic expressions. Therefore, it is difficult to keep the compositional principle in the new theoretical framework. Second, other questions arise, such as how to describe these states and what the states' properties are. These states may vary across fields of studies. Therefore, general modeling is problematic, as discussed by [240]. However, in the author's opinion, it is a starting point for a foundational study of computing. Thus, the rest of this chapter will provide its definitions in category theory and examples from the computing field.

5.1 Strong monads

This section and the next one are based on the previous research by [23, 34] and they provide the mathematical definition of the parameterized monads.

To begin with, let us recall that the monads in this chapter has three main definitions.

- An underlying functor $\mathbb{M} : \mathcal{C} \to \mathcal{C}$
- A unit $\eta_A : A \to \mathbb{M}\ A$, meaning a lifting function from A to $\mathbb{M}A$
- A multiplication, μ_A, function $: \mathbb{M}\mathbb{M}A \to \mathbb{M}A$

In addition, a particular class of monads, which is called a strong monad, shows the interaction of the lifted environments of monads with their contexts. This strong monad has an additional rule, which is called a strength or a natural transformer rule $\tau_{AB} : A \times \mathbb{M}B \to \mathbb{M}(A \times B)$.

An object from a monad $\mathbb{M}A$ models a computation that produces a value of type A. This computation could have several properties, such as side effects or exceptions, during the computation process. Meanwhile, $\mathbb{M}$ retains the underlying relation in $\mathcal{C}$. A simple arrow $A \to \mathbb{M}B$ is a program that turns an input of type A to output of type B. From a computational viewpoint, this is the compositional property of the program.

A unit function, η, transfers an object of type A to its target representation in $\mathbb{M}A$. In terms of computation, it lifts and returns a (typed) value to the new space (type). The multiplication function uses this to combine or sequence computations. If $f : A \to \mathbb{M}B$ and $g : B \to \mathbb{M}C$, then the sequence

$$A \xrightarrow{f} \mathbb{M}B \xrightarrow{\mathbb{M}g} \mathbb{M}\mathbb{M}C \xrightarrow{\mu_C} \mathbb{M}C.$$

means that an input of type A is passed as an argument to a computation f. Next, the result is passed to the computation g. Finally, the double operator is placed on the monad $\mathbb{M}\mathbb{M}$ to $\mathbb{M}$ by the μ law. This computation shows the combination of two computations to produce a single output that can be reused for other computations under the same monad. The definition of the combination of computations requires that the input and output of each computation match.

The strength function transforms a function f, from A to $\mathbb{M}B$ to the monad $\mathbb{M}(A \times B)$. That is, one changes the output of the function. If $f : A \to \mathbb{M}B$, C is an additional context, and then

$$C \times A \xrightarrow{C \times f} C \times \mathbb{M}B \xrightarrow{\tau_{CB}} \mathbb{M}(C \times B)$$

This monadic definition brings the discussion back to the alternative definition of the basic monad $\langle \mathbb{M}, \eta, \star \rangle$ in the section 3.4 and [21, 33]. The $\star$ rule can be interpreted in a different way. That is, if $f : A \to \mathbb{M}B$, then $f^* : \mathbb{M}A \to \mathbb{M}B$. That is, the parameter m of type $\mathbb{M}A$ is given in the operator $m \star f$. It is still equivalent to $m \star f : \mathbb{M}A \to (A \to \mathbb{M}B) \to \mathbb{M}B$ if $m : \mathbb{M}A$.

One can change $\langle \mathbb{M}, \eta, \mu \rangle$ to $\langle \mathbb{M}, \eta, \star \rangle$ by keeping the definitions of $\mathbb{M}$ and η. Furthermore, the $\star$ binding operator can be redefined as $f^* = \langle \mathbb{M}f; \mu_B \rangle$, for $f : A \to \mathbb{M}B$. $\langle \mathbb{M}f; \mu_B \rangle$ is represented as a sequence

$$\mathbb{M}A \xrightarrow{\mathbb{M}f} \mathbb{M}\mathbb{M}B \xrightarrow{\mu_B} \mathbb{M}B.$$

It is transformed from $\mathbb{M}A$ to $\mathbb{M}B$, as seen $\star$ from the above view.

The converse method, which is a transformer of $\langle \mathbb{M}, \eta, \star \rangle$ to $\langle \mathbb{M}, \eta, \mu \rangle$ is defined as follows.

Assuming that $f : A \to B$, one extends functors from $\langle \mathbb{M}, \eta, \star \rangle$ to functors in $\langle \mathbb{M}, \eta, \mu \rangle$ as $\mathbb{M}f = (f, \eta_B)^*$. Since $f : A \to B$, $(f, \eta_B) : A \to \mathbb{M}B$, and $(f, \eta_B)^* : \mathbb{M}A \to \mathbb{M}B$. Therefore, if $f : A \to B$, then $\mathbb{M}f : \mathbb{M}A \to \mathbb{M}B$.

The η operation remains the same.

When setting $\mu_A = id^*_{\mathbb{M}A}$, it can be seen that $g^* : \mathbb{M}A \to \mathbb{M}B$ if $g : A \to \mathbb{M}B$. If $Id_{\mathbb{M}A} : \mathbb{M}A \to \mathbb{M}A$, then $Id^*_{\mathbb{M}A} : \mathbb{M}\mathbb{M}A \to \mathbb{M}A$. Thus, this is the μ law.

The τ operation, or natural transformation, of $\langle \mathbb{M}, \eta, \mu \rangle$ is an additional property of this monad. In addition, this operation obeys axioms in accordance with the η, μ commutative and associative laws.

5.2 An introduction to parameterized monads

As mathematical models of effects computation, monads have limitations, despite their rigorous compositional principle. A notorious difficulty occurs when combining two monads. However, parameterized monads face another limitation—the need to describe the states of monadic expressions. Although a monadic expression $\mathbb{M}A$ preserves the underlying structure of a computation with a value of type A, it gives little information regarding its associated states or contexts. There are state monads and reader monads, and a state monad is defined in the denotation manner rather than at the typed declaration level. However, there are benefits to enriching monads with states as parameterized monads.

First, from a programming perspective, parameterized monads are used to give more precise conditions of states to help a program run by declaring them to be explicitly in their prestate. For example, one could require that a previous function output—a boolean value or the given list—be sorted. This is the prestate condition of the parameterized monad, also known as the indexed monad. In general, this is a monadic version of dependent types, which can be used to more precisely characterize these monads.

From the states perspective, a parameterized monad allows a change of the type of state during the computation time when it is still type safe.

To do this, a parameterized monad not only indicates its return value but also its precondition (input) and postcondition (output). In other words, the monad keeps three types under its control. A state monad cannot do this since it does not allow its state to change type during the computation.

The underlying extension of a monad to a parameterized monad is quite intuitive: one may extend the underlying category $\mathcal{C}$ with an additional category of *state* $\mathcal{S}$. Objects in $\mathcal{S}$ represent *state descriptions*, and arrows represent *logical entailments*. In a rough sense, one can explain the *state* as a region, as explained by [33, 241] .

According to [23], implementing this idea in a monad must redefine basic operations in the monads. First, he extended the underlying functor $\mathbb{M} : \mathcal{C} \to \mathcal{C}$ to $\mathbb{M} : \mathcal{S}^{op} \times \mathcal{S} \times \mathcal{C} \to \mathcal{C}$. The objects in $\mathbb{M}(\mathcal{S}_1, \mathcal{S}_2, A)$ are not merely values of a specific type A; they are the computation that begin in states described in $\mathcal{S}_1$ and end in states described in $\mathcal{S}_2$, producing a return value of the type A. From this author's point of view, this process is similar to a continuation version in a monad. [1]One can strengthen the pre-description and weaken the post description by using additional properties of arrows such as contravariance. That is, the arrows preserve identities and the compositional law. A functor F from $\mathcal{C}$ to $\mathcal{D}$ is called contravariance if

- Associate each object X in $\mathcal{C}$ an object $F(X)$ in $\mathcal{D}$.

[1] This idea was also discovered by [223]

- Associate each morphism $f : X \to Y$ in $\mathcal{C}$ a morphism $F(f) : F(Y) \to F(X)$ in $\mathcal{D}$ such that
  - $F(Id_X) = Id_{F(X)}$ for every object X in $\mathcal{C}$.
  - $F(g \circ f) = F(f) \circ F(g)$ for all morphism $f : X \to Y$ and $g : Y \to Z$ in $\mathcal{C}$.

As one extends the functors from $\mathbb{M} : \mathcal{C} \to \mathcal{C}$ to $\mathbb{M} : \mathcal{S}^{op} \times \mathcal{S} \times \mathcal{C} \to \mathcal{C}$, making the extension more coherent becomes more difficult since each functor in the original formations becomes two more dimensions of *states* and arrows of relations between them. Therefore, one must strengthen the conditions of the unit, multiplication, and transformation laws—that is, η, μ, τ rules.

The unit operator, η is *not* transformed from A to $\mathbb{M}A$. Instead, it transforms from A to $\mathbb{M}(S, S, A)$ and should follow a *dinatural* in S. *dinatural* is another name for keeping the structure coherence. We limited our literature to category theory at this point since we were focusing at its applications. Its dinaturality has a commutative law, which is described as

$$\begin{tikzcd}
A \arrow[r, "\eta_{SA}"] \arrow[d, "\eta_{S'A}"'] & \mathbb{M}(S,S,A) \arrow[d, "\mathbb{M}(S,f,A)"] \\
\mathbb{M}(S',S',A) \arrow[r, "\mathbb{M}(f,S',A)"] & \mathbb{M}(S,S',A)
\end{tikzcd}$$

For each function of related states, $f : S_1 \to S_2$ in $\mathcal{S}$. This diagram means that the computation from A to $\mathbb{M}(S,S^{'},A)$ is commutative for η and f.

The multiplication operator, μ is

$\mu_{(S_1,S_2,S_3,A)} : \mathbb{M}(S_1,S_2,\mathbb{M}(S_2,S_3,A)) \to \mathbb{M}(S_1,S_3,A)$.

Given two functions f,g such that $f : A \to \mathbb{M}(S_1,S_2,B)$ and $g : B \to \mathbb{M}(S_2,S_3,C)$. Their combination is illustrated as follows

$$\begin{tikzcd}
A \arrow[r, "f"] & \mathbb{M}(S_1,S_2,B) \arrow[r, "\mathbb{M}(S_1,S_2,g)"] & \mathbb{M}(S_1,S_2,\mathbb{M}(S_2,S_3,C)) \arrow[r, "\mu(S_1,S_2,S_3,A)"] & \mathbb{M}(S_1,S_3,C)
\end{tikzcd}$$

We canceled the middle state, S_2, as it was the post state for the first computation and immediately was supplied as a pre-state for the second one. If the post state of the first and prestate of the second computations mismatch, it should be dinatural to maintain structural coherence. Explicitly, given $f : S_2 \to S_2^{'}$, the computations sequenced in the arrows should be preserved:

$$\begin{tikzcd}
\mathbb{M}(S_1,S_2,\mathbb{M}(S_2^{'},S_3,A)) \arrow[r, "\mathbb{M}(S_1,f,\mathbb{M}(S_2^{'},S_3,A))"] \arrow[d, "\mathbb{M}(S_1,S_2,\mathbb{M}(f,S_3,A))"'] & \mathbb{M}(S_1,S_2^{'},\mathbb{M}(S_2^{'},S_3,A)) \arrow[d, "\mu_{(S_1,S_2^{'},S_3,A)}"] \\
\mathbb{M}(S_1,S_2,\mathbb{M}(S_2,S_3,A)) \arrow[r, "\mu_{(S_1,S_2,S_3,A)}"] & \mathbb{M}(S_1,S_3,A)
\end{tikzcd}$$

The sequence of arrows that change from S_2 to $S_2^{'}$ is called weakening the post state, whereas the other one is called strengthening the prestate. The diagram shows that the outcomes should be the same, regardless of whether they strengthen the prestate or weaken the post state for the purpose of compositional matching.

The strength operator τ is $\tau_{AB} : A \times \mathbb{M}(S_1, S_2, B) \to \mathbb{M}(S_1, S_2, A \times B)$.

To summarize, one can see that $\mathcal{C}, \mathcal{S}$ form two categories. The $\mathcal{S}$-*parameterised monad* $\mathbb{M}(\eta, \mu)$ of $\mathcal{C}$ is defined with

- A functor $\mathbb{M} : \mathcal{S}^{op} \times \mathcal{S} \times \mathcal{C} \to \mathcal{C}$
- A unit $\eta_{\mathcal{S},A} : A \to \mathbb{M}(\mathcal{S}, \mathcal{S}, A)$ with dinatural in $\mathcal{S}$.
- A multiplication, or pruning, $\mu_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3,A} : \mathbb{M}(\mathcal{S}_1, \mathcal{S}_2, \mathbb{M}(\mathcal{S}_2, \mathcal{S}_3, A)) \to \mathbb{M}(\mathcal{S}_1, \mathcal{S}_3, A)$ with dinatural in $\mathcal{S}_2$.
- A strength $\tau_{A,\mathcal{S}_1,\mathcal{S}_2,B} : A \times \mathbb{M}(\mathcal{S}_1, \mathcal{S}_2, B) \to \mathbb{M}(\mathcal{S}_1, \mathcal{S}_2, A \times B)$

These new operations must follow monadic compositional laws and axioms, such as $\eta, \mu = \mathbb{M}(\mathcal{S}_1, \mathcal{S}_2, \eta); \mu = id$.

5.3 Computing monads in the parameterized monads

This section will reanalyze the examples of parameterized monads in computing given by [23] that follow. In the author's opinion, it is still a prominent research issue to discover, refine, and extend these examples further.

5.3.1 Strong monads inclusion

According to [23], every strong monad can be embedded into the parameterized monad by removing the controlling states. Formally, given the monads $(\mathbb{M}, \eta, \mu)$, its parameterized monads embed $(\mathbb{P}, \eta^{'}, \mu^{'})$ is

$$\mathbb{P}(S_1, S_2, A) = \mathbb{M}\ \ A$$
$$\begin{cases} \eta^{'}_{SA} = \eta_A \\ \mu^{'}_{(S_1\ \ S_2\ \ S_3\ \ A)} = \mu_A \\ \tau^{'}_{S_1\ \ S_2\ \ A\ \ B} = \tau_{AB} \end{cases}$$

5.3.2 Parameterized monads morphism

Given two state categories, $\mathcal{S}, \mathcal{S}^{'}$, and a functor (or morphism) between the two categories $F : \mathcal{S}^{'} \rightarrow \mathcal{S}$ and a parameterized monad $(\mathbb{P}, \eta, \mu)$, the morphism of state will change, as explained by [23], to the new parameterized monad $(\mathbb{P}^{'}, \eta^{'}, \mu^{'})$ is

$$\begin{cases} \mathbb{P}^{'}(S_1^{\prime}, S_2^{\prime}, A) = \mathbb{P}(F(S_1^{\prime}), F(S_2^{\prime}), A) \\ \eta^{'}_{S^{\prime}\ \ A} = \eta_{F(S^{\prime})\ \ A} \\ \mu^{'}_{S_1^{\prime}\ \ S_2^{\prime}\ \ S_3^{\prime}\ \ A} = \mu_{F(S_1^{\prime})\ \ F(S_2^{\prime})\ \ F(S_3^{\prime})\ \ A} \end{cases}$$

5.3.3 The state monad

A related study of Hoare state monads is studied by [52]. However, [23] studied the interpretation of the state monad in parameterized monads with a focus on the abstraction level. One can manipulate the *states* or *stores* by using an update function $f : \mathcal{S} \to \mathcal{S}$. Formally, the standard side effects monad choose an object $\mathcal{S}$ to represent the computer's store and, thus the monad is the functor: $\mathbb{M}A = (\mathcal{S} \times A)^{\mathcal{S}}$, i.e $\lambda s.(s, a)$ in functional programming by [185, 50]. The $\mathcal{S}$ in the power represents the old stores, and $\mathcal{S} \times A$ represents the new stores associated with its values.

The problem with this representation is that this study used only one object to represent the store during the programs. Therefore, to update or manipulate the single cell inside the store, one must process the *whole store*, which is not practically preferable. This is because the function f is total, spanning the entire store $\mathcal{S}$. In addition, this manipulation is not expressive enough for complex programs that require precision and micro manipulations of cells or the types of stores is changing over time. Examples are the Hoare logic cited by [51] and the strong update explained by [237]. Otherwise, one may want a system that is capable of giving the semantics for specific memory address-manipulated functions, such as *malloc* or *alloc* in the programming language C.

To solve this issue, we divided the store into smaller stores to reason and used *separation logic*, as explained by [244], to process the entire store. One may let $\mathcal{C}$ represent a Cartesian closed category, $\mathcal{S} = \mathcal{C}$, defining $\mathbb{M}(S_1, S_2, A) = (S_2 \times A)^{S_1}$ with definitions of η, μ, τ, as usual. The parameterized monad thus takes its old state, or the prestate of type S_1, computed to post state S_2, with a value of type A.. Thus, the state transformations are now explicitly given. For each A of $\mathcal{C}$, this study defined the *read* and *update* the store as:

$$read_A : \mathbb{M}(A, A, A)$$
$$read_A = \lambda s.(s, s)$$
$$write_{XA} : A \to \mathbb{M}(X, A, 1)$$
$$write_{XA} = a \mapsto \lambda s.(a, \odot)$$

The *read* function indicates that we began at a state where a store was of type A and ended at a stage where a store was of type A, and we returned an object of type A. This function does not change its state. On the other hand, the *write* function does not return a value but may change its type of state. This can be interpreted as follows: Taking an arbitrary store of type X, one may replace it with a value of type A and return an unit value $\odot$ if successful. Its prestate has a store of a type X, whereas its post state has a store of a type A.

This study sought to divide the entire store $\mathcal{S}$ into smaller parts to facilitate reading or writing in one part of a store rather than an entire one. As $\mathcal{S}$ is a Cartesian closed category, this study used a linear compositional representation of $\mathcal{S}$, partitioning it into several parts:

$\mathcal{S} = A_1 \times \cdots \times \_ \times \cdots \times A_i \times \cdots \times A_n$. For example, one could separate $\mathcal{S}$ into 3 parts: $A \times B \times C$ and our *read*, and *write* functions are operated only at a specific part:

$$
\begin{aligned}
&read_{S(A)} : \mathbb{M}(S(A), S(A), A)\\
&read_{S(A)} = \lambda s.\text{let } S(a) = s \text{ in}(s, a)\\
&write_{S(A)} : A \to \mathbb{M}(S(X), S(A), 1)\\
&write_{S(A)} = a \mapsto \lambda s.(S(\mapsto a)s, \odot)
\end{aligned}
$$

In this frame, $S(A)$ is the first part of S if it is equal to $A \times B \times C$. In this example, reasoning occurs only at a specific part A rather than throughout the entire S. The function *read* is interpreted as follows: Read location s, and an action result is a value a; return (s, a) with the sugar syntax like this: Let (S(a) = s) in. We bound s to our first part—that is, A, by the type $S(A)$. Similarly, the *write* function was interpreted this way: we replaced a value in a part X by a value part a of part A, and X could be either A, B, or C. In addition, we returned an unit of type 1 for a sign of termination. $S(\mapsto a)s$ means updating the location s with a specific part of store in $S(\_)$ by a value a. Overall, the *write* function is explained as follows: Given a value a of type A, if one procures an arbitrary location s of type X, one can write a to s and return a termination value $\odot$.

The *read* and *write* functions are complex to explain because they manipulate the hardware memory in the computer. Although a program can be view as a function, the actual performances of reading and writing are quite physical during real interaction with the computer's memory. Notably, the change of the entire store should be noted, as there could be malfunctions or exceptions during the execution of the commands in the hardware. Examples could include reading recursive functions in specific locations with nontermination.

Practical applications require various degree of reasoning about stores, including the global state or the local state. Thus, a generic monad with only one type of state would not be expressive enough in practice. In addition, if one changed the type of state manually, this would not be a monad anymore. The proposed solution is to provide a method to automatically govern the store and update if one is only changing a small changing part of the store. This could be achieved through an assumption of the monoidal structures of the store and by performing computations over the structure. Such computations are called lifting operations in separation logic. The detail analysis was referenced in the work of [23], who used the monoidal structure and a lifting function to handle the case through separation logic.

5.3.4 The composable continuation monad

The interpretation of composable continuation by [7] was addressed by [9, 23], and delimited continuation in monads was studied by [245]. The latter of this subsection involves sketching the composable continuation interpretation in parameterized monads, as shown by [23]. Basically, Atkey defined the generalized continuation with the unit and binding operators by following the previous study of [9] as

$$\mathbb{M}\ R_1\ R_2\ A = (A^{R_2})^{R_1}$$

$$\begin{cases} \eta(x) = & \lambda k.k\ x \\ \mu(f) = & \lambda k.f(\lambda k'.k'\ k) \\ \tau(a,f) = & \lambda k.f(\lambda b.k(a,b)) \end{cases}$$

Since the type system of the composable continuation by [7] has the general judgement for an expression

$$\Gamma, \alpha \vdash e : A, \beta.$$

Using the equivalence between type and category theory, the author adopted Atkey's interpretation for the judgement in category as a Cartesian product (Atkey represented it as an arrow).

$[\![\Gamma]\!] \times \mathbb{M}\ [\![\beta]\!]\ [\![\alpha]\!]\ [\![A]\!]$

Thus, the *reset* and *shift* operators were defined by Atkey as

$$reset : \mathbb{M}\ B\ A\ A \rightarrow \mathbb{M}\ C\ C\ B$$

$$reset = \lambda c.\lambda k.k(x(\lambda x.x))$$

$$shift : ((A \rightarrow \mathbb{M}\ C\ C\ B) \rightarrow \mathbb{M}\ E\ D\ D) \rightarrow \mathbb{M}\ E\ B\ A$$

$$shift = \lambda f.f(\lambda v.\eta(k\ \ v))(\lambda x.x)$$

The *reset* operator changed the current continuation to be empty, represented as an identity function, and fed it to the current argument c, returning the result for the future continuation.

The *shift* operator, such as the one given by [75], calls f by the continuation function k such that given a value A, it evaluates the current surrounding context up to the recent *reset* and returns the answer. Thus, it applied the result in this study to the empty continuation.

5.3.5 The writer monad

The writer monad in the previous chapter, or the tracing explained by [34], was found by adding the monoid structure $(G, e, \bullet)$ to the additional space for logging information. Finally, the writer monad was given as

$$\begin{cases} \mathbb{M}\ \ S_1\ \ S_2\ \ A = A \times op(S_1, S_2),\ \text{where } S_1, S_2 \subset G, op \text{ is an operator.} \\ \eta_A = \lambda a.(a, e) \\ \mu_A = ((a, m_1), m_2) \to (a, m_1 \bullet m_2) \\ \tau_{A,B} = (a, (b, m)) \to ((a, b), m) \end{cases}$$

In a parameterized monad, [23, 34] generalized the monoid structure G to be in the smaller category S. The definition was slightly changed to be

$$\mathbb{P}_{\mathcal{S}_1}(S_1, S_2, A) = A \times \mathcal{S}(S_1, S_2)$$
$$\begin{cases} \eta_{SA}(a) = (a, id_S) \\ \mu_{S_1\ S_2\ S_3\ A}((a, s_1), s_2) = (a, (s_1 \otimes s_2)) \end{cases}$$

where $\mathcal{S}_1$ is a small category and S_1, S_2 are in the category S, which is the subcategory of $\mathcal{S}_1$. An example of this monad is the stack machine given below, in which A is the natural number denoted in the number of stack, and G is an actual stack on the computer.

5.3.5.1 The stack machine

Next, this study considered the category **stackProgram**, in which objects are natural numbers for the stack depths, and the arrows are given below

$$\frac{}{n\ \underset{\longrightarrow}{[\,]}\ n} \qquad \frac{i \in \mathbb{Z}}{n\ \underset{\longrightarrow}{push.i}\ n+1} \qquad \frac{}{n+2\ \underset{\longrightarrow}{pop}\ n+1} \qquad \frac{}{n+1\ \underset{\longrightarrow}{dup}\ n+2}$$

$\mathrm{n}_1 \underset{\rightarrow}{t_1} n_2\ \ n_2 \underset{\rightarrow}{c_2} n_3 \frac{}{n_1\ \ \underset{\longrightarrow}{(c_1,c_2)}\ \ n_3}$

Table 5.1: stack machine

Basically, the push, pop, and dup operators, which represent the ⊗ operator, with the additional program transformations c_1, c_2 on the last arrow, that explain the composition rule. The parameterized monad is the $\mathbb{P}$ monad above with the parameterized category $|stackProgram|$, which is the discrete category of *stackProgram*. Thus, one can add additional operators with the monads, as follows

$$push_n : \mathbb{Z} \rightarrow \mathbb{P}_{stackProgram}(1, n, n+1)$$
$$push_n = \lambda i.(\star, (push.i))$$
$$clear_n : 1 \rightarrow \mathbb{P}_{stackProgram}(1, n+2, n+1)$$
$$clear_n = \lambda\_.(\star, pop)$$
$$dup_n : \mathbb{P}_{stackProgram}(1, n+1, n+2)$$
$$dup_n = \lambda\_.(\star, dup)$$

in which $\mathbb{Z}$ is the set of natural numbers.

This is a basic stack program. One can add types for subtyping relations or substructure the stack for the local context. However, the basic idea is that there is a stack with its own arrows and additional operators (that can be the Hoare specification) on the stack.

5.3.6 The IO monad

One may recall that the definition of the interactive IO monad in Section 3.6.6 by [33, 187, 240] was a tree, and operators appeared on the tree in the following manner:

Input monad $\mathbb{M}A = \mu X.A + X^{\mathbb{U}}$

$\eta_A : a$ to a tree with only one leaf labelled with a.

if $f : A \to \mathbb{M}B, c \in \mathbb{M}A, f \star c$ is the tree that replaces leaves of c labelling by a to f a.

Output monad $\mathbb{M}A = \mu X.A + (\mathbb{U} \times X)$

η_A is a map $a \to (\epsilon, a)$

if$f : A \to \mathbb{M}B, f \star (s, a) = (s; s^{'}, b)$ where $fa = (s^{'}, b)$ and $s; s^{'}$ is a concatenation of s and $s^{'}$

The above definition shows no relation between the input, output, and the current states. The parameterized IO monad by [23] shows how one can embed the meaningful state to the IO monads. In order to show this interpretation, he limited the example category $\mathcal{C}$ to be a **set** category. The state category S is a small category, in which objects are states and arrows, such as $S_1 \to S_2$, which are the proofs that S_1 permits all operators that S_2 permits.

If Ω is a set of IO operations in the sense of [192], it is a world. If $action \in \Omega$, it has two governed sets of input, output, and associated states

- in(action): the set of input values under the operation *action*.
- out(action): the set of output values under the operation *action*.
- pre(action): the precondition state that *action* may perform.
- post(action): the post condition state after the *action* is performed.

By restricting the *action*, the objects in the parameterized IO monad are given inductively as follows by an usual definition of the monad, as in the tree illustrated by [23].

$$\frac{a \in A \quad f : S \to S'}{e(f, a) \in \mathbb{P}_\Omega(S, S', A)}$$

$$\frac{action \in \Omega \;\; o \in out(action) \quad k \in in(action) \to \mathbb{P}_\Omega(post(action), S', A) \;\; f : S \to pre(action)}{o(f, action, o, k) \in \mathbb{P}_\Omega(S, S', A)}$$

Table 5.2: IO actions

The first rule defines the return value, and the second rule defines the computation of the input or output actions. The trees for the parameterized IO monad have values at leaves and operations at the node for branching of possible input values for each operation. In addition, an arrow of S exists between each node, acting as witness of compatible with or tracing the operations. It is separated from the normal IO monad.

In the space of S, an arrow in $\mathbb{P}_\Omega(f, g, A)$ precomposes f to the S-arrow at the roof and post composes g to all the S-arrow at the leaves of the tree. On the other hand, in the normal IO space, if $f : A \to B$, $\mathbb{P}_\Omega(S_1, S_2, f)$ would be the usual composition rules on the tree. Thus, the monadic η rule is $\eta\ a = e(id, a)$, and the multiplication rule (μ) concatenates trees by replacing each leaf of the first tree as the roof of the second tree.

Finally, there is an additional primitive operation, which acts as a continuation, for each $op \in \Omega$

$$run_{op} : out(op) \to \mathbb{P}_\Omega(pre(op), post(op), in(op))$$
$$run_{op} = \lambda x.o(id, op, x.\lambda i.(e(i)))$$

Examples of stateful IO devices and session types IO were given by [23], as shown below. There have been other studies on these topics as well, such as current research on information flow; but Atkey's examples clearly illustrate the essential of governing operators under the states in the scope of this dissertation. Further development, for example the work of [36], shows how to substance and operate the states from the database perspective.

5.3.6.1 Stateful IO devices

An overview of the operations is given as

action	*pre*(*action*)	*post*(*action*)	*out*(*action*)	*in*(*action*)
activate	*inactive*	*initialising*	1	1
initData	*initialising*	*initialising*	$\mathbb{B}^a$	1
finishInit	*initialising*	*active*	1	1
read	*active*	*active*	1	$\mathbb{Z}$
write	*active*	*active*	$\mathbb{Z}$	1
shutdown	*active*	*inactive*	1	$\mathbb{Z}$

Table 5.3: controlled IO devices

where $\mathbb{B}$ is the boolean values of $\{true, false\}$.

The input/output in the stateful IO device is operated by six operations **activate, read, write, shutdown, initiate data, finish initiate**, in accordance with three states *inactive, initialising, active.* Before performing the usual **read** and **write** operators in the *active* state, we assumed that the computation is in the *inactive* state and the operators **activate, initiative data, finish initiate** would transform the *inactive* state to the *initialising* and *active* states, respectively. Finally, the **shutdown** operator resets the state from *active* to *inactive.*

In this case, the state category S consisted of three states as objects, and there was no arrow between any states.

5.3.6.2 Session types

The session types were introduced in a study by [246, 247]. One may assume that $X_1, X_2, \cdots$ are sets of values for input/ouput. The states descriptions are then the abstract traces of the IO behavior of a program using the following context-free grammar

$$\mathcal{S} =_{def} ?X | !X | S_1 + S_2 | S_1 \cdot S_2 | \circ$$

A session $?X$ means that the program must take an input value of type X, and $!X$ indicates that it must output a value of type X. The formulae $S_1 + S_2$ shows the choice operator of either performing S_1 or S_2. On the other hand, the formulae $S_1 \cdot S_2$ means sequencing two programs, S_1 and S_2, respectively. Finally, the formulae $\circ$ indicates that termination or no action is possible.

The arrow in S is given by the smallest preorder that considers $S_1 \cdot S_2$ as an associative binary operation, and $\circ, S_1 + S_2$ are met.

The operations are given as follows:

$action$	$pre(action)$	$post(action)$	$out(action)$	$in(action)$
$input_{X,S}$	$?X.S$	S	1	X
$output_{X,S}$	$!X.S$	S	X	1

Table 5.4: session types

These operations generate an infinitive actions index using values of the type X and session S. The translating primitive operations in the parameterized monads are

$$input_{X,S} : 1 \to \mathbb{P}(?X \cdot S, S, X)$$

$$output_{X,S} : X \to \mathbb{P}(!X \cdot S, S, 1)$$

5.4 Specification structures in parameterized monads

There are ongoing research attempts to study the structure of the parameterized monads. For example, Chapter 4 in a study by [248] provided a general theory for parameterized monads lifting as a transformer of monoidal category. Another attempt to explain this was made using the Dijsktra monad in the work of [240, 249]. To conclude, [240] argued that there is not yet a contemporary general story about the pre- and post-conditions of monads. However, these authors argued that applying the relevance theory in linguistics would lend the insight principle to the problem.

From the Hoare logic perspective, the lifting operators were shown in the definition 6 by [23] with $\_ \otimes S$ as a precondition of strengthening and $S \otimes \_$ as a post condition weakening. In addition, there has been related research on this topic. For example, [53] used a lax functor to describe the operator. Another example was given by [39] to provide a specification structure in category theory with the notion of towers using Hoare logic. [39] inferred that this structure is equal to the lax functor, which is the theoretical background explained by [53]. However, further study of this topic is out of this research scope since there remains no clear axiomatic system for this definition. Thus, this study provided the basic definition of category theory shown below

Definition: let $\mathbb{C}$ is a category. A specification structures S over $\mathbb{C}$ is

- a set P A of properties over A for each object A in C
- a relation $R_{A,B} \subseteq P\ A \times \mathbb{C}(A, B) \times P\ B$ for each object pair A, B in C.

Thus, the Hoare triples $\phi\{f\}\psi$, a short abbreviation of $R_{A,B}(\phi, f, \psi)$. If $f : A \to B, g : B \to C, \phi \in P\ A, \psi \in P\ B, \theta \in P\ C$, the axioms for the relation are

(1) $\phi\{id_A\}\psi$

(2) $\phi\{f\}\psi, \psi\{g\}\theta \Rightarrow \phi\{f \circ g\}\theta$

Given C and S, the new category $\mathbb{C}_S$ is defined with the objects as pairs

(A,ϕ), where $A \in Obj(\mathbb{C})$ and $\phi \in P\ A$. The morphism $f : (A, \phi) \rightarrow (B, \psi)$ is a morphism $f : A \rightarrow B$ in C with $\phi\{f\}\psi$

The above axioms guarantee that $\mathbb{C}_S$ is a category. In addition, the faithful functor

$$\mathbb{C} \twoheadleftarrow \mathbb{C}_S$$

is given as

$$A \leftarrow\!\shortmid (A, \phi)$$

Indeed, given the faithful functor $F : \mathbb{D} \rightarrow \mathbb{C}$, the specification structure is defined as

$$P\ A = \{\phi \in Obj(\mathbb{D}) | F(\phi) = A\}$$

$$\phi\{f\}\psi =_{def} \exists \alpha \in \mathbb{D}(\phi, \psi) st.F(\alpha) = f.$$

Thus, there is an equivalent between the specification structures S and the faithful functor $\mathbb{C} \twoheadleftarrow \mathbb{C}_S$. In this way, [39] defined the towering of categories as

$$\mathbb{C}_0 \twoheadleftarrow \mathbb{C}_1 \twoheadleftarrow \mathbb{C}_2 \twoheadleftarrow \cdots \twoheadleftarrow \mathbb{C}_n$$

where the starting $\mathbb{C}_0$ is the basic semantic universe to model computational situations with obvious behavior specifications. Thus, the tower refines $\mathbb{C}_0$ to $\mathbb{C}_n$ by performing specification and progressively verifying more kinds of properties.

5.5 Type systems for parameterized monads

Due to the duality between type theories and category theories discussed by [157], providing a type system for parameterized monads is an equivalent way to model this phenomenon in category theory. This section will outline the basic type system for parameterized monads by following the research of [23]. [223] showed the equivalent between delimited continuation and parameterized monads. However, there is no satisfactory type system for delimited continuation yet, as shown by [219, 6, 40], who explained that the latest research has been based on subtyping. Thus, the same situation is anticipated for the parameterized monad.

The typed command calculus mentioned by [23] is based on the previous research by [250]. The fine-grained calculus in a study by [250] differed from the λ_c calculus used by [8] that employed syntactic characterization of judgments of the *producers* regarding which can produce effects and *values* which cannot. Furthermore, λ_c has no distinction and concerns that all judgments are *producers*.

Basically, this fine-grained calculus includes two form of judgment $\Gamma \vdash^v v : A$ and $\Gamma \vdash^p M : A$ for the type values and producers, respectively. The main construction of the calculus is

$$\frac{\Gamma \vdash^v v : A}{\Gamma \vdash^p \mathit{produce}\ \ v : A} \qquad \frac{\Gamma \vdash^p M : A \quad \Gamma, x : A \vdash^p N : B}{\Gamma \vdash^p M\ \ \mathit{to}\ \ x.N : B}$$

Overall, these processes involve lifting a value to a producer's category and the composition rule, which computes the effective computation M and assigns it as x to the next (effective) computation N. In this author's opinion, these can be seen as axioms on the above specification structures.

Hence, Atkey altered the calculus by adding the state computations to the calculus. Thus, the typed command calculus has three basic judgment formations

$S_1 \vdash^s s : S_2 \quad \Gamma \vdash^v e : A \quad \Gamma; S_1 \vdash^c c : A; S_2$

The first judgment manipulates the states, which is concretely shown as lists in Category S. The second one is type values in the category C with usual constructions in type or category theories of variable, units, pairs, projections, and primitive functions. The last judgment produces effects that include compositional rules for pure values, state terms, sequencing, and primitive functions. Formally, according to [23], these terms are

$s =_{def} \bullet | s.m$

$e =_{def} x | f \;\; e | \star_1 | (e_1, e_2) | \pi_i e | \lambda(x^A; S).c$

$c =_{def} (e; s) | let \;\; x \Leftarrow c_1 \;\; in \;\; c_2 | p \;\; e | e_1 \;\; e_2$

where m, f, and p are primitive functions for the state, value, and computation, which ranges over their primitive types Φ_S, Φ_V, Φ_C. The types for the states are $S, S_1, S_2, \cdots$ for the state category, and the value types are given through the context-free rules

$A =_{def} X \in \mathcal{T}_V | 1 | A_1 \times A_2 | (A_1; S_1) \to (A_2; S_2)$

where $\mathcal{T}_V$ are the primitive types. The context Γ of value types consists of a list of pairs between variable names and their associated types, the usual definition of type theory. Thus, the typing rules explained by [23] are

State calculus

$$\frac{}{S \vdash^s \bullet : S}(S - ID) \qquad \frac{S_1 \vdash^s s : S_2 \quad (m : S_2 \longrightarrow S_3) \in \Phi_S}{S_1 \vdash^s s.m : S_3}(S - primitive)$$

Value calculus

$$\frac{x : A \in \Gamma}{\Gamma \vdash^v x : A}(V - var) \qquad \frac{\Gamma \vdash^v e : A_1 \;\; (f : A_1 \longrightarrow A_2) \in \Phi_V}{\Gamma \vdash^v f\; e : A_2}(V - primitive)$$

$$\frac{}{\Gamma \vdash^v \star_1 : 1} V - 1I$$

$$\frac{\Gamma \vdash^v e_1 : A_1 \quad \Gamma \vdash^v e_2 : A_2}{\Gamma \vdash^v (e_1, e_2) : A_1 \times A_2}(V - \times I) \qquad \frac{\Gamma \vdash^v e : A_1 \times A_2}{\Gamma \vdash^v \pi_i\; e : A_i}(V - \times E_i)$$

$$\frac{\Gamma, x : A_1; S_1 \vdash^c c : A_2; S_2}{\Gamma \vdash^v \lambda(x^{A_1}; S_1).c : (A_1; S_1) \to (A_2; S_2)}(V - \to I)$$

Command calculus

$$\frac{S_1 \vdash^s s : S_2 \quad \Gamma \vdash^v e : A}{\Gamma; S_1 \vdash^c (e, s) : A; S_2}(S - V - C)$$

$$\frac{\Gamma \vdash^v e : A \quad (p : (A; S_1) \longrightarrow (B; S_2)) \in \Phi_c}{\Gamma; S_1 \vdash^c p\; e : B; S_2}(C - primitive)$$

$$\frac{\Gamma; S_1 \vdash^c c_1 : A; S_2 \quad \Gamma, x : A; S_2 \vdash^c c_2 : B; S_3}{\Gamma; S_1 \vdash^c let\; x \Leftarrow c_1\; in\; c_2 : B; S_3}(C - let)$$

$$\frac{\Gamma \vdash^v e_1 : (A; S_1) \to (B; S_2) \quad \Gamma \vdash^v e_2 : A}{\Gamma; S_1 \vdash^c e_1\; e_2 : B; S_2}(C - \to E)$$

Table 5.5: typed command calculus

The **state calculus** is quite simple since it requires no structure in the state. It can be extended to the symmetry monoidal type of calculus [2] in the later part of work by [23] if the state is symmetry monoidal one. This consists of two primitive rules: The first one initiates the state, and the second explains the use of the state's manipulation in the compositional manner.

The **value** and **command calculi** are defined interactively for introduction and elimination rules, which are the abstraction and application rules for $V{-} \to I, C{-} \to E$, respectively. The standard rules of the value calculus consists of the identity, products, and abstraction rules. $V{-} \to I$ introduces the term in an abstraction formation, which is only a syntax producing no effect. $C{-} \to E$ is an elimination rule that has various potential effects. In this author's opinion, if one were to substance the $C{-} \to E$ rules, for example, one could produce the algebraic effects and handlers system described by [64].

The S-V-C rule relates three calculi together. The $C-\mathit{primitive}$ signifies the primitive type of command, and the $C-\mathit{let}$ is the polymorphic sequencing function.

The substitution rules of the value e for others in the **value** and **command calculi** are defined as usual by

[2] an equivalent notion of linear logic, as discussed in a study by [30]

$$y[e/x] = \begin{cases} x \text{ if } x \neq y \\ e \text{ if } x = y \end{cases} \quad (f\ e^{'}[e/x] = f(e^{'}[e/x])$$

$\star\,[e/x] = \star$

$(e_1, e_2)[e/x] = (e_1[e/x], e_2[e/x])$

$(\pi_i\ e^{'})[e/x] = \pi_i(e^{'}[e/x])$

$(\lambda(y^A; S).c)[e/x] = \lambda(y^A; S).(c[e/x])$ with y fresh in e

and

$(e^{'}; s)[e/x] = (e^{'}[e/x]; s)$

$(p\ e^{'})[e/x] = p\ (e^{'}[e/x])$

$(\text{ let } y \Leftarrow c_1 \text{ in } c_2)[e/x] = \text{ let } y \Leftarrow c_1[e/x] \text{ in } c_2[e/x]$ with y is fresh in e

$(e_1^{'}\ e_2^{'})[e/x] = (e_1^{'}[e/x], e_2^{'}[e/x])$.

Finally, the substitution for the state in the **command calculus** is given by

$(e; s^{'})[s/\bullet] = (e; s.s^{'})$

$(p\ e)[s/\bullet] = p\ e$

$(\text{let } x \Leftarrow c_1 \text{ in } c_2)[s/\bullet] = \text{let } x \Leftarrow c_1[s/\bullet]$

$(e_1\ e_2)[s/\bullet] = e_1\ e_2$

Examples for the calculi explained by [23] include the composable continuation's and session types' typing rules, as follow

$$\frac{\Gamma; A \vdash^c c : B; B}{\Gamma; c \vdash^c \text{reset } c : A; C} \qquad \frac{\Gamma, f : (T; D) \to (A; D); B \vdash^c c : O; O}{\Gamma; B \vdash^c \text{shift } f.c : T; A}$$

and

$$\frac{\Gamma \vdash^v e : X}{\Gamma; !X.S \vdash^c \text{output}_{X,S} e : 1; S} \qquad \frac{}{\Gamma; ?X.S \vdash^c \text{input}_{X,S} : X; S}$$

From the above typing rules, one can construct the programs in composable continuation and session types as usual in the literature.

5.6 Discussion

A related theoretical framework was given by the graded monad described by [53] and [236]. Orchard claimed that his framework was more generalized than parameterized monads. However, in the author's opinion, this framework has the strength of interpreting delimited continuation, whereas this interpretation has been known to cause difficulty in the graded monad. This interpretation is essential to provide monads enough expressive power as the delimited continuation approach to achieve the scope analysis in linguistics, as shown in Chapter 19 of [12].

In addition, the recent development of Hoare logic as separation logic by [244] or structural parameterized monads has hinted at an improvement on the topic of glue semantics explained by [251]. This development has provided a model for updating a local context and lifting it to verify a complete context rather than to update a local context and verify the entire context, which would yield resource exhaustion.

Despite the advantage of parameterized monads, this seems to have the disadvantage of writing an inefficient identity function in programming languages. In the author's opinion, the advantages of parameterized monads would outweigh this minor disadvantage in linguistic theories.

Furthermore, there are open theoretical research questions to investigate from this research. For example, one could research how to extend the towering categories. Moreover, the inclusion of the premises in the calculus is problematic in practice, as discussed as the operator *presentIn* in a study by [252].

The applications of indexed monads were studied by [253] for the parsing program, whereas [254] applied it to the domain-specific language. Various forms of monads, such as exceptional errors, were discussed by [249]. Other well-known applications in the literature can be found in the information flow, session types, and the parameterized probabilistic monad. In addition, [255] provided a timed monad, which required time synchronization, similar to parameterized monads.

Linguistically, higher order monads were illustrated by [17][p. 10], and a double continuation was shown by [9], required in an example such as

If ⟨a persuasive lawyer visits a relative of mine ⟩, I'll inherit a fortune.
(selective exceptional existential scope)

We can interpret it as a special case of a parameterized monad, as

$\mathbb{M}\mathbb{M}t \subseteq \mathbb{M}\ s_1\ s_2\ t$

where the explicit state s_2 is the choice of reading the indefinite order. One could say that it is the towering category of two parameterized monads, equivalent to the reading order

$\mathbb{M}\mathbb{M}t = \mathbb{M}\ s_1\ s_2\ t \star \mathbb{M}\ s_2\ s_3\ t$

The advantage of a parameterized monad over a higher-order structure monad is that the former has a firm foundation, as affirmed by [23]. This is the underlying theoretical framework of combining states with other monads. In addition, the research of [17] also combined states and set monads to interpret dynamic semantics by using a framework of monad transformers, as shown by [61]. In a similar manner, the combination in parameterized monads is detailed explained in the chapter 7.

In comparison with monad transformer, this study's approach has advantages in the ability to vary the states inside a sentence. To provide a similar mechanism for state changing, or scope taking inside a sentence, [17] combined monads with the continuation monad listed by [45]. Thus, this study provided a simpler interpretation of the research of [17]. In addition, the parameterized monads framework is semantically oriented by the compositional principles as typing rules, whereas the monad transformers are more operationally oriented. Finally, the mathematical background of the parameterized monad is quite solid and based on category theory, as explained by [35].

Chapter 6

Linguistic structures of parameterized monads

The pioneering research of using monads to interpret linguistics side effects is credited to [63, 6] and is summarized in the chapter 4. At this point, two comments must be made. Firstly, the term *side effects* confused researchers unfamiliar with functional programming and monads. Side effects are not *side*, nor small, effects. Thus, I propose to call them *effects in linguistics*, or an area of interaction between semantics and pragmatics, instead.

Secondly, Shan [6], [75][p. 142], based his foundation on the continuations approach in the delimited control by [7, 74], and criticised other approaches such as [8] as being less concrete. He backed up his reasoning on an assumption that each computational side effect corresponds to a notion of computation expressed as a monad or monad morphism [6][p. 91]. However, a recent study by [13, 245] shows that the picture is not that simple. They compared the expressive power between these foundations of delimited control, algebraic effects and handlers, and monads for modelling effects, and show that these are equivalent in a simple type system. Furthermore, the results of comparison are still conjectural for a complex typing system such as polymorphic types.

In addition, according to [256], which [6] is based on, continuation can provide other effects such maybe, exceptions, or state. However, proving correctness in the continuation-based formulation is challenging and non-trivial. On the other hand, denotational semantics with metavariables in monads by [8] is a traditional way to prove the correctness of program specification[1] such as [38, 50, 33].

The research in this dissertation is based on parameterized monads by [23] which is described in the previous chapter. Thus, this chapter provides an alternative foundation for the effect in parallel with continuations in linguistics by [12] and we are presenting its benefit on analysing the donkey sentence in the next chapter. In a short comparison, our reasoning is based on constructive logics, while Barker and Shan base theirs on classical logic by [214, 149]. To the author's knowledge, the constructive logic has the BHK interpretation, allowing the quantification to be varied in order to compensate for changing of a variable's scope by modifying its formula (a double negation rule, for example, $A \to B = \neg B \to \neg A$ or $A \to B = \neg(A \wedge \neg B)$). In addition, its underlying theory of category theory also has the Yoneda's lemma with a straight interpretation of the implication.[2]

[1] The correctness could mean coreference. [24] also talked about superfluous integration of different logics

[2] another variance of the lemma is the propositional dependence logic in by [257].

The contemporary characterization of related approaches in linguistics are listed as follows. The continuation approach is pursued in [6, 46, 12, 149, 164]. The algebraic effect and handler are pursued by Marsik [10], and the monadic approach is pursued in [16, 15, 21, 17, 22, 63].

An advantage of the parameterized monads framework is that it provides a clear representation of dynamic semantics by using the pre- and post-states in parameterized monads is alighted with Hoare's style in pre- and post-conditions in dynamic semantics. Furthermore, it is also well connected with type-theoretical research in linguistics by the duality between type and category theory. Hence, it inherits a strong contextual modelling background. Finally, it is a denotational semantics with a strong foundation for proving correctness.

In a broad sense, states in this framework are defined similarly to the information states in [179, 181, 117]. The distinction is the theoretical background. Those other frameworks are based on the dependence logic in [153], or on relational algebra and are database-oriented. In contrast, this framework is oriented towards category theory. Thus, there is a diversity of research to draw on in category theory. For example, [31] also shows how we can interpret databases in category theory, with a rich set of examples. A database field is a scheme S, equivalent to a category, and an instance is a functor $I : S \to \textbf{set}$ from the scheme to a set (or any algebraic structure). Hence, their linguistic frameworks are able to be interpreted in category theories[3].

Comparing their strengths, the information states framework by [179, 181, 235] has an advantage in analysing plurality, while this framework has a rich structural analysis such as an evaluation order, and is able to modularise phenomena of the interaction between pragmatics and semantics such as in [21].

Thus, I define the information states in this categorical framework as the context in type theory. Another type-theoretic notion for interpreting information states is the record type in [137]. Record types are a powerful framework to intergrade situation semantics, Montague's semantics for compositionality, and the DRT, surpassing [24]. However, I choose context for the benefit of presentation and, according to [258], the minor diffidence between the dependent record types and the Σ type is the fields label. Furthermore, a context is a loose interpretation of the Σ type with a list of assumptions rather than a Σ type. According to [141], type-theoretic context also has an advantage of handling the presupposition projection.

There are two advantages of this approach. Firstly, type theories has a rich

[3] It is left as a future research direction

analysis of contexts. Previous applications of contexts in type theories in linguistics include [141, 60, 112, 137, 259, 118], with theoretical studies in [260, 261, 262]. Secondly, the translation of contexts from type theories to category theory is quite straightforward due to the duality between them as discussed in chapter 3 and [157, 201]. This **transition technique**,[4] for example, is explained in general category theory textbooks such as [29, 30, 201]. Formally, it is rewritten as below

$$\boxed{\begin{gathered} [\![\varnothing]\!] = 1 \\ [\![\Gamma ; x : A]\!] = [\![\Gamma]\!] \times [\![A]\!] \\ [\![\Gamma ; x_1 : A_1 ; \cdots ; x_n : A_n]\!] = [\![\Gamma]\!] \times [\![A_1]\!] \times \cdots \times [\![A_n]\!] \end{gathered}}$$

Table 6.1: Context in category theory.

A further development of this idea is the slide category by [263, 260]. However, this dissertation uses the notion of type-theoretic judgement, which is related to [19], rather than slide category, to express the context. It is because we model the context in linguistics by following previous research by [86, 141].

[4] Similar to the transition semantics in [181] where the state transition is equivalent to the operator on the formulae

6.1 First-order logic interpretation of natural languages

The theoretical background to the interpretation of first-order logic to category theory is covered in [30, 264]. These works, especially the latter, can lead us to contemporary research such as homotopy type theory in [134]. Overall, however, the majority of interpretation is in a closed Cartesian category, and a linear logic is interpreted as a symmetry monoidal.

The literature on logical interpretation of natural languages is rich. A contemporary research, for example, is developed in [106, 44, 265]. By the Curry-Howard-Lambek correspondence, the interpretation between logic, type and category theories are equivalent, from an abstract perspective. Thus, the author introduces examples of the basic use of first-order logics for interpreting natural languages. It should be noted that, for example, the DRT in [56] is interpretable in first-order logic as discussed in [24][p. 2].

This section is based on the interpretation of propositional and predicate logics in an implementation of the functional programming language Haskell by [25]. Thus, this section provided the interpretation of predicate and propositional logics in category theory where Haskell is an implementation in the programming languages of the theory. The logics are examples of internal languages of category theory. Informally, the interpretation means that category theory is expressive enough to include other languages such as the first-order logic.

To begin with, [25][p. 69] provides a simple, context-free grammar of an English fragment by following inductive rules

S	⟶	**NP VP**
NP	⟶	Snow White|Alice|Dorothy|Goldilocks|Little Mook|Atreyu|**DET CN**|**DET RCN**
DET	⟶	the|every|some|no
CN	⟶	girl|boy|princess|dwarf|giant|wizard|sword|dagger
RCN	⟶	**CN** that**VP**|**CN** *that* **NP TV**
VP	⟶	laughed|cheered|shuddered|**TV NP**|**DV NP NP**
TV	⟶	loved|admired|helped|defeated|caught
DV	⟶	gave

Table 6.2: An English grammar example.

Propositional logic can also be defined in [25][p. 74] as

atom	⟶	$p \mid q \mid r \mid \mathbf{atom}'$
Formula	⟶	$\mathbf{atom} \mid \neg\mathbf{Formula} \mid \mathbf{Formula} \wedge \mathbf{Formula} \mid \mathbf{Formula} \vee \mathbf{Formula}$

Table 6.3: Propositional logic formula

where connectives $\neg, \wedge, \vee$ mean *not, and, or*, respectively. *atom* means a basic proposition, and is listed as $p, p, r, p', q', r', \cdots$. From the above rules, we can define infinite formulae such as $\neg\neg p, p \vee \neg q$, etc.

Furthermore, [25][p. 76] extends the propositional logic by quantifications, and structures basic proposition to a logic called ‘predicate one’. The predicate logic is also called first-order logic, or first-order predicate logic. The quantification in this logic ranges only over entities, not over other formulae. The structured basic proposition expressed that we detailed and characterized the proposition by its arity or the number of its taking parameters. The grammar is given in by [25][p. 76] as

$$\begin{array}{ccc}
\mathbf{v} & \longrightarrow & x|y|z|\mathbf{v}^{'} \\
\mathbf{P} & \longrightarrow & P|\mathbf{P}^{'} \\
\mathbf{R} & \longrightarrow & R|\mathbf{R}^{'} \\
\mathbf{S} & \longrightarrow & S|\mathbf{S}^{'} \\
\mathbf{atom} & \longrightarrow & \mathbf{P\ v}|\mathbf{R\ v\ v}|\mathbf{S\ v\ v\ v} \\
\mathbf{F} & \longrightarrow & \mathbf{atom}|\mathbf{v} = \mathbf{v}|\neg\mathbf{F}|\mathbf{F} \wedge \mathbf{F}|\mathbf{F} \vee \mathbf{F}|\forall\mathbf{v}.\mathbf{F}|\exists\mathbf{v}.\mathbf{F}
\end{array}$$

Table 6.4: predicate logic formulaes

Where **v** is the list of variables. **P, R, S** is a list of propositions by arity. $\exists, \forall$ stand for existence and universal quantifiers. The implication $F_1 \rightarrow F_2$, for example, is interpreted as $\neg(F_1 \wedge F_2)$.

From the above grammar, and taking a pronoun or an anaphora as a variable, we can interpret various examples of English sentences in [266, 44] as follows. For the sake of interpretation, we are skipping the scope of the pronoun at this point.

Thus, the sentence *Maria borrowed the textbook from her professor.* is parsed as

$$\exists x.[\mathbf{textbook}(x) \wedge \exists y.(\mathbf{professor(y)}) \wedge \mathbf{own}(x,y) \wedge \mathbf{borrow}(Maria,y,x)].$$

A discourse such as *A kid is going home. He is whistling* is parsed as

$$\exists x.(\mathbf{Kid}(x) \wedge \mathbf{going\_home}(x)) \wedge \mathbf{Whistle}(p_1).$$

If $p_1 = x$, the interpretation is equivalent to

$$\exists x.(\mathbf{Kid}(x) \wedge \mathbf{going\_home}(x)) \wedge \mathbf{Whistle}(x)$$

Thus, the sentence can be rewritten as

A kid who is going home is whistling

Another complex example by [266] is the discourse

A kid walks down the park.

There is also a dog.

It frightens him and he chases it.

with a formalization as

$\exists x.\mathbf{Kid}(x) \wedge \mathbf{Walk}(x).$

$\exists y.\mathbf{Dog}(y).$

$\mathbf{Frighten}(p_1, p_2) \wedge \mathbf{Chase}(p_2, p_1).$

It can be rewritten as an overall discourse as

$\exists x.\mathbf{Kid}(x) \wedge \mathbf{Walk}(x) \wedge \exists y.\mathbf{Dog}(y) \wedge \mathbf{Frighten}(p_1, p_2) \wedge \mathbf{Chase}(p_2, p_1).$

If we assume an anaphora resolution, it is reduced to

$\exists x.\mathbf{Kid}(x) \wedge \mathbf{Walk}(x) \wedge \exists y.\mathbf{Dog}(y) \wedge \mathbf{Frighten}(y, x) \wedge \mathbf{Chase}(x, y).$

Another sentence with rewritten discourse rules and anaphoric resolution is

Once there was a Queen.

Her son fell in love with a frog.

The prince kissed it and she got mad.

with its basic formalization

$\exists x.\mathbf{Queen}(x)$.

$\exists y.(\mathbf{Son}(y) \wedge \exists z.(\mathbf{Frog}(z)) \wedge \mathbf{love}(y, z))$

$\mathbf{Kiss}(p_1, p_2) \wedge \mathbf{Mad}(p_3)$.

and the rewritten discourse as

$\exists x.\mathbf{Queen}(x) \wedge \exists y.(\mathbf{Son}(y) \wedge \exists z.(\mathbf{Frog}(z)) \wedge \mathbf{love}(y, z)) \wedge \mathbf{Kiss}(p_1, p_2) \wedge \mathbf{Mad}(p_3)$.

Now, the resolved anaphora yields the reading

$\exists x.\exists y.\exists z.\mathbf{Queen}(x) \wedge (\mathbf{Son}(y) \wedge (\mathbf{Fro}(z)) \wedge \mathbf{love}(y, z)) \wedge \mathbf{Kiss}(y, z) \wedge \mathbf{Mad}(x)$.

The above formulae concern with Egli's theorem with its application on solving anaphora resolution in the research by [266, 44]. Egli's theorem redefines the scope of the existence variables in first-order formulae. Formally, it is parsed as

$\exists x.\phi \wedge \psi \leftrightarrow \exists x.(\phi \wedge \psi)$.

with a corollary [44]

$(\exists x.\phi \rightarrow \psi) \leftrightarrow \forall x.\phi \rightarrow \psi.$

[266] provides an implementation of the theorem. However, in the author's opinion, the strength of the DPL, which is associated with the dynamic of the assignments by [44], has not been captured in the research. In addition, in the case of cataphorics, I define an alternative theorem as

$\phi \wedge (\exists x.\psi) \leftrightarrow \exists x.(\wedge\psi)$

Other examples, with their first-order formalization, from [44] are given below

A Canadian farmer, whose horse was ill, went to see his veterinarian. She lent him her donkey. with

$\exists x.(\mathbf{Canadian\_farmer}(x) \wedge \exists y.\mathbf{own\_horse}(x,y) \wedge \mathbf{ill}(y)) \wedge \exists e.\mathbf{see\_veterinarian}(x,e) \wedge \exists v.\mathbf{own\_donkey}(p_2,v) \wedge \mathbf{lent}(p_2,v,p_1).$

There is a boy in the garden. He sneezes. with

$\exists x.\mathbf{boy}(x) \wedge \mathbf{in\_garden}(x) \wedge \mathbf{sneeze}(p_1).$

There once was a king. He lived in a castle. with

$\exists x.\mathbf{King}(x) \wedge \exists y(\mathbf{Castle}(y) \wedge \mathbf{live\_in}(p_1, y))$.

If someone is a king, he lives in a castle. with

$\exists x.\mathbf{King}(x) \rightarrow \exists y(\mathbf{Castle}(y) \wedge \mathbf{live\_in}(p_1, y))$.

A diver found a pearl. She lost it again. with

$\exists x.\mathbf{Diver}(x) \wedge \exists y.(\mathbf{pearl}(y) \wedge \mathbf{found}(x, y)) \wedge \mathbf{lost}(p_1, p_2)$.

In addition, [44] also provides problems outside anaphora resolution. They are, for example, the binding variables operator, i.e. changing the variable names in

She is seeing a woman. She is seeing a woman with a basic interpretation

$\exists y.\mathbf{woman}(y) \wedge \mathbf{see}(p_1, y) \wedge \exists y.\mathbf{woman}(y) \wedge \mathbf{see}(p_1, y)$

and the rewritten formula

$\exists y.\exists x.\mathbf{woman}(x) \wedge \mathbf{see}(p_1, x) \wedge \mathbf{woman}(y) \wedge \mathbf{see}(p_1, y)$

or the entailment relation in the discourse

If a man is from Athens, he is not from Rhodes. There is a man from Athens here. So, he is not from Rhodes. with the interpretation

$\exists x.\mathbf{man}(x) \wedge \mathbf{from\_Athens}(x) \rightarrow \neg\mathbf{from\_Rhodes}(p_1), \exists x.\mathbf{man}(x) \wedge \mathbf{from\_Athens}(x) \vDash \neg\mathbf{from\_Rhodes}(p_1).$

or

A: A man has just drunk a pint of sulphuric acid.

B: Nobody who drinks sulphuric acid lives through the day.

A: Very well then, he will not live through the day.

with

$\exists x.\mathbf{man}(x) \wedge \mathbf{drink\_sulphuric\_acid}(x), \neg\exists x.\mathbf{drink\_sulphuric\_acid}(z) \wedge \mathbf{live\_through\_day}(z) \vDash \neg\mathbf{live\_through\_day}(p_1)$

However, this formalization may lead to an incorrect reading consequence as discussed in [44]

If Jane has a garden, she sprinkles it right now and if Jane owns a house, she has a garden. Now Jane actually owns a house. So she sprinkles it right *now.*

$\exists y.\mathbf{Gargen}(y) \wedge \mathbf{has}(J, y) \rightarrow \mathbf{Sprinkle}(J, p_1), \exists x.\mathbf{House}(x) \wedge \mathbf{Own}(J, x) \rightarrow \exists y.(\mathbf{Garden}(y) \wedge \mathbf{House}(J, y)), \exists x.(\mathbf{House}(x) \wedge \mathbf{Own}(J, x) \vDash \mathbf{Sprinke}(J, p_1))$.

Besides the basic quantification ∃ and ∀, [44] extends them to other operators such as the modality, generalized quantification, presupposition, and belief. The modality operator ◇ illustrates an expression of English words outside affirmative words such as *may, might.* Thus, for example, they are represented in

Someone is hiding in the closet. He might have broken the vase.

with a formalization as

$\exists x.\mathbf{Hide\_in\_closet}(x) \wedge \Diamond \mathbf{broke\_the\_vase}(x)$.

So, we need to modify Egli's theorem for modality as

$\exists x.\phi \rightarrow \psi \Leftrightarrow \exists x.(\phi \wedge \psi)$ if and only if x is not free in the scope of the modal operator ◇ in ψ

The presupposition is being used with the partial operator ∂ to add the situation or event in the analysis of a sentence. $\vDash_s \partial\phi$ means that, in the information state s, the formula ϕ is presupposed to hold in [44][p. 54]. Thus

A fat man was pushing his bicycle.

is parsed with a presupposition that a man own a bicycle

$\exists x.\mathbf{fat\_man}(x) \wedge \exists y.\partial\mathbf{Own\_bicycle}(x, y) \wedge \mathbf{push}(x, y)$.

However, this system is inherited the contradiction of giving information as given in

Someone might have broken the vase. She didn't do it.

with a formalization as

$\exists x.\Diamond\mathbf{broken\_vase}(x) \wedge \neg\mathbf{broken\_vase}(x)$. Since *it* may not referred to the *vase*. Thus, the above formalization may be incorrect.

The generalized quantification can also be given with additional operators such as

No boy likes a girl.

with

$NO(x).(\mathbf{boy}(x))\exists y.\mathbf{girl}(y) \wedge \mathbf{like}(x,y).$

At most five students handed in a cake.

with

$AT\_MOST\_FIVE(x).\mathbf{student}(x).\exists y.\mathbf{cake}(y) \wedge \mathbf{handed}(x,y).$

the *belief* operator is also discussed in [44, 86, 21] with a special notation $\mathcal{B}$ in the examples below from [44]. It should be noted that [21] parsed the belief operator as the reader monad. Intuitively, the information state is someone's belief; hence the belief operator is the reader monad in that information state.

Ralph believes that Ortcutt is a spy. So Ralph believes about Ortcutt that he is a spy.

with

$\mathcal{B}(R,\mathbf{spy(O)}) \vDash \exists x_C(x = O) \wedge \mathcal{B}(R,\mathbf{spy}(x)).$

Ralph (mistakenly) believes that the man with the brown hat is Ortcutt. And

Ralph believes that he is spying.

with

$$\mathcal{B}(R, \exists x_C(\mathbf{brown\_hat}(x) \wedge x = O)) \wedge \mathcal{B}(R, \mathbf{spy}(p_1)).$$

Ralph believes that the man in the brown hat is a spy.

with

$$\mathcal{B}(R, \exists x.\mathbf{brown\_hat}(x) \wedge \mathbf{spy}(x))$$

Ralph believes that the man seen at the beach is not a spy

with

$$\mathcal{B}(R, \exists x.\mathbf{see\_at\_beach}(x) \wedge \neg\mathbf{spy}(x))$$

Ralph believes of Ortcutt that he is a spy.

with

$$\exists x.x = O \wedge \mathcal{B}(R, \mathbf{spy}(x)).$$

Ralph believes of Ortcutt that he is not a spy.

with

$\exists x.x = O \wedge \mathcal{B}(R, \neg\mathbf{spy}(x))$.

Ralph believes there are spies.

with

$\mathcal{B}(R, \exists x.\mathbf{spy}(x))$.

There is someone whom Ralph believes to be a spy.

with

$\exists x.\mathcal{B}(R, \mathbf{spy}(x))$

6.2 Structured information states

The above extensions operators of first-order logic by [44] show their relation to the (information) state. This section provides perspectives on the previous study of the states. All formulations of states can be regarded as an instance of a datatype, particularly in the sense of Muskens and Hoare [24, 51].

6.2.1 Berg's criteria for information states

Berg [117][p. 127] formulated criteria for the states in his research on the definition of discourse for plurality. Since the states are ambiguous in real-life situations, the criteria act as a following guideline. While Berg's main concern is about plurality objects, they are generally accepted as a special case of our definition of states in the section 5.2. His criteria are

- States assign plural objects to discourse referents
- States are able to provide relationships between plural objects
- If a plural object is a subset of another plural object, then the relationship is preserved
- States express relationships between objects if relationships are introduced in the discourse explicitly.
- States only provide values to variables which are introduced in the discourse.

In the author's opinion, Berg's states can be roughly formulated as the record type by [137], or as a heterogeneous collection in the sense of [231, 36] or [31] in category theory. Examples of his states are given below. A basic relationship of the state $G = \{g, h, k, l, m\}$ is expressed as

ID	man	$woman$
g	$Bill$	$Mary$
h	$John$	Ann
k	$Harry$	$Joan$
l	$Charles$	$Joan$
m	$\perp$	$Silvia$

and a state G' to express the old men

man_name	*truth_value*
John	*Y*
Harry	*Y*
Charles	*N*
Bill	*N*

Berg's states can use column to express the properties on the state. Thus, a state expresses more properties, such as old men, men, women, and those women who love old men. An example of a state with dependence relations is

ID	*man*	*woman*	*old_man*	*woman_love_old_men*
g	*Bill*	*Mary*	⊥	⊥
h	*John*	*Ann*	*John*	*Ann*
k	*Harry*	*Joan*	*Harry*	*Joan*
l	*Charles*	*Joan*	⊥	⊥
m	⊥	*Silvia*	⊥	⊥

From the state, we can derive additional operators such as the dependence notion of the relation between *man* and *woman* or the subset relation, for example, *old_ man* as a subset of *man*.

6.2.2 Information states as presuppositions

This section employs previous analysis of context in type theories in linguistics. Since we have the duality between type theory and category theory—dependent type as local Cartesian, Π type as exponential, Σ type as ×—the translation from previous research to category theory is quite straightforward. In type theories, the quantifiers $\forall$ and $\exists$ are interpreted as the Π, Σ types by [86], and contexts are modelled as a list of presuppositions in natural languages by [141, 86].

The research by [141] used contexts and judgements in type theories to parse the *ist* notion in [252]. The notion (*ist* C i) is defined as an additional formulae in first-order logic to express that an expression i is true in the context C. An example of the *ist* notion is the *believe* notion, or $\mathcal{B}$, above.

Intuitively, Boldini's idea is to represent the context C as a regular context in type theory and i as the typing declaration of the notion. Thus, any operator on the notion is related to the contextual morphism and the context is the primary object in his interpretation. For example, let us analyze Boldini's sentence to demonstrate the context in type theory.

The eldest son of the Smith's is at the university.

In order to make the above sentence meaningful, presuppositions below are required

- The noun phrase **the Smith's** implies that they are a married couple.
- The noun phrase **eldest son** implies semantical presuppositions: the Smiths have more than one son, and, among those, there is an eldest, by the constraint of an adjective **eldest**.

Thus, the noun phrase **the Smith's** is parsed as

Smith's : (x_1 : Man) (x_2 : Woman) couple (x_1, x_2) $\wedge$ married(x_1, x_2).

In this formalization, we use the dependent type, i.e. the type *married couple* depends on two persons: husband and wife, which are represented as *man* and *woman* types. The type is interpreted as: there are a man x_1, a woman x_2, and x_1, x_2 are couple and married. In order to certify that **the Smith's** is an element of the type *married couple*, a proof of x_1, x_2 is provided. In this case, they are the Smith husband and wife.

However, our approach is slightly different from Boldini's one, viz. *[Smith's: Man × Woman, x_1: married(p(Smith's), q(Smith's))]* by the type formation. We require one type to declare the fact that **the Smith's** is a married couple, while Boldini needs two. More clearly, he breaks **the Smith's** into two types: they are a couple which consists of a man and a woman, and a proposition that they are married.

Let *sons* be a predicate that takes a married couple and produces a list of their sons, i.e. *sons: married couple* → *[child].* $\sharp$ is a predicate that takes a list and returns the cardinality of the list, i.e. $\sharp$*: (A : Type)* → *[A]* → *Nat.* Then the proposition that *the Smith's has more than one son* is parsed as: $(\sharp\, sons(Smith's)) > 1$, where $>: Nat \rightarrow Nat \rightarrow Prop$. By the *propositions as types* principle, constructing an element of the type $(\sharp\, sons(Smith's)) > 1$, i.e. $x : ((\sharp\, sons(Smith's)) > 1)$ is equal to providing a judgement that this proposition is true.

Let *inc : (A : Type)* → *[A]* → *A* → *Prop*, i.e. a proposition that an element is included in the list. Then the noun phrase, **the eldest son of the Smith's** is parsed as

$$(\exists prop_1 : (\exists(e : child)inc(sons(Smith's), e))) \\ \wedge ((\Pi prop_2 : \exists(x : child)inc(sons(Smith's), x))(age(p(pro_1)) > age(p(prop_2)))).$$

Thus, the presuppositions of the sentence **the eldest son of the Smith's is at the university** is

$$\Gamma = \left[\begin{array}{c} Smith's : (x_1 : Man)(x_2 : Woman)couple(x_1, x_2) \wedge married(x_1, x_2) \\ v_1 : ((\sharp\, sons(Smith's)) > 1) \\ v_2 : (\exists prop_1 : (\exists(e : child)inc(sons(Smith's), e)))\wedge \\ ((\exists prop_2 : \Pi(x : child)inc(sons(Smith's), x))(age(p(pro_1)) > age(p(prop_2)))) \\ v_3 : University \end{array} \right]$$

[5]

Thus, the above sentence is parsed as: $\Gamma \vdash is\_at(v_2, v_3)$. In the author's opinion, v_2, v_3 are discourse references, and presuppositions are constraints on the references. Examples of parsed linguistic phenomena in the context by [141] are parallelism, anaphora, and ellipsis; another one is the formalization of the definite description *the* in [142, 267].

From the above analysis, the parsing process of the sentence in parameterized monads is quite straightforward

$S : \mathbb{M}\ \Gamma\ \Gamma\ \mathbb{B}$

$S = \lambda\gamma.is\_at(v_2, v_3)$

where $\mathbb{B}$ is the Boolean type.

The formalization of the *ist* notion in monads has not been studied. However, in the author's opinion, monads, in the sense of [8], is an instance of this notion. We can think of the notion *(ist C i)* with C being a computational type (i.e. the lifted monadic space in the sense of [8] or the Freyd category in [23]); the notion hence expresses the relation between a type with the value i and its computational type.

The *ist* notion was regarded as a judgement in [141]. His approach looked upon the notion as a short abbreviation of an assertion *is true*, while the definition of judgements $\vdash$ in [93], following Kant, is another word for an assertion of *I assert that.* Thus, the author conjectures a hypothesis that two notions are equivalent. In addition, the judgements is being used in declaring a type system in section 5.5, especially the typing declaration $V - \to I$. Hence, the formulae $(x : A; S)$ in $(\Gamma, x : A, S)$ can be regarded as the formalization of *ist* notion in parameterized monads where C is S, and i is an assertion that x is a value of type A. Thus, we can concretise

$ist(C\ \ i)$

as either $C \vdash^{v} i$ or $C \vdash^{p} i$

Therefore, a sequence or nested use of the *ist* notion is equivalent to a program with monads and the *do* notation. This usage would lead to the related research by [253] using indexed monads and applicative functors to construct parsers. However, we should notice that the notion *ist* does not need to satisfy the monad laws, even the laws of the applicative functor. On the other hand, restricted rules, such as when using parameterized monads, yield additional properties such as composable continuation.

In this dissertation, the author represents the contexts as a computing state, or practically as a stack machine, in a parameterized monad as per

[23][p. 7–8]. The dynamic operators on the stack follow the structures adding to state section 3.4 or [15]. Hence, axioms over the stack, such as in [268], define the practical correctness of the logical implication ⇒. This is done because, in order to make an implication claim, we need to check that the implication holds for every element in the stack. Hence, the axioms associated with stacks play important roles.

This idea has been researched in [227]. In addition, this idea is similar to the incremental dynamic semantics in [230], of which a fully elaborated interpretation is in [25, chapter 12, pp. 303–349]. However, we should bear in mind that we improve upon their research by including the swapping technique for the free-variable binding mechanism.

6.2.3 Information states as dot types

The dot objects of complex types such as PHYS or INFO, in the sense of lexical semantics, have been studied in [172, 170, 269, 270]. A recent formalization frameworks for the objects are the coercive subtyping by [170] or the disjoint union $\uplus$ in set theory by [269]. On another hand, this dissertation uses the state to represent the dot objects. Hence, we propose an alternative framework to interpret the objects. This is a category-theoretic interpretation and it faithfully represents the Cartesian product interpretation of the dot objects, as discussed in [173]. Furthermore, the operations on the states can be substantiated by using the operators on meta-objects such as the downward monotone, disjoint union, multiple sets, etc.

The basic introduction of the research on the dot objects is referred to [270, 271]. Basically, dot objects are objects with distinct aspects. We can think of these aspects as corresponding to the above states' construction where the meaning of a word is interpreted by its relevance perspectives, in Chapter 2, rather than by the whole of usages and descriptive meaning. A linguistics example is the copredication or the polysemy phenomenon. An example in [270, 170] is the word *lunch*, as follows

The lunch yesterday was delicious but took forever.

The *lunch* has both a property of describing a food and a property of describing an event. Another example is the word *book* in

John picked up and mastered the mathematical book.

Since the *book* is an object of the verbs *picked up* and *mastered*, it has both the physical and informative properties. Major examples in the survey by [173] are listed below with orthogonal types combined as **Act** $\star$ **Proposition**, such as *promise*, in

I heard John's quick promise from yesterday.

John's promise took months to realize.

State $\star$ **Proposition**, such as *belief*, in

Nothing can shake John's belief.

John's belief is obviously false.

Attribute $\star$ **Value**, such as *temperature*, in

The temperature is 90.

The temperature is rising.

Event $\star$ **Information**, such as *lecture*, in

My lecture lasted an hour.

Nobody understood my lecture.

Event $\star$ **Human**, such as *appointment*, in

Your next appointment is at 3:00 pm.

Your next appointment is a blonde.

Event $\star$ **Music**, such as *concert*, in

The rain started during the concert.

The concert was confusing.

Performance $\star$ **Music**, such as *song*, in

Sophie bought some Lerner and Lowe songs.

Sophie coughed during the song.

Event $\star$ **Physical**, such as *lunch*, in

My lunch lasted too long today.

I pack my lunch on Thursdays.

Information $\star$ **Physical**, such as *book*, in

Mary burned my book on Mahler.

Mary believes all of Chomsky's books.

Material $\star$ **liquid**, such as *coffee*, in

John picked the coffee from the tree.

John drank the coffee in the cup.

Organization $\star$ **(Information** $\star$ **Physical)**, such as *magazine*, in

The magazine fired its editor.

The cup is on top of the magazine.

I disagreed with the magazine.

Process $\star$ **Result**, such as *classification*, in

Linnaeus's classification of the species took 25 years.

Linnaeus's classification contains 3000 species.

Producer $\star$ **Product**, such as the company named Honda, in

Honda raised prices last week.

I used to drive a Honda.

Tree $\star$ **Fruit**, such as *orange*, in

We planted an orange last year.

Mary peeled an orange for breakfast.

Tree $\star$ **Wood**, such as *oak*, in

We trimmed our oak last fall.

We used oak for our cabinets.

Sound $\star$ Information ($\star$ phys), such as *music*, in

I heard the music for hours.

Sophie can read music fluently.

The interpretation of dot objects in parameterized monads is quite straightforward. We use the state with the symmetry monoidal structure to represent and govern the types of the dot objects. The changing of the states in parameterized monads is the selection or coercion use of the type. Thus, the state highlights the important aspect of parameterized monads as providing an additional classification of dot types. For example, if a *book* has both the PHY and INFO types, then the pre-state defined it, and the post-state selects which actual type (either PHY or INFO) is being used. The selection, for example, depends on the actual usage during the compositional process of the word. Thus, examples of the dot objects, under the disjoint union $\uplus$, are represented as follows:

$[\![\mathit{table}]\!] : \mathbb{M}\ \Gamma\ (\Gamma; PHYS)\ \mathbb{B}$

$[\![\mathit{table}]\!] = \lambda\gamma.\exists x : \mathbf{e}.(\mathbf{sing(table}(x)) = \mathbb{T})$

$[\![\mathit{book}]\!] : \mathbb{M}\ \Gamma\ (\Gamma; PHYS \uplus INFO)\ \mathbb{B}$

$[\![book]\!] = \lambda\gamma.\exists x : \mathbf{e} \wedge (\mathbf{sing}(\mathbf{book}(x)) = \mathbb{T})$

$[\![books]\!] : \mathbb{M}\ \Gamma\ (\Gamma; PHYS \biguplus INFO)\ \mathbb{B}$

$[\![books]\!] = \lambda\gamma.\exists x : \mathbf{e} \wedge (\mathbf{book}^{*}(x) = \mathbb{T})$

$[\![be\_informative_{pl}]\!] : \mathbb{M}\ \Gamma\ (\Gamma; INFO)\ \mathbb{B}$

$[\![be\_informative_{pl}]\!] = \lambda\gamma.\exists x : \mathbf{e} \wedge (\mathbf{info}^{*}(x) = \mathbb{T})$

Table 6.5: dot types interpretation in parameterized monads

where multisets[6] are also a representation of the disjoint unions of sets.[7] For example, the multiset $\{1,1,2\} = \{1,2\} \biguplus \{1\}$, or $\{1,1,1\} = \{1\} \biguplus \{1\} \biguplus \{1\}$. Additional state operators are listed as follows. We define the singular or **sing** operator as $\mathbb{M}\ G\ H\ [\![\mathbf{sing}(x)]\!] = \mathbb{T}$ if and only if $G = H \& G(x)$ is a singleton. In a rough sense, this is the uniqueness condition of the definite description in [92].

Other related operators on the state, such as the distributed operator δ, are studied in [272, 117]. A further analysis of conditions can be found in [273] where Ivlieva associated each sentential semantics with an event. Thus, a linguistic expression has a semantic value which is associated with an event. In other words, the research by Ivlieva is a context sensitive analysis, similar to the introduction of the above context γ. Indeed,

[6] Set theory may not be a proper theoretical framework since the meta-objects could be the mereology objects.

[7] The disjoint unions of sets is similar to the linear logic, or the symmetric monoidal version in set theory.

in the author's opinion, we can view an event as a state monad. For example, the sentence *John loves Mary* is only true if there is an event e that it happens. An example with the rewritten notion is described as follows

$$[\![love]\!]_{\langle e \langle e \langle s,t \rangle \rangle \rangle} = \lambda x.\lambda y.\lambda e.love(e)(y)(x)$$

$$[\![[\text{John love Mary}]_{VP}]\!]_{\langle s,t \rangle} = \lambda e.love(e)(John)(Mary)$$

$$[\![[\text{John loves Mary}]_{S}]\!]_{\langle t \rangle} = \mathbb{T} \text{ iff } \exists e.[love(e)(John)(Mary)]$$

6.3 Discussion

The parameterized monads interpretation of the dot type is close to the formal lexical approach of the dot objects by [172] rather than the typed theoretic approach by [170]. In addition, following Chapter 7 of interpreting the cDRT in parameterized monads, this approach is categorized as a dynamic semantic approach rather than a static one by type theories.

There are two further prominent developments on this topic. Firstly, in the semantic perspective, we can define states as the characterization of ontology in [272]. Secondly, we can use states to represent conversational threads, as in [274], with an attempt to extend the logic of the demonstrative (LD) in [203] for practical purposes by adding the conversational thread to the LD.

In the author's opinion, the conversation thread can be interpreted as states. The consequences of conversational threads constitute the specification structure. However, a challenging practical question arises: how to define the similarity between two conversations? the author follows the traditional dynamic semantic interpretation, such as [275], to suggest that two conversations are similar if they differ by at most one variable. However, in the author's opinion, this definition of similarity leads to a challenging question: what is the measure of 'one variable'?

The author also notices another application of parameterized monads to parse a linguistic phenomenon, which is called the switch-referencing. The author sketches its definition and representation below; however, this application still needs a further investigation. According to [276, 277], the switch-reference is a phenomenon in which a morpheme is added to a syntax to state the differences between subjects in clauses of a sentence. [276, p. 45–46] provides a clearer definition:

> Switch-reference is a morpheme, found at the juncture of two clauses, that typically indicates whether the subjects of those two clauses co-refer. For instance, in Kiowa, there are two sentential connectives translated as *when*. When the subjects of the two joined clauses co-refer, the form of *when* is *chḕ̱* (/*tsẽ* :/), glossed

as SS, as seen in (25). When the subjects are disjoint, the form is $\underline{\grave{\bar{e}}}$ (/ẽ:/), glossed as DS(26).

(25)	$H\acute{\bar{e}}b\grave{a}$**ch$\underline{\grave{\bar{e}}}$**	$\grave{e}m$	$s\acute{\bar{a}}u.$
	$[\varnothing - h\acute{e} : b\grave{a} = ts\tilde{e} :]$	$\tilde{e}m-$	$s\acute{ɔ} :$
	$[3s]enter.PF =$ **when.SS**	$[3s : RFL]-$	**sit down.PF**
	When she$_1$ came in	she$_{1/*2}$ sat down	

(26)	$H\acute{\bar{e}}b\grave{a}$**$\underline{\grave{\bar{e}}}$**	$\grave{e}m$	$s\acute{\bar{a}}.$
	$[\varnothing - h\acute{e} : b\grave{a} = \tilde{e} :]$	$\tilde{e}m-$	$s\acute{ɔ} :$
	$[3s] - enter.PF =$ **When.DS**	$[3s : RFL]-$	sit down.PF
	[when she$_1$came in]	she$_{*1/2}$	sat down.

Jacobsen first proposed the term, *switch-reference* to describe a proposed morpheme in Washo (Hokan-Coahuilan, California) that only appeared at the juncture of two clauses whose subjects were disjoint in reference. The term *switch-reference* referred to this apparent switch.

Since the first simple description, this phenomenon has been found to be a universal phenomenon, especially in Papua New Guinea, Australia, or, with other name, as the *same-reference* in Mojave, as discussed in [276].

The interpretation of the switch reference in parameterized monads is quite straightforward. It is

$$(\Gamma, pivot) morpheme (\Gamma, anti - pivot \wedge \text{condition}).$$

where the pre- and post- states are $(\Gamma, pivot)$ and $(\Gamma, anti - pivot \wedge \text{condition})$, respectively. Γ, in our notation, is a short abbreviation for the discourse representation of the *morpheme* as discussed in [277]. For example, the semantic representation of $\mathbf{ch}\overline{\underline{\grave{\mathbf{e}}}}$ is

$$(\Gamma, x)\mathbf{ch}\overline{\underline{\grave{\mathbf{e}}}}(\Gamma, \lambda y. \wedge y = x)$$

The semantic representation of **è** is

$$(\Gamma, x)\grave{\mathbf{e}}(\Gamma, \lambda y \wedge y \neq x)$$

Chapter 7

The cDRT in parameterized monads

This chapter shows another application of parameterized monads in chapter 5, building upon on [15, 17] which show the interpretation of dynamic semantics in monads. An advantage of this approach is a combination of a compositional principle through the λ-calculus expression, or Montagovian semantics, and the discourse structure. The discourse structure is represented in the state monad, and the λ-calculus provides the denotation of an expression over the state. The related research by [17] also goes further to advocate using monads as a framework to interpret dynamic semantics.

The idea of combining Montagovian semantics and the discourse structure, discussed in 2.2.3, also appeared in the cDRT framework by [24]. Muskens combined two frameworks by using the *grafting* technique which transposes the discourse representation semantics (DRS) to the extended framework of Montague's semantics. Hence, the two frameworks are fused. This technique is feasible through the observation that the language of the DRS is first-order logic. Muskens thereby enriches the research by Montague [4] by adding axioms to include the transformation of the DRS in the enriched framework. In the author's opinion, this idea is similar to the denotational semantics in programming languages. For example, it is interpreted in the intermediate languages in [33].

However, the interaction between two frameworks (specifically monads and the cDRT) has not been studied as well as the properties of the combined framework in the cDRT has not been researched. The state monad in [15] is based on [8], and it cannot formalize the cDRT because Musken's framework requires both pre- and post-conditions for each expression. In this section, I will show how we construct Musken's framework in parameterized monads. Hence, I point out how to use category theory as a potential metalanguage to study the underlying structure of the cDRT. This construction contributes to the current accepted research by enriching the literature of monads in linguistics. It also allows the parameterized monads to express both the linguistic category in [278] and types in [24] in state declarations.

The idea of combining the DRT and Montague's semantics also appears in the records type in [137]. The current research on this framework does not achieve that substance, but that occurrence shows the prominence of monads, which is ultimately due to the duality between category theory and type theory in Chapter 3. Intuitively, a record type is an abstract of a stack, or a database schema. Mathematically, its equivalent notion in category theory is fibred category in [261]. Computationally, the recent research of combining database and functional programming is carried out by [231, 232, 36].

The interpretation of the cDRT by [24, 278] in parameterized monads is quite straightforward because both are influenced by the Hoare's logic with pre- and post- states as primitive objects. In addition, they used λ calculus as an underlying languages and it is also being used in the the previous research by [15] which this research is based upon. However, the distinction is that this research yields the categorical semantics rather than model semantics as per traditional semantic interpretation of the DRT.

Furthermore, the distinction between the cDRT and parameterized monads is that the cDRT is based on the simple theory of types by [3]. According to [8], while the simple theory of types is expressive enough to capture computational expressions, it is not rich enough to stand alone as a theory nor as a model of interpretation. For example, [4] has to use model theory to encode the meaning of λ expressions. A further critique can be found in [69]. On the other hand, we use parameterized monads from [23], which followed [8], and has a strong base in category theory, as discussed earlier in Chapter 3.

Both simple theory of types and also monads, a special class in the category theory, describe functions. However, according to [8, p. 21], while both proofs and programs, i.e. the Curry–Howard correspondence, denote functions, they are not the same. In the author's opinion, the difference is that programs focus more on the practical aspect of functions. Thus, if the simple theory of types and monads are regards as proofs-oriented and programs-oriented respectively, the parameterized monads framework is a pragmatic oriented framework for the cDRT.

A recent development of the cDRT is the PCDRT framework [179, 43, 279, 280]. The application of parameterized monads can be substantial in a similar manner by integrating selective generalized quantifiers or plurality in parameterized monads. The author does not explore this idea in detail; however, it seems to be a prominent direction for future research.

Intuitively, the discourse, in an analogy to the computational view, represents the data structure. The data structure can be simple, as a declared example in a toy language [25], or complex, as a database scheme. Thus, dynamic semantics roughly means data-oriented representations. In the author's opinion, the data structure is stored in the states. Hence, the semantics means that treating the pre- and post-states, which represent the change of the data structure, is the primary objective. This idea, which is also explored in [254, 36], is also an advantage of parameterized monads over monads. Therefore, interpreting the basic cDRT in parameterized monads is quite straightforward. However, the straightforwardness does not mean that it is obvious. For example, the reader can see the research in [16] for an alternative possibility. Thus, the author summarises the basic definitions of the cDRT in the next section before providing the translation.

7.1 An introduction to the cDRT

The source papers are [24, 281] for the interpretation of the cDRT framework in parameterized monads. This interpretation, which is described below, has related research in Chapter 3 of [65].

7.1.1 Logic of change

Musken's logic of change consists of four axioms with a type (sorted) logic. Four basic (sorted) types of the logic are

- e for entities
- t for truth values
- π for registers or storages
- s for states

Registers and states denote discourse referents and a list of discourse referents, respectively. Intuitively, a register acts as a place for the referencing function that Muskens calls pigeon-holes. Suppose that we encounter an indefinite such as *a pigeon*. We will create a register called u_{pigeon} which stores an entity of *pigeon* to refer to later on:

Sue has a pigeon$_1$. *She feeds it*$_1$

We can structure registers in addition to the one for indefinites and the one for names. The value in the register for indefinites can be changed or updated; these are called *variable registers.* On the other hand, the values in the registers for names are fixed and called *constant registers.* In programming languages, the first type of register contains variables which can be substituted, and the second type are constants.

In Musken's system, a variable u is used for an unspecific referent, and named with a capital letter such as *Alice, Bob, Tim, Tom* for a specific one. A generic of two is used as v. On the other hand, variables without references denote type e, which is called x with a lower-cased constant such as *alice, bob, tim, tom.* In summary:

type	**meaning**	**variable denotions**	**constant denotions**
s	states	i, j, k, h	
e	entities	$x_1, x_2, \cdots$	$alice, mary, \cdots$
π	registers	v	$u_1, u_2, \cdots$(unspecific discourse referents) $\cdots Alice, Mary, \cdots$(specific discourse referents)

An example of a relation between states and registers is given as the following table. Columns represent states which consist of two forms of registers. Rows show how registers are changed in each state.

	i_1	i_2	i_3	i_4	$\cdots$
$u_1 : \star$	Bob	Joe	Joe	Tim	$\cdots$
$u_2 : \star$	Tim	Tim	Ann	Sue	$\cdots$
$u_3 : \star$	$\star$	Bob	Lee	Lee	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$Tim :$	Tim	Tim	Tim	Tim	Tim
$Joe :$	Joe	Joe	Joe	Joe	Joe

The above table is referred as $\bigvee$ with the type $\pi \to s \to e$, so $\bigvee \ \delta \ i$, the value of the register δ in the state i, is an entity of type e. Hence, a state i can be rewritten as $\lambda v. \bigvee \ v \ i$.

The above structure has additional operators of definitions and axioms as follows.

- $i\,[\delta_1 \cdots \delta_n]\,j$ is an abbreviation of $\forall v((\delta_1 \neq v \wedge \cdots \wedge \delta_n \neq v) \longrightarrow (\bigvee(v)(i) = \bigvee(v)(j)))$ where i, j are states of type s and $\delta_1, \cdots, \delta_n$ are registers of type π. The formula expresses that i and j differ at most in $\delta_1, \cdots \delta_n$.

- $i\,[\,]\,j$ is equal to $\forall v. \bigvee(v)(i) = \bigvee(v)(j)$

- VAR is a predicate of type $\pi \to t$. It is a singling-out of the variable registers in Musken's definition, or, in the author's perspective, it initializes the register. It is represented by the $\star$ notion above.

- Axioms of the above definitions are

$$\begin{cases} \text{Axiom 1} & \forall i.\forall v.\forall x.(VAR(v) \rightarrow \exists j.i[v]j \wedge \bigvee(v)(j) = x) \\ \text{Axiom 2} & VAR(u)\text{where u is an unspecific referent} \\ \text{Axiom 3} & u_n \neq u_m\text{if n } \neq \text{m} \\ \text{Axiom 4} & \forall i.(\bigvee(Tom)(i) = tom), \forall i.\bigvee(Joe)(i) = joe, etc. \\ & \text{i.e. a specific name referent is unchanged.} \end{cases}$$

Axiom 1 clarifies the relevance of register under the states. Axiom 2 initiates the unspecific referent. Axiom 3 addresses an independence between unspecific references. Finally, Axiom 4 offers a fixed point of specific references such as proper names. It also establishes the connection between the constant reference and constants in entities.

According to [24], this logic has an unselective binding property. Since a state is a list of items, a quantification over the state corresponds to multiple quantifications over its substructural items. Formally, the quantification is written following the unselective binding lemma.

Suppose that the registers of type π contain unspecific referents $u_1, \cdots, u_n$, and entities of type e contain variables $x_1, \cdots, x_n$, and ϕ is a formula which does not contain j. The substitution is written as $[\bigvee(u_1)(j)/x_1, \cdots, \bigvee(u_n)(j)/x_n]\phi$. Then we have the following axioms

$i)$ $\forall i.\exists j(i[u_1, \cdots, u_n]j \quad \wedge \quad [\bigvee(u_1)(j)/x_1, \cdots, \bigvee(u_n)(j)/x_n]\phi) \quad \leftrightarrow$ $\exists x_1.\exists x_2.\cdots\exists x_n.\phi$

ii) $\forall j.(i[u_1,\cdots,u_n]j \to [\bigvee(u_1)(j)/x_1,\cdots,\bigvee(u_n)(j)/x_n]\phi) \leftrightarrow \forall x_1.\cdots x_n.\phi$

A reader is referred to [281, 278, 280] for a further explanation of this logic.

7.1.2 Translating boxes to the logic

Let us recall that the semantics for DRT are given as following, in [24]

$SEM1:$ $\| R(\delta_1,\cdots,\delta_n) \| = \{a|\langle \| \delta_1 \|^a,\cdots,\| \delta_n \|^a\rangle \in I(R)\}$

$\| \delta_1$ **is** $\delta_2 \| = \{a| \| \delta_1 \|^a = \| \delta_1 \|^a\}$

$SEM2:$ $\| \textbf{not } K \| = \{a|\neg a'\langle a,a'\rangle \in \| K \|\}$

$\| K_1\textbf{or}K_2 \| = \{a|\exists a'(\langle a,a'\rangle \in \| K_1 \| \vee \langle a,a'\rangle \in \| K_2 \|)\}$

$\| K_1 \Rightarrow K_2 \| = \{a|\forall a'(\langle a,a'\rangle \in \| K_1 \| \to \exists a''\langle a',a''\rangle \in \| K_2 \|)\}$

$SEM3:$ $\| x_1,\cdots,x_n|\gamma_1,\cdots,\gamma_m \| = \{\langle a,a'\rangle|a[x_1,\cdots,x_n]a' \wedge a' \in \| \gamma_1 \| \bigcap \cdots \bigcap \| \gamma_m \|$

$SEM4:$ $\| K_1;K_2 \| = \{\langle a,a'\rangle|\exists a''(\langle a,a''\rangle \in \| K_1 \| \wedge \langle a'',a'\rangle \in \| K_2 \|)\}\}$

where $\delta_1,\cdots,\delta_n$ are discourse referents. Hence, the semantics of DRT in λ calculus are given by abbreviation rules below in [24]

$ABB1$	$R\{\delta_1, \cdots, \delta_n\}$	$\lambda i.R(\bigvee(\delta_1)(i))\cdots(\bigvee(\delta_n)(i))$
	δ_1 **is** δ_2	$\lambda i.\bigvee(\delta_1)(i) = \bigvee(\delta_2)(i)$
$ABB2$	**not** K	$\lambda i.\neg j.K(i)(j)$
	K **or** $K^{'}$	$\lambda i.\exists j.(K(i)(j) \vee K^{'}(i)(j)$
	$K \Rightarrow K^{'}$	$\lambda i.\forall j.K(i)(j) \rightarrow \exists k.K^{'}(j)(k)$
$ABB3$	$[u_1, \cdots, u_n \mid \gamma_1, \cdots\gamma_m]$	$\lambda i.\lambda j.(i[u_1, \cdots, u_n]j \wedge \gamma_1(j) \wedge \cdots\gamma_m(j))$
$ABB4$	$K; K^{'}$	$\lambda i\lambda j.\exists k(K(i)(k) \wedge K^{'}(k)(j)))$

The only distinction between the two interpretations of the DRS as discussed in Chapter 2 and Musken's innovation is that the languages of boxes in the first one is the metalanguage in DRT, while in the second one it is an abbreviation or an intermediate language in the sense in [33]. Other aspects are being kept the same.

The abbreviation works as in the following description. Firstly, following Musken, let us consider a condition in a box such as u_2 *abhors John*. We can use the interpretation of the **is** in abbreviation 1 (ABB1) for *abhors*. Thus, *abhors* can be rewritten by adding the state i to the condition as $\lambda i.abhors(\bigvee(u_2)(i))(\bigvee(John)(i))$. By axiom 3, John is a fixed point, so this formula is equal to $\lambda i.abhors(\bigvee(u_2)(i))(john)$. Now, let us consider a more complex pair of sentences

A man$_1$ *adores a woman*$_2$*. She*$_2$ *abhors him*$_1$*.*

with its DRT or box interpretation being

$[u_1\ u_2|man\ u_1, woman\ u_2, u_1\ adores\ u_2, u_2\ abhors\ u_1]$

Applying the same steps as for the basic condition, this box is equal to the below λ interpretation by using the abbreviation 3 and the interpretation of the above basic condition

$\lambda i.\lambda j.(i[u_1, u_2]j \wedge man(u_1)(j) \wedge woman(u_2)(j) \wedge (u_1\ adores\ u_2)(j) \wedge (u_2\ abhors\ u_1)(j))$

where a common noun *man, woman* is a predicate that requires a state j to have a meaning. Given the structure $\bigvee$ with values in registers, this formula is concretised out as

$\lambda i.\lambda j.i[u_1, u_2]j \wedge man(\bigvee(u_1)(j)) \wedge woman(\bigvee(u_2)(j)) \wedge adores(\bigvee(u_1)(j))(\bigvee(u_2)(j)) \wedge abhors(\bigvee(u_2)(j))(\bigvee(u_1)(j))$

This mechanism can be applied to all boxes. However, this structure does not provide the truth in Boolean logic. In order to do so and giving its interpretation in the predicate logic, Musken employed the unselected binding lemma with the convention definition of truth values in DRT.

Namely, a condition γ is true in a state i in the structure $\bigvee$ if $\gamma(i)$ holds at $\bigvee$. If γ is true in all states i, we say that γ is *true*. Thus, a box K is *true* in state i if $\exists j.K(i)(j)$ is true. Hence, "K is *true*" is a short abbreviation for "K is true in all states of $\bigvee$." Thus, the above sentence has a truth condition in the structure $\bigvee$:

$$\exists j.i[u_1, u_2]j \wedge man(\bigvee(u_1)(j)) \wedge woman(\bigvee(u_2)(j)) \wedge adores(\bigvee(u_1)(j))(\bigvee(u_2)(j)) \wedge abhors(\bigvee(u_2)(j))(\bigvee(u_1)(j))$$

which can be refined by using the unselected binding lemma without substitution:

$$\exists x_1, x_2.man(x_1) \wedge woman(x_2) \wedge adores(x_1)(x_2) \wedge abhors(x_2)(x_1)$$

The above abbreviation rules yield a λ abstraction of a linguistic term. They can also provide application rules, if we see a box as a representation of a condition on the abstracted variables with explicit states or registers. For example, a common noun such as *farmer* is represented as $\lambda v.[|farmer\ \ v]$, and an indefinite *a* as $\lambda P.\lambda P'.[u_2|] \wedge P(u_2) \wedge P'(u_2)$. The application rule of the λ terms is performed normally under Montagovian semantics, or via λ-calculus in [3], which rewrites the combination expression *a farmer* as $\lambda P'.[u_2|] \wedge [|farmer\ \ u_2] \wedge P'(u_2)$. A further rewriting can be achieved by the merging lemma as $\lambda P'.[u_2|farmer\ \ u_2] \wedge P'(u_2)$.

7.1.3 Semantics of a fragment of English

This section, after [24], substantiates the compositional rules above. The syntax or grammar, for example, can be defined by following inductive rules in [24, p. 17]

$T \rightarrow T\ \ S|S$

$S \rightarrow S^{'}S|NP\ \ VP$

$S^{'} \rightarrow IMP\ \ S$

$VP \rightarrow AUX\ \ V^{'}|V^{'}$

$V^{'} \rightarrow V_t\ \ NP|V_{in}$

$NP \rightarrow DET\ \ N^{'}$

$N^{'} \rightarrow N|N\ \ S.$

$Det \rightarrow a,every,no,some$

$NP \rightarrow he,she,it|Mary,\cdots|who,\ whom,\ which$

$N \rightarrow farmer,\ boy$

$AUX \rightarrow doesn't$

$V_t \rightarrow own$

$V_{in} \rightarrow stink$

$IMP \rightarrow if$

The semantics are substantiated by the types, basic lexical definitions of English terms, and additional structural rules on the parsed tree of lexical items.

This is also described as the standard Montagovian semantics in section 2.1.1.

The types are characterized by the number of registers used. Static types are also demonstrated in section 2.1.1 and in [4]. For convenience, $s \to s \to t$ is given as [] for short. Thus, boxes have a type []. Other major syntactic categories have following types: common nouns or intransitive verbs have type $\pi \to s \to s \to t$ (or $[\pi]$). $[[\pi]]$, for example, is an abbreviation of $(\pi \to s \to s \to t) \to (s \to s \to t)$. Transitive verbs have a type $[[[\pi]]\pi]$. Noun phrases have a type $[[\pi]]$. Determiners have a type $[[\pi][\pi]]$. Verb phrases have a types$[[[\pi]]]$ or $[\pi]$ which depends on whether they have an auxiliary or not.

The lexical items are interpreted as following. Firstly, the relation between discourse referents and anaphoric pronouns is basically as follows. Each possible antecedent A, such as a determiner or a proper name, introduces a discourse referent and is denoted $dr(A)$. Thus, $dr(\mathbf{no})$ and $dr(Alice)$ would be $u, Alice$, respectively. On another hand, an anaphoric pronoun selects a referent in the created discourse referents with the *ant*. For example, *ant(it)* $= a$ in the sentence

Sue$_1$ *has a*$_2$ *pigeon. She*$_1$ *feeds it*$_2$

while $dr(a) = u_2$ (or *dr(ant(it))*)and $dr(ant(she_1)) = Sue.$

In the author's opinion, the operators *dr, ant* perform as *read, write* operators on the register u in a computing sense. dr writes a new register while ant reads from a register. Hence, the pronoun is interpreted in association with an unspecific discourse referent, i.e. $it = \lambda P.P(u_{it})$, and a proper name is associated with a specific discourse referent, i.e. $Sue = \lambda P.P(Sue)$.

Other lexical items are interpreted in accordance with standard Montagovian semantics and Musken's above types as follows

Expression	λnotation	Type
a^n	$\lambda P'.\lambda P.([u_n\|]; P'(u_n); P(u_n))$	$[[\pi][\pi]]$
no^n	$\lambda P'.\lambda P.[\|\mathbf{not}([u_n\|]; P'(u_n); P(u_n))]$	$[[\pi][\pi]]$
$every^n$	$\lambda P'.\lambda P.[\|([u_n\|]; P'(u_n) \Rightarrow P(u_n))]$	$[[\pi][\pi]]$
$Mary^n$	$\lambda P.P(Mary)$	$[[\pi]]$
he_n	$\lambda P.(P(\delta)), \delta = dr(ant(he_n))$	$[[\pi]]$
e_n	$\lambda P.P(v_n)$	$[[\pi]]$
who	$\lambda P'.\lambda P.\lambda v.P(v); P'(v)$	$[[\pi][\pi]\pi]$
$farmer$	$\lambda v.[\|farmer\ v]$	$[\pi]$
$love$	$\lambda Q.\lambda v.Q(\lambda v'.[\|v\ loves\ v'])$	$[[[\pi]]\pi]$
$doesn't$	$\lambda P.\lambda Q.[\|\mathbf{not}(Q(P))]$	$[[\pi][[\pi]]]$
if	$\lambda pq.[p \Rightarrow q]$	$[[][]]$

To the basic lexical rules above, Musken adds inductive rules to construct a mother node in a tree from its daughters. They are

- COPYING: if $A \rightsquigarrow \alpha$ and A is the unique daughter of B then $B \rightsquigarrow \alpha$.
- APPLICATION: if $A \rightsquigarrow \alpha, B \rightsquigarrow \beta$ and A, B are daughters only of C then $C \rightsquigarrow \alpha(\beta)$
- SEQUENCING: if $T \rightsquigarrow \tau, S \rightsquigarrow \delta$ and T, S are daughters of X then $X \rightsquigarrow \tau; \delta$
- QUANTIFYING-IN: if $NP^n \rightsquigarrow \eta, S \rightsquigarrow \delta$ and NP^n, S are daughters of X then $X \rightsquigarrow \eta(\lambda v_n.\delta)$
- REDUCTION: if $A \rightsquigarrow \alpha$ and β is reduced from α by λ conversion, then $A \rightsquigarrow B$.

An example of the parsed tree of a sentence *a farmer walks. He laughed*, by [278], is given as following

a	$\lambda P \lambda Q.([u_1 \vert].P(u_1); Q(u_1))$
$farmer$	$\lambda v.[farmer[v]]$
$a\ farmer$	$\lambda Q.([u_1].[\vert farmer(u_1)]; Q(u_1))$
$a\ farmer\ walks$	$[u_1 \vert].[\vert farmer(u_1)]; [\vert walks(u_1)]$
he_1	$\lambda P.P(\delta)(\delta = dr(ant(he)))$
he_1	$\lambda P.P(u_1)$
$he\ laughed$	$[\vert laugh(u_1)]$

$$a\ farmer\ walks.\ he\ laughed : [u_1|].[|farmer(u_1)]; [walks(u_1)]; [|laugh(u_1)]$$

Using the merging lemma leads to:

$[u_1|farmer(u_1), walk(u_1), laugh(u_1)]$

7.1.4 An accessibility and weakest-precondition calculus

Accessibility in DRT is used to handle the potential referents of a pronoun. Musken defines **acc**(u,K) as the set of accessible discourse referents from u in K. Roughly speaking, the set represent the potential scopes of a discourse referent. They are defined as following inductive rules:

(active discourse referents)

$\mathbf{adr}([u_1, \cdots, u_n|\gamma_1, \cdots, \gamma_m]) = \{u_1, \cdots, u_n\}$

$\mathbf{adr}(K_1; K_2) = \mathbf{adr}(\mathbf{K}_1) \bigcup \mathbf{adr}(K_2)$

$$\begin{array}{rcc}
\mathbf{acc}(u, \phi) & = & \varnothing \text{ if } \phi \text{ is atomic} \\
\mathbf{acc}(u, \mathbf{not}\ K) & = & \mathbf{acc}(u, K) \\
\mathbf{acc}(u, K_1 \mathbf{or}\ K_2) & = & \mathbf{acc}(h, K_1)\text{if u appeared in } K_1 \text{ otherwise } K_2 \\
\mathbf{acc}(u, K_1 \Rightarrow K_2) & = & \mathbf{acc}(u, K_1)\text{if u appeared in } K_1 \text{ otherwise } \mathbf{acc}(u, K_2) \bigcup \mathbf{adr}(K_1) \\
\mathbf{acc}(u, [u_1, \cdots, u_n|\gamma_1, \cdots \gamma_m]) & = & \mathbf{acc}(u, \gamma_i) \bigcup \{u_1, \cdots u_n\}\text{if u appeared in } \gamma_i \\
\mathbf{acc}(u, K_1; K_2) & = & \mathbf{acc}(u, K_1)\text{u appeared in } K_1 \text{ otherwise } \mathbf{acc}(u, K_2) \bigcup \mathbf{adr}(K_1)
\end{array}$$

A further development of this resolution can be seen in the verb phrase ellipsis analysis in [278].

The weakest-precondition calculus in [24, p. 27–29] concerns translating the language of boxes to (predicate) logics. A simple DRS structure can use the unselective binding lemma. On the other hand, a complex structure can be developed, using the below calculus which originates in Hoare's logic [24][p. 27]. Let **tr** be a translation function from conditions to predicate logic formulae, **wp** (weakest precondition) have a box and a first order formula as input, and a predicate logic formula as output. Thus, $\mathbf{tr}(\gamma)$ yields a truth condition of the condition γ, while $\mathbf{wp}(K, \mathbb{T})$ ($\mathbb{T}$ is the affirmative truth-only sentence), the truth condition of K. The **wp** is the reverse engine[1] that, given a box K and an output truth value t, finds the weakest condition in the predicate logic formation ϕ such that K applied with ϕ yields t. Thus, finding the semantics are equivalent to find the precondition.

Let † be the assignment function from discourse referents $\{u_1, \cdots u_n\}$ to individual variables $\{x_1, \cdots, x_n\}$ and constants as $u_n^\dagger = x_n$,
$Tom^\dagger = tom, Tim^\dagger = tim, etc$
The calculus is illustrated as

$$\mathbf{tr}(R\{\delta_1, \cdots, \delta_n\}) = R(\delta_1^\dagger, \cdots, \delta_n^\dagger)$$

$$\mathbf{tr}(\delta_1 \textbf{ is } \delta_n) = (\delta_1^\dagger = \delta_2^\dagger)$$

$$\mathbf{tr}(\textbf{not } K) = \neg(\mathbf{wk}(K, \mathbb{T}))$$

$$\mathbf{tr}(K_1 \textbf{ or } K_2) = \mathbf{wp}(K_1, \mathbb{T} \vee \mathbf{wp}(K_2, \mathbb{T}))$$

$$\mathbf{tr}(K_1 \Rightarrow K_2) = \neg\mathbf{wp}(K_1, \neg\mathbf{wp}(K_2, \mathbb{T}))$$

$$\mathbf{wp}([u_{k_1}, \cdots u_{k_n} | \gamma_1, \cdots \gamma_m], \Psi) = \exists x_{k_1} \cdots x_{k_n} . \mathbf{tr}(\gamma_1) \wedge \cdots \wedge \mathbf{tr}(\gamma_m) \wedge \Psi$$

$$\mathbf{wp}(K_1; K_2, \Psi) = \mathbf{wp}(K_1, (\mathbf{wp}(K_2, \Psi)))$$

An example of the box for the sentence A man_1 adores a $woman_2$. She_2 abhors him_1. with its box representation:

[1] You can think of it as reverse mathematics.

$[u_1\ u_2|man\ u_1, woman\ u_2, u_1\ adores\ u_2, u_2\ abhors\ u_1]$

is process as following

$\mathbf{wp}([u_1\ u_2|man\ u_1, woman\ u_2, u_1\ adores\ u_2, u_2\ abhors\ u_1], \mathbb{T})$

$= \exists x_1 x_2.\mathbf{tr}(man\ u_1) \wedge \mathbf{tr}(woman\ u_2) \wedge \mathbf{tr}(u_1\ adores\ u_2) \wedge \mathbf{tr}(u_2\ abhors\ u_1)$

(apply the second last rule)

$= \exists x_1, x_2.man(x_1) \wedge woman(x_2) \wedge adores(x_1)(x_2) \wedge abhors(x_2)(x_1)$

(apply the first rule 4 times)

[24, p. 28] also noted that accessibility and **wp** are closely related.

7.2 The translation to parameterized monads

Due to the equivalence between logics, types, and category theories as discussed in Chapter 3, we expect an equal framework for Musken's system in category theory. I provide a translated version of Musken's framework in parameterized monads as follows. The related research of the translated systems of the cDRT are chapter 3 of [65] and [16]. In comparison to those systems, this is a distinct framework underlaid by Hoare's logic.

The overall idea of this framework is similar to Muskens' viewpoint. Namely, we use programming language theories as our background research. Thus, Musken's state is similar to states in parameterized monads as an abstraction of a computer memory. Furthermore, the idea of using two states and a chain has also appeared in [24, p. 14]:

> In a similar way all other boxes can be rewritten as certain terms $\lambda i.\lambda j$, where j is a first-order formula. Of course, for practical purposes we greatly prefer the more transparent box notation and in fact it will turn out to be completely unnecessary to expand definitions.

and [24, p. 20]:

> expressions of the fragment will be equivalent to terms consisting of a chain of λs followed by an expression of the box type $s(st)$,

In the author's opinion, the idea of using two states in $\lambda i.\lambda j$ for semantics, or $s(s(t))$ for types, is equivalent to the pre- and post-states in Hoare's logic or parameterized monads and a chain of it is a specification structure in section 5.4. Alternatively, we can view a state s as a result, in which case a short notion of $s \to s \to t$ to [] is actually the continuation in the sense of [12].

The central point of composition in Muskens' framework is the interpretation of λ-calculus in category theory. It is illustrated in section 3.1.1. In addition, Muskens' four basic types are interpreted as follows: e, t are being kept the same; registers or storage are stack programs; and states are the state category in the parameterized monad as in section 5.2.

In addition, moving Musken's states to parameterized monad states yields an advantage in our typing declaration with an additional space for typing declaration. I used it to encode category in the sense by [278] as in the below lexical items. Hence, we have a richer typing declaration system than Musken's type. In the author's opinion, this idea can be developed further to include the categorical grammar.

Semantically, section 4.3 shows how to translate the DRS into the state monad. Indeed, the interpretation of states by Muskens as $\lambda v. \bigvee(v)(i)$ is the definition of the state monad. In addition, this framework extends an expressive power of a monadic term and focuses on state transitions under the same formulae. This has an advantage of an explicit governing or subsentential management of registers. In detail, the semantics are interpreted as follows.

The interpretation rules of boxes are defined by using the dynamic logical operators, implication and negation ($\Rightarrow, \sim$), in section 7.3.1 below. The four axioms are kept the same and interpreted as primitive functions.

•	$R_{\tau_1\to\cdots\tau_n\to t}(\alpha^1_{s\to\tau_1}\cdots\alpha^n_{s\tau_n}) : \mathbb{M}\ s_1\ s_1\ t$
	$R_{\tau_1\to\cdots\tau_n\to t}(\alpha^1_{s\to\tau_1}\cdots\alpha^n_{s\tau_n}) =_{def} \lambda s_1.R(\alpha^1(s_1),\cdots,\alpha^n(s_n))$
•	$K \Rightarrow K^{'} =_{def} K \Rightarrow K^{'}$
•	$\neg K =_{def} \sim K$
•	$\alpha = \beta : \mathbb{M}\ s_1\ s_1\ t$
	$\alpha = \beta =_{def} \lambda s_1.\lambda \hat{x}.\alpha(\hat{x})(s_1) = \lambda \hat{x}.\beta(\hat{x})(s_1)$

Table 7.1: Logical operators on boxes.

R is a relation, and K is a box in a sense of DRT [56]. In short, on the left of the | is a context which is a list of *reference markers*. Normally, it is represented as a list of variables which act as discourse references. The right of the | describe the constraints. The box interpretation in parameterized monads is

$[v_1\cdots v_m \mid k_1\cdots k_n] : \mathbb{M}\ s_1\ s_2\ t$
$[v_1\cdots v_m \mid k_1\cdots k_n] =_{def} \lambda s_1 \cdot s_1\,[v_1,\cdots,v_m]\,s_2 \wedge k_1(s_1) \wedge \cdots k_n(s_1)$

Table 7.2: Interpretation of a box.

where $s_1[v_1,\cdots,v_m]s_2$ means s_2 is different with s_1 at most in values of $v_1,\cdots,v_m$.

In addition, the box sequencing $K;K^{'}$ is interpreted as the dynamic conjunction ; in the following section. The merging lemma of boxes is described in the following. if $v_1^{'}\cdots v_m^{'}$ are not in $k_1,\cdots,k_l$,

$$\boxed{[v_1 \cdots v_k | k_1, \cdots, k_l]; [v_1^{'} \cdots v_m^{'} | k_1^{'}, \cdots, k_n^{'}] =_{def} [v_1 \cdots v_k v_1^{'} \cdots v_m^{'} | k_1 \cdots k_l, k_1^{'} \cdots k_n^{'}]}$$

Table 7.3: Merge two boxes.

The truth and entailment are:

- A formula K is true at s_1 if there exists an s_2 such that $K : \mathbb{M}\ s_1\ s_2\ \top$. If we write K is true, it means that $\forall s_1.\exists s_2.K : \mathbb{M}\ s_1\ s_2\ \top$
- K entails $K^{'}$ at s_1 or $K \vDash_{s_1} K^{'}$ if and only if, K is true at i implies $K^{'}$ is also true at i.

Lexically, the author provides basic lexical examples of a fragment of English by [278] in parameterized monads as follows. The other five additional rules for lexical items are kept the same.

Expression	λnotation	Type
a^n	$\lambda P^{'}.\lambda P.([u_n \vert]; P^{'}(u_n); P(u_n))$	$\mathbb{M}\ s_1\ (s_2 \wedge s_1[u_n]s_2)\ (e \to t) \to \mathbb{M}\ s_2\ s_3\ (e \to t) \to \mathbb{M}\ s_1\ s_3\ \mathbf{NP}$
no^n	$\lambda P^{'}.\lambda P.[\vert \mathbf{not}([u_n \vert]; P^{'}(u_n); P(u_n))]$	$\mathbb{M}\ s_1\ s_2\ (e \to t) \to \mathbb{M}\ s_2\ s_3\ (e \to t) \to \mathbb{M}\ s_1\ s_3\ (e \to t)$
$every^n$	$\lambda P^{'}.\lambda P.[\vert([u_n \vert]; P^{'}(u_n) \Rightarrow P(u_n))]$	$\mathbb{M}\ s_1\ s_2\ (e \to t) \to \mathbb{M}\ s_2\ s_3\ (e \to t) \to \mathbb{M}\ s_1\ s_3\ \top$
$Mary^n$	$\lambda P.P(Mary)$	$\mathbb{M}\ s\ (s, m)\ \mathbf{NP}$
he_n	$\lambda P.(P(\delta)), \delta = ant(he_n)$	$\mathbb{M}\ s\ s\ \mathbf{NP}$
e_n	$\lambda P.P(v_n)$	$\mathbb{M}\ s\ s\ e$
who	$\lambda P^{'}.\lambda P.\lambda v.P(v); P^{'}(v)$	$\mathbb{M}\ s_1\ s_2\ (e \to t) \to \mathbb{M}\ s_2\ s_3\ (e \to t) \to$ $\mathbb{M}\ (s_2 \vee s_3)\ s_4\ e \to \mathbb{M}\ s_1\ s_4\ \top$
$farmer$	$\lambda v.[\vert farmer\ v]$	$\mathbb{M}\ s\ s\ \mathbf{N}$
$stink$	$\lambda v.[\vert stinks\ v]$	$\mathbb{M}\ s\ s\ \mathbf{VP}$
$love$	$\lambda Q.\lambda v.Q(\lambda v^{'}.[\vert v\ loves\ v^{'}])$	$\mathbb{M}\ s_1\ s_2\ (e \to t) \to \mathbb{M}\ s_2\ s_2\ e \to \mathbb{M}\ s_2\ s_2\ e \to \mathbb{M}\ s_1\ s_2\ \mathbf{TV}$
$doesn't$	$\lambda P.\lambda Q.[\vert \mathbf{not}(Q(P))]$	$\mathbb{M}\ s_1\ s_2\ (e \to t) \to \mathbb{M}\ s_2\ s_3\ e \to \mathbb{M}\ s_1\ s_3\ \top$
if	$\lambda p.\lambda q.[p \Rightarrow q]$	$\mathbb{M}\ s\ s\ \mathbf{S} \to \mathbf{S} \to \mathbf{S}$

Table 7.4: An English grammar example

Basically, the indefinite *a* introduces a new discourse reference to the discourse as u_n, and the quantifiers—such as *no, every*—access and evaluate the discourse variables. A sketch of an analysis of the scope of the indefinite as variables (not as quantifiers) in monads was discussed in section 4.3.5. A further elaborated analysis can be seen in [17, 16].

The compositional rules to interpret this fragment of English in monads is shown in the next section. How to interpret this fragment in parameterized monads, providing the categorical semantics to the DRT, is explained through the pilot study of the interpretation of the donkey anaphora in parameterized monads.

Finally, [24, p. 28] also noted that the accessibility and **wp** are closely related. This idea is close to the author's analogy between scope-taking and proof search discussed below. In addition, the relation between the weakest-precondition calculus or Dijkstra's programming logics and Hoare's logic is explained in [282, p. 45–46]. Basically, Dijsktra's logic is interpreted in Hoare's logic by Hilbert's ϵ-calculus. The recent study of the ϵ-calculus can be found in [283].

Alternatively, Muskens' calculus and unselective binding lemma concerns the semantics of the existence quantifiers. Hence, it can be seen as the interaction of static formulae and dynamic information states. BHK's interpretation and [181] show that both are equal, and that they also provide a general schematic interpretation. In addition, by using the above interpretation of Dijsktra's logic in Hoare's logic, the weakest-precondition calculus is the structured analysis of parameterized monads.[2] [3]

7.3 Dynamic semantics in parameterized monads

The idea of interpreting dynamic semantics in monads was pioneered by Shan and Unger [63, 15] and subsequently advocated by Charlow and Grove [17, 16]. [17, p. 34-85] claims to interpret dynamic semantics as side-effects, using continuations and a combination of states and sets to interpret the semantics. I also follow his approach but interpret via a diffident framework of category theory, in particularly parameterized monads, and interpret the donkey phenomena in this framework. The parameterized monad has a property that the Charlow's continuation monad approach to the donkey anaphora [12] lacks: a clear framework to combine state and non-deterministic side-effects. According to [17, p.117], the two side-effects are required to provide a semantic to interpret the donkey anaphora. This framework provides a semantics to combine two side-effects as below. The basic definition of dynamic semantics is given in section 2.2.4. I sketch the combination of two side-effects, and an interpretation of the donkey anaphora, throughout the rest of this chapter.

[2] The weakest-precondition calculus seems close to the contemporary research of Dijkstra's monad in [284, 249]

[3] The idea of accessibility and swapping technique seems similar, too.

The most recent related research on dynamic semantics is [19] with dynamic category semantics and [164, 46] with type-theoretic dynamic logic (TTDL). According to [10], an advantage of TTDL is that it imposes no requirement for non-standard notions of binding and scope. This point is crucial to support our *swapping technique* that advocates the post-evaluation of linguistic variables and the analogy between the evaluation order and the mathematical proof. Furthermore, the logic which is developed in [46] and discussed in [10, p. 121–125] is similar to this section on the interpretation of logical connectives. However, the difference is that we interpret the implication directly[4] [5] whereas [46] has to use the continuation to interpret the implication (i.e. $A \to B := \neg(A; \neg B)$ or $A \to B := \neg(A \land \neg B)$). Lebedeva's use of continuation leads to a double negation interpretation of the donkey anaphora as

$$\neg\exists x.\mathbf{farmer}(x) \land \exists y.\mathbf{donkey}(y) \land \mathbf{own}(y,x) \land \neg(\mathbf{beat}(y,x))$$

This is a classical-logic interpretation of the phenomenon, according to [149, 214]. On the other hand, I approach the problem through constructive

[4] The transition technique avoids this representation by using the analogy between a formula's interpretation and changing, i.e. updating, in states as transition semantics [181].

[5] We can think of the implication as the intuitionistic implication in [257]. The implication is well aligned in this framework, since we can think of the record type in [259] as the team semantics in [153]. The interpretation of record type to category theory is achieved by using fibred categories in [261].

logic. The problem lies in the interpretation of the implication. In the constructive-logic framework, the implication rules are interpreted using Yoneda's lemma in [31]. Intuitively, the lemma means that we conclude $A \Rightarrow B$ if every instance of A implies an instance of B, rather than the double negation that every instance of $\neg B$ implies $\neg A$.[6].

According to [43, 27], the dynamic interpretation for a formula ϕ in a given state g is $g[\![\phi]\!]h$, where g and h are the pre- and post-dynamic conditions. In our parameterized monads framework, g and h are interpreted as pre- and post-states. Hence, the dynamic interpretation of ϕ under the pre-condition g is the parameterized monad $\mathbb{M}\ g\ h\ \phi$. A formula ϕ is true, relative to a state g, if there exists a state h such that $\phi : \mathbb{M}\ g\ h\ \top$.

For a general formula ϕ, if the pre-condition g is not mentioned, the dynamic semantic interpretation of the formula, i.e. $[\![\phi]\!]$, is $\{\langle g, h\rangle : g[\![\phi]\!]h\}$. This means that we generate all suitable pre-conditions and related post-conditions for ϕ. Thus, the parameterized monadic type for $[\![\phi]\!]$,i.e $\eta(\phi)$, is

$$\mathbb{M}\ s_1\ s_2\ \phi,$$

where $s_1 = \bigcup g, s_2 = \{\langle g, h\rangle | g \in s_1\}$. g, h are defined as in traditional dynamic semantics.

[6] Informally, this idea is also in Hilbert's study of systems for formalising mathematics.

The linguistic term ϕ has a traditional interpretation in dynamic semantics by [275]. According to [44, p. 11–12] and [27, p. 4–5], dynamic predicate logic uses constructive logic and artificial intelligence as backgrounds. Furthermore, their interpretation of the meaning of a sentence is to change the information of the interpreter. The information is stored in states, and represented as noun phrases. Consequently, verbs are interpreted as predicates. Hence, the meaning of a sentence is a process of verifying the data structure, which is stored in states. This means that a sentence is true if the output is sound, and an empty output indicates that the sentence is false. Not all linguists may agree with that hypothesis, but we adopt this hypothesis in our research.

Thus, I formalize the interpretation of NP and VP in the parameterized monad framework as below:

$$[\![NP]\!] : \mathbb{M}\; s_1\; s_2\; \mathbf{NP}$$

$$[\![NP]\!] = \lambda s_1.s_2 = operator(s_1) \wedge return\; \eta(NP)$$

Where *operator* is an additional operator on the structured **NP**, defined inductively by the inductive definition of **NP** as the above grammar in Section 7.3. In the above grammar, s_2 is being defined by adding a new discourse reference to s_1 if **NP** is a proper name, and $s_2 = s_1$ otherwise. On the other hand, we follow a tradition of interpreting verb phrases as tests on state in dynamic semantics. Formally,

$$[\![VP]\!] : \mathbb{M}\ s_1\ s_2\ \top$$

$$[\![VP]\!] = \lambda N\ s_1 . s_2 = s_1 \wedge \mathit{return}\ \odot \ \text{if NP VP else } {}^{7} s_2 = \{\} \wedge \mathit{return}\ \bot$$

where $\odot$ is a singleton element of the type of the truth condition $\top$.

In the above declaration, we have a type declaration and a meaning declaration. The type declaration is an abstract representation of a term. In this example, the monadic notion $\mathbb{M}$ means we lift linguistic terms, which are represented by λ terms, into their evaluated environments of pre- and post-conditions. In the monads in [8], for example, the meaning of an utterance is lifted to be interpreted with its state or situation by using the state monad in [15].

However, we should keep in mind a particularly important property: monads preserve the static meaning of a term when we lift it to the monadic space. The static meaning is expressed by the meaning declaration. In this formalization, static meaning is expressed by the λ terms. Related research

[7] That is by verifying, such as possible world semantics, that the sentence is true

by Barker [12] and Charlow [17, p. 60–61] shows this concept in the towering notions.

7.3.1 Linguistic logical operators in parameterized monads

Definition a dynamic **conjunction**, **;** , is defined in parameterized monads as a traditional interpretation in DRT in [56]. $\llbracket a;b \rrbracket = \llbracket a \rrbracket$ and $\llbracket b \rrbracket$, i.e $\llbracket a \rrbracket \wedge \llbracket b \rrbracket$. Thus, the interpretation of the conjunction in parameterized monads, which the pre- and post-states are primary, is

$$\llbracket ; \rrbracket : \mathbb{M}\ s_1\ s_2\ \top$$

$$\llbracket ; \rrbracket = \lambda\ \mathbf{l}\ \mathbf{r}\ s_1.s_2 = \bigcup_{s \in \mathbf{l}\ s_1} \mathbf{r}\ s \wedge \odot$$ [8]

Definition a **negation**, ~, of a formula is defined through the absence of proper output, i.e. s_2, in the formula's typing declaration. Thus

$$\llbracket \sim\ \phi \rrbracket : \mathbb{M}\ s_1\ s_2\ \eta(\phi)$$

$$\llbracket \sim\ \phi \rrbracket = \lambda\ s_1.s_2 = s_1\ \wedge \neg \exists s.\llbracket \phi \rrbracket : \mathbb{M}\ s_1\ s\ \top$$

Definition the **implication**, ⇒, such as **if**, between two formulae ϕ and ψ, is defined if every output state s of ϕ, when taken as an input state of ψ, produces a proper output. Formally

[8] This interpretation of conjunction follows Barker's polymorphic typed interpretation of a conjoinable coordinator [166, p. 16]

$[\![\phi \Rightarrow \psi]\!] : \mathbb{M}\ s_1\ s_2\ \top$

$[\![\phi \Rightarrow \psi]\!] = \lambda s_1.s_2 = \{\langle g, h\rangle | \forall g \in s_1.g = h \wedge \forall k.[\![\phi]\!] : \mathbb{M}\ g\ k\ \top \wedge \exists k^{'} [\![\psi]\!] : \mathbb{M}\ k\ k^{'}\ \top\}$

This definition is equal to the dynamic implication ⇉ in [16, p. 65].

Definition the **disjunction**, $\vee$, is defined as a selection or choice operator of assignment functions. It is random and has a *nondeterministics* property. For example, if the state s_1 consists of four computer scientists o, p, q, r and we write $s_1[x]s_2$ as "s_2 is different at most with s_1 by the variable x," then $s_2 = (s_1 \wedge o^x) \vee (s_1 \wedge p^x) \vee (s_1 \wedge q^x) \vee (s_1 \wedge r^x)$. a^x means that we *update* the value of x to a.

Definition the **truth** of a formula ϕ, in a traditional interpretation in dynamic semantics, is defined relative to a given state s_1, i.e. ϕ is true relative to s_1 if there exists s_2 such that $\phi : \mathbb{M}\ s_1\ s_2 \top$.

Linguistically, we represent an **indefinite**, such as English **a, an**, as an existence $\exists$ variable [92], and the disjunction as an assigning function which provides a value to the variable. The indefinite has a prestate s_1 as a collection of entities that satisfy properties of the indefinite and the post state s_2 differs at most at one variable with s_1. It is written formally as

$[\![\exists x.\phi]\!] = [\![[x];\phi]\!]$

where

$[\![[x]]\!] : \mathbb{M}\ s_1\ s_2\ \top$

$[\![[x]]\!] = \lambda s_1. \bigcup_{g \in s_1} s2 = \{\langle g, h\rangle | g[x]h\} \wedge \odot$

and ; is the conjunction operator. Let us illustrate the point by way of an example.

a scientist $: \mathbb{M}\ s_1\ s_2\ \mathbf{S}$

a scientist $= \lambda\mathcal{P}\ s_1.\mathbf{scientist}(x) \wedge \mathcal{P}\ (x) \wedge s_1[x]s_2 \wedge \eta(x)$

where **S** is a type of scientists. A general treatment of an indefinite noun phrase **NP** is

$[\![\mathbf{a\ NP}]\!] = [\![[x];\mathbf{NP}(x)]\!]$

NP, in the above example is **scientist**.

Furthermore, the linguistic **universal quantifications**, such as ∀, are interpreted as the implication ⇒ in parameterized monads. Formally, universal quantification is written as

$[\![\forall x.\phi]\!] = [\![[x] \Rightarrow \phi]\!]$

Thus, a sentence

every NP VP

is interpreted as

$\forall x.\mathbf{NP}(x) \Rightarrow \mathbf{VP}(x)$

7.3.2 Dynamic predicate logic in parameterized monads

This section sketches an interpretation of dynamic predicate logic (DPL) [27] in the parameterized monads with the **set** category. The language is simple but powerful. For example, [285] shows how the context change potential in [111] is interpreted in dynamic predicate logic. In addition, a related research by [235] shows an interpretation of DPL in the dependence logic.

The context Γ includes the domain of individuals and the interpretation function I, i.e $\Gamma = D_e \bigcup I$ where $I(R) \subseteq D^n$ if R is a n-ary relation.

$[\![R(x_1, x_2, \cdots, x_n)]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B}$

$[\![R(x_1, x_2, \cdots, x_n)]\!] = \mathbb{T}$ if and only if $G = H$ and $\forall g \in G, R(g(x_1), g(x_2), \cdots, g(x_n)) \in I(R)),$

$[\![x = y]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B}$

$[\![x = y]\!] = \mathbb{T}$ if and only if $G = H$ and $\forall g \in G, g(x) = g(y)$.

$[\![\neg\phi]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B}$

$[\![\neg\phi]\!] = \mathbb{T}$ if and only if $\forall G^{'} \subseteq G, [\![\phi]\!] : \mathbb{M}\ \ G^{'}\ \ H\ \ \mathbb{F}$.

$[\![\phi \wedge$[9]$\psi]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B}$

$$[\![\phi \wedge \psi]\!] = \mathbb{T} \text{ if and only if } \begin{cases} [\![\phi]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B} \\ [\![\phi]\!] = \mathbb{T} \end{cases} \text{and} \begin{cases} [\![\psi]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B} \\ [\![\psi]\!] = \mathbb{T} \end{cases}$$

$[\![\phi \vee \psi]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B}$

$[\![\phi \vee \psi]\!] = \mathbb{T}$ if and only if there exists $G^{'}$ and $G^{''}$ such that $G = G^{'} \bigcup G^{''}$ and $\begin{cases} [\![\phi]\!] : \mathbb{M}\ \ G^{'}\ \ H\ \ \mathbb{B} \\ [\![\phi]\!] = \mathbb{T} \end{cases}$ or $\begin{cases} [\![\psi]\!] : \mathbb{M}\ \ G^{''}\ \ H\ \ \mathbb{B} \\ [\![\psi]\!] = \mathbb{T} \end{cases}$

$[\![\phi \rightarrow$[10]$\psi]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B}$

$[\![\phi \rightarrow \psi]\!] = \mathbb{T}$ if and only if $\forall G^{'} \subseteq G$ if $\begin{cases} [\![\phi]\!] : \mathbb{M}\ \ G^{'}\ \ H\ \ \mathbb{B} \\ [\![\phi]\!] = \mathbb{T} \end{cases}$ then $\begin{cases} [\![\psi]\!] : \mathbb{M}\ \ G^{'}\ \ H\ \ \mathbb{B} \\ [\![\psi]\!] = \mathbb{T} \end{cases}$.

[9] The notion is the dynamic conjunction

[10] The notion is the dynamic implication

Table 7.5: Dynamic predicate logic in parameterized monads.

In the case of negation and disjunction, there are alternative definitions based on the truth condition in [235]. In the author's opinion, this is the same truth condition (as below) and dynamic condition (as above) of a sentence in [117]. More clearly,

$$\begin{cases} [\![\neg\phi]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B} \\ [\![\neg\phi]\!] = \mathbb{T} \end{cases} \text{if and only if} \begin{cases} [\![\phi]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B} \\ [\![\phi]\!] = \mathbb{F} \end{cases}.$$

$$\begin{cases} [\![\phi \vee \psi]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B} \\ [\![\phi \vee \psi]\!] = \mathbb{T} \end{cases} \quad \text{if} \quad \text{and} \quad \text{only} \quad \text{if} \quad \begin{cases} [\![\phi]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B} \\ [\![\phi]\!] = \mathbb{T} \end{cases} \quad \text{or}$$

$$\begin{cases} [\![\psi]\!] : \mathbb{M}\ \ G\ \ H\ \ \mathbb{B} \\ [\![\psi]\!] = \mathbb{T} \end{cases}.$$

As we are focusing on the dynamic aspect, we adopt these interpretations for negation and disjunction as above. We do not represent the inclusion, as discussed in [286, 117] to the framework. The usefulness of these choices is shown in the interpretation of time in [235].

Further examples of how to translate between static semantics (or propositional semantics) and dynamic semantics is given in [287, 288, 273], and in [116, Chapter 10]. In addition, an update function, an important part of dynamic semantics (as discussed in [285]), is given in parameterized monads in [34, Chapter 8].

7.3.3 Combining state and set monads in parameterized monads

In this section, the author will show how we combine a state's related monad in parameterized monads. They include the state monad and reader monad. The reader monad is a state monad that does not process the explicit state. Its advantage is to reduce computation time, at the cost of having less expressive power than the state monad.

In this example, the author will combine state and reader monads with power set monads. Power set monads are used to describe subsets. The declaration includes the lifting, i.e. η, and passing, i.e. $\star$, rules. The combination is similar to the **reader.set** and **state.set** monad in [17]. However, we are working on parameterized monads, an extension of monads. The combination of **reader** and **set** monads is interpreted as

$a^{\eta} : \mathbb{M}\ s_1\ s_2\ (\alpha \to t)$

$a^{\eta} = \lambda s_1 . s_2 = s_1 \wedge \{a\}$

$m \star \pi : \mathbb{M}\ s_1\ s_2\ \alpha \to (\alpha \to \mathbb{M}\ s_2\ s_3\ \beta) \to \mathbb{M}\ s_1\ s_3\ \beta$

$m \star \pi = \lambda s_1 . s_2 = s_1 \wedge s_3 = s_2 \wedge \bigcup_{a\ \in\ m\ s_1, v\ \in\ \text{the set of variables of } s_2} \pi(a/v)\ s_2$

Similarly, the combination of the **state** and **set** monads is interpreted as

$\mathbb{M}\alpha = \lambda s . (\alpha \times s)$

$a^{\eta} : \mathbb{M}\ s\ (\mathcal{P}\ (\alpha \times s))\ \alpha$

$a^{\eta} = \lambda s_1.s_2 = \{\langle a, s\rangle\}a$

$m \star \pi : \mathbb{M}\ s_1\ s_2\ \alpha \to (\alpha \to \mathbb{M}\ s_2\ s_3\ \beta) \to \mathbb{M}\ s_1\ s_3\ \beta$

$m \star \pi = \lambda s_1.\pi[a/v]s_2$

where α is a set. $a : \alpha$ means that a is an element of that set. An example of a set is the set of individuals of a class at school. A set monad is the power set of its elements in [289] and [17]. In other words, if $a : \alpha$, then the monadic dimension of α is the power set of α, i.e. all subsets of the set α. It has a following formal definition:

$\mathbb{M}\ s_1\ s_2\ \alpha = \lambda s_1.s_2 = s_1 \wedge \mathcal{P}(\alpha)$

Where $\mathcal{P}(\alpha) = \{f : \alpha \to t\}$. For example, if $\alpha = \{1, 2, 3\}$ then

$\mathcal{P}(\alpha) = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

Thus, we can write $\mathbb{M}\alpha = \mathbb{M}\ s_1\ s_1\ \alpha$, ignoring the states for demonstration purposes. We can define the set monads as

$\mathbb{M}\alpha = \mathcal{P}(\alpha)$

$a^{\eta} : \mathbb{M}\alpha$

$a^{\eta} = \{a\}$

$m \star \pi : \mathbb{M}\alpha \to (\alpha \to \mathbb{M}\beta) \to \mathbb{M}\beta$

$m \star \pi = \bigcup_{a \in m} \pi[a/v]$

We use the set monad to describe the subset relation in linguistics. For example, suppose that we have a set of individuals e. Now we want to describe a subset of individuals who are **man**. In set-theoretic interpretation, we use the subset notion to say that a set of **man** belongs to the set of individuals. In the author's opinion, it is essential to interpret indefinite or definite descriptions such as **a man** or **the man**. For example, we can model the indefinite **a man** as a subset of the set of **man** which, in turn, is also a subset of the set of individuals as $\mathbf{a\ man} : \mathbf{man} \wedge \mathbf{man} \subset e$.

Let us illustrate the idea by explaining the truth value of a sentence through the compositional principle in Charlow's sentence
John meets a man

If we define **a man** in the set monad as $\mathbf{a\ man} = \{x : e | \mathbf{man}(x)\}$, then the sentential compositional derivation process of the truth value is
$[\![\mathbf{John\ met\ a\ man}]\!] =$
$\mathbf{John}^{\eta} \star \mathbf{met}^{\eta} \star \mathbf{a\ man}^{\eta} =$
$(\mathbf{John}^{\eta} \star \mathbf{met}^{\eta}) \star \mathbf{a\ man}^{\eta} =$
$(\mathbf{J}^{*} \star \mathbf{met}^{*}) \star \{x : e | \mathbf{man}(x)\} =$
$(\mathbf{J}^{*} \star \{\lambda x.\lambda y.\mathbf{met}^{*}(x, y)\}) \star \{x : e | \mathbf{man}(x)\} =$
$\{\lambda y.\mathbf{met}(J^{*}, y)\} \star [\{x : e | \mathbf{man}(x)\}]_{y} =$
$[\{x : e | \mathbf{man}(x)\}]_{y} \star \{\lambda y.\mathbf{met}(J^{*}, y)\} =$
$\{\mathbf{met}(J^{*}, \{x : e | \mathbf{man}(x)\})\} =$
$\{\mathbf{met}(J^{\star}, x) | \mathbf{man}(x)\}$

In addition, a comprehensive monadic set theoretic interpretation of a sentence is referred to [16].

7.3.4 Another example of how the compositional principle acts in parameterized monads

This section demonstrates how the sentences

A man walks in the park. He whistles.

in [42] are interpreted in parameterized monads. Firstly, we define illustrated lexical semantics in parameterized monads. A proper name is interpreted as a constant, adding an additional element to the discourse. Thus, a personal name, *Alice* for example, has the following formalization

$$[\![Alice]\!] : \mathbb{M}\ s_1\ s_2\ \top$$

$$[\![Alice]\!] = \lambda\mathcal{P}\ s_1.s_2 = (s_1, A) \wedge \mathcal{P}(A)$$

As previously discussed, an indefinite, for example *a man*, is interpreted through set monads as

$$a\ man : \mathbb{M}s_1s_2\mathbf{Man}$$

a man $= \lambda s_1.s_2 = s_1 \wedge \{x : e|\mathbf{man}(x)\}$.

A general pronoun is treated as reader monad in [15]. Hence, its formalization in parameterized monads is

$he : \mathbb{M} s_1 s_2 \mathbf{man}$

$he = \lambda \mathcal{P}.\lambda s_1.s_2 = s_1 \wedge \mathcal{P}(sel\ s_1)$

$[\![she]\!] : \mathbb{M}\ s_1\ s_2\ \mathbf{woman}$

$[\![she]\!] = \lambda \mathcal{P}\ s_1.s_2 = s_1 \wedge \mathcal{P}(sel\ s_1)$.

A verb is interpreted as a test in dynamic semantics because its purpose is to verify the given states. Thus, a verb *walk*, for example, is formalised as

walk $: e \to e \to \mathbb{M} s_1 s_2 t$

walk $= \lambda x_1.\lambda x_2.\mathbf{walk\_in}(x_1, x_2)$.

A preposition in our formalization describes a subset. Thus, a preposition *in the park*, for example, is formalised as

$in\ the\ park : e \to e$

in the park = $\lambda x_2.\mathbf{park}(x_2)$

The compositional rules are applied to the phrase as

a man walk =

$\lambda s_1.\{x : e|\mathbf{man}(x)\} \wedge s_1[x]s_2 \wedge \lambda x_2.\mathbf{walk}(x_2, x)$

Where $s_1[x]s_2$ means s_2 differs from s_1 at most by the variable x. Thus, the sentence *a man walks in the park* has the following interpretation

$\lambda s_1.\{x : e|\mathbf{man}(x)\} \wedge s_1[x]s_2 \wedge \lambda x_2.\mathbf{walk}(x, x_2)(\lambda x_2.\mathbf{park}(x_2))$
$= \lambda s_1.\{x : e|\mathbf{man}(x)\} \wedge s_1[x]s_2 \wedge \lambda x_2.\mathbf{park}(x_2) \wedge \mathbf{walk}(x, x_2)$
$= \lambda s_1.\lambda x_2.\{x : e|\mathbf{man}(x)\} \wedge s_1[x]s_2 \wedge \mathbf{park}(x_2) \wedge \mathbf{walk}(x, x_2).$

The sentence *he whistles* is interpreted by substituting the variable $\mathcal{P}$ in the formalisation of the pronoun *he* to the predicate *whistle*:

$\lambda s_1.\mathbf{whistle}(sel\ s_1)$

Thus, the concatenation of the two sentences

A man walks in the park. He whistles.

is

$$\lambda s_1.\lambda x_2.\{x : e|\mathbf{man}(x)\} \wedge s_1[x]s_2 \wedge \mathbf{park}(x_2) \wedge \mathbf{walk}(x, x_2) \wedge \lambda s_1^{'}.\mathbf{whistle}(sel\ s_1^{'})$$

For the sake of simplification, we assume that $s_1^{'} = s_2$. Interestingly, the relation between these two states can be further analysed using the weakest-precondition calculus in [24]. Hence, we can combine the two sentences as

$$\lambda s_1.\lambda x_2.\{x : e|\mathbf{man}(x)\} \wedge s_1[x]s_2 \wedge \mathbf{park}(x_2) \wedge \mathbf{walk}(x, x_2) \wedge \mathbf{whistle}(sel\ s_2)$$

7.4 The donkey anaphora in parameterized monads

7.4.1 The definition of the problem

According to [290], the donkey anaphora phenomenon has existed since the middle ages. It is important in linguistics because it expresses the linguistic property that the latter part of a sentence depends on its preceding parts. It is called the *progressive conjunction* in [86], or the *internal dynamic* in [27]. Unger's state monad approach [15] cannot model the donkey anaphora because the state (in the sense of Chapter 3) is fixed during the sentential analysis. We are going to formalize the phenomenon in the parameterized monads framework by allowing the state, in the sense of Chapter 5, changing during subsentence analysis. This parsing or formalization also means that parameterized monads have an expressive power compatible with the type-theoretical and dynamic predicate-logic frameworks in [26] and [27], respectively.

The problem that makes the illustrated sentence important is that it shows a mismatch between the natural-language expressions and their logical interpretation. The interpretations of linguistic expressions are more flexible by reading with any order whereas the logical semantics entails strict adherence to a mechanical process. A detailed analysis of this sentence is found in [42]. Recent research with the dynamic approach to formalize the donkey anaphora has been carried out in [291, 43].

In general, the problem is associated with the scope reading of a pronoun. The problem is formulated in [43], formulated with a new dynamic framework in [291]. In comparison with their approaches that use inquisitive semantics, the parameterized monads provide an alternative modularity and compositional approach to the problem.

Return to the definition of the problem: according to [43], the phenomenon, in an English sentence, is formally stated as below

$\mathbb{Q}(\cdots NP_x \cdots)(\cdots it_x/them_x \cdots)$ such that

- $\mathbb{Q}$ is a quantifier such as *every, most, always, some*
- The element in the first bracket is the nuclear scope of the quantifier
- NP_x is an indefinite which acts as an antecedent.
- The pronoun $it_x/them_x$ is the referencing expression for the antecedent.

An example of the phenomenon is the sentence below

every farmer who owns a $donkey_x$ beats it_x

Or its variation as

If a $farmer_x$ owns a $donkey_y$, he_x beats it_y.

The problem with the logical interpretation is that the variable x in the consequence clause of the sentence does not bind to the $farmer_x$ in conditional clause. Hence, it triggers a problem of variables in linguistics as discussed in section 2.1.1.

7.4.2 The compositional dynamic semantic interpretation of the problem

From above construction, the formalization of the donkey sentence

Every farmer who owns a donkey beats it.

is straightforward by using the cDRT framework (i.e *Sentence* → *DRS* → Musken's logic of change → parameterized monads) as

$\forall x.(\mathbf{farmer}(x); \exists y.\mathbf{donkey}(y); \mathbf{own}(x,y) \Rightarrow \mathbf{beat}(x,y))$.[11]

The quantifiers ∀, ∃ are interpreted in the linguistic logical operators section. Generally speaking, the ∀ quantifier means the uniqueness condition such as in [292]. The ∃ quantifier is similar to the discourse referent in a box in Musken's framework.

From the formalization, we can process the second step, namely varying the scope-binding of variables or performing the evaluation order. The idea of using multiple stages to derive the semantics of a sentence is also developed in [209, p. 2] by analysis of Postal, Reinhart, and B*ü*ring, in [17], and in chapter 4 of [6]. In monads, multi-stage derivation manifests through multiple intermediate languages en route to the denotational semantics [33].

In this framework, I define the *swapping technique* for the second stage, to resolve the scope of linguistic expressions. The idea behind this technique is taken from mathematical logic with BHK's interpretation (explained below) of variables in a mathematical formula.[12] Linguistically, the idea of taking the scope of an indefinite freely after syntactic analysis can also be found in [17].

[11] For the complete formalized fragment of English, see the cDRT section in 7.1.

[12] I note that we are working with the domain of linguistic expressions.

This idea is similar to the idea of changing a variable's scope using delimited control operators *shift, reset* in [293, 75], or by type lifting in [216]. Since the technique is not based on syntactic analysis, it lacks the analytic rigour of syntactic theory, but it is more abstract than that. The greater abstraction allows us to catch and interpret the variable inside and outside a scope more reasonably. Thus, the formalized sentence is transformed into

$$\forall x.\forall y.(\mathbf{farmer}(x);\mathbf{donkey}(y);\mathbf{own}(x,y) \Rightarrow \mathbf{beat}(x,y))$$

This technique is also called Egli's theorem in [44, 266, 294], where it is used to swap the location of variables and, consequently, for pronoun resolution in the formalization process of a sentence. The theorem states that an occurrence of the existence variable in the left-hand side of the implication formula must also appear in the right-hand side. Formally, it is written as

$$(\exists x.\phi) \rightarrow \psi \Leftrightarrow \exists x.(\phi \rightarrow \psi)$$

This technique is also called the *modified version for dynamic conjunction* in [43], where it is expressed as

$$(\exists x.\phi);\psi \Leftrightarrow \exists x.(\phi;\psi)$$

However, we are defining a new theoretical framework to interpret the phenomenon. The → in the above formulae is the first-order logic implication, whereas we are using our own definition of the dynamic implication, ⇒.

7.4.2.1 The BHK interpretation

The supporting idea behind our reasoning is the Brouwer–Heyting–Kolmogorov (BHK) interpretation in mathematical logic [295, 296, 297, 261]. The interpretation means that a quantification of a variable, for example the universal quantification $\forall.x$, is either interpreted in the surrounding formula, or transposes x to the contexts (in some areas called an environment or state) and then reused in another formula by bringing x back. From a computing perspective, this is equivalent to avoiding analysis of a variable in a program's expression, and instead storing it in memory for later recall if requested.

Roughly speaking, the idea of transposing the variable to a context is the dependent type in [130], which introduces a dependent term (or variable) for a type in an internal language in a typing declaration. The idea of bringing a variable from the context back to a formula has also appeared in the formation rule of the Π type to capture the λ-abstraction formulae in type theory in [99]. Hence, the ability to reuse and refer to the variable under abstraction is the projection rule π_1, π_2. [86], for example, calls this property a progressive conjunction in linguistics.

In this dissertation, we replace the typed construction notions, such as Π or Σ, by quantification notions such as $\exists$ or $\forall$.[13]. Hence, we can rewrite and change the scope of quantification such as the $\forall$ quantification in $\forall x.\phi \Rightarrow \psi$. We can substantiate and substitute variables in the formulae by the above grammar ad infinitum, while keeping the underlying mechanism unchanged.

(1)	$\Leftrightarrow [x] \Rightarrow \phi \Rightarrow \psi$
(2)	$\Leftrightarrow [x] \wedge \phi \Rightarrow \psi$
(3)	$\Leftrightarrow [x]; \phi \Rightarrow \psi$
(4)	$\Leftrightarrow (\exists x.\phi) \Rightarrow \psi$

.

Table 7.6: Equivalences in the BHK interpretation.

The first equivalence lifts the declaration of the variable x under the quantification in a formula to its environment or context. Similarly, the second equivalence lifts the formula ϕ to the environment which now consists of x and ϕ, i.e. $[x] \wedge \phi$. The third equivalence shows how we operate on the environment. Thus, the environment is rewritten as the dynamic conjunction, i.e. $[x]; \phi$. Finally, the last equivalence means that we return the rewritten formula in the environment to the existing formulae ψ which now appears in the existence quantification, i.e. $(\exists x.\phi) \Rightarrow \psi$.

In the author's opinion, an example of the above process is in Unger's corresponding operators *write, write, swap, read* on references in the state

[13] A dependent version of this is called Henkin's quantifier and can be found in [86]

monad [15]. The resolution process, or variable scope binding, for example, can take place in the third equivalence, i.e. rewriting an environmental formula $[x] \wedge \phi$ to $[x];\phi$. We can swap variables around; another word for swap is *algorithm*[14], since the formula is rewritten as $\phi;[x]$. The resolution process can be concretised out as an anaphora resolution [298]. The third equivalence, of course, also provides a space for practical scope analysis.

By a similar mechanism, we can establish Egli's theorem, i.e. lift the formula $(\exists x.\phi) \rightarrow \psi$ to the state, becoming $[x] \wedge \phi \wedge \psi$. Hence, we rewrite it as $[x];(\phi \wedge \psi)$ and return it as $\exists x.(\phi \rightarrow \psi)$ [15]. A detailed analysis of Egli's theorem in [44] with a short introduction is in section 6.1.

Finally, in the author's opinion, our technique is similar to the normalization by evaluation (NBE) technique in type theories [262, 299]. NBE is also performed by normalizing a formula to its algebraic space, then reifying it back. The normalizing and reification steps are similar to the above steps (1) and (4). NBE is distinguished with call-by-value and call-by-name in [221, 6] since it doesn't specify the concrete syntactic strategy for evaluation.

[14] In general, a logic introduces a general guide line, whereas an algorithm (for any) concretises it out [252].

[15] It has a stronger version by replacing an existence $\exists$ operator to the universal operator $\forall$, i.e $\forall x.(\phi \rightarrow \psi)$. A further analysis of it is given in [43][p. 12]

7.4.2.2 Scope-taking as proof-search

The above mechanism doesn't show the complexity of the formalised formulae because their concrete syntax is not discussed. Linguistics formalisation is complex since it is pervasive from daily conversation through to scientific communication. A brief glance at its syntax, limited to the needs of this thesis, is reviewed for the readers' convenience below. For a contemporary general overview, see [159]; for a summary of the first-order interpretation of natural languages, as discussed in the philosophy of languages, see [106, 265]; for the typed theoretic interpretation, see [86, 118, 112]. An application or phenomena-oriented perspective can be found in [252] and [12].

One major insight in the jungle of formulae is the Curry–Howard correspondence, which states the equivalent between logics and types. Wadler [81] restates the analogies of the Curry–Howard isomorphism as

`propositions as types`

`proofs as programs`

In addition, he also postulates another analogy which concerns the practical derivation of the correspondence as

`normalisation of proofs as evaluation of programs`

Thus, our technique, which is proof-oriented and similar to normalization of proofs or to the normalization-by-evaluation technique, is analogous to the evaluation order (or evaluation of programs) in [6]. In addition, previous research in [6, 45] advocates the analogy between the evaluation of programs and the evaluation of scope taking. Furthermore, studying the presupposition phenomena also yields insight: [300, 82] show that the presupposition projection is a proof search, and [16] also shows that the projection is scope-taking. Thus, I propose a new analogy as follows

`scope-taking in linguistics is analogous to proof-search in logics.`

The analogy[16] is observed by following below research

$$\begin{tikzcd}
{[81]} \arrow[r, "\text{proofs search as evaluation of programs}"] & {[6, 45]} \arrow[d, "\text{Evaluation of programs as scope taking}"] \\
{[300, 82]} \arrow[u, "\text{presupposition as proofs search}"'] & {[16]} \arrow[l, "\text{presupposition as evaluation of scopes}"]
\end{tikzcd}$$

7.4.2.3 An analysis of the phenomenon

In comparison with Charlow's continuation monad [17][17], the author argues that the technique presented in this dissertation is more general than the continuation monad despite the fact that both techniques describe the interaction of a formula with its environment. Indeed, for example, [301]

[16] This analogy would act as this part's hypothesis.

[17] Barker [12] uses delimited continuation, which is similar to the continuation monad but not the same. See [9] for explanation

had shown that continuation monad is a symmetric case of the parametric continuation monad. This technique, however, is independent from theoretical implementation.

In the author's opinion, the proposed technique is similar to the ad-hoc solution of handlers in [10, 240], i.e. scoping the algebraic effects in the sense of [64]. However, we provide a clear mathematical reasoning behind the technique in our framework. It is, in essence, a free-scope analysis, instead of the syntactic analysis in Barker's continuation approach [12]. In the author's opinion, the choice between this proposed solution and the syntactic analysis may depend on the actual language being parsed. English, for example, has a strict left-to-right reading order, so the syntactic analysis is well-analysed, in contrast to how syntactic analysis plays out in languages with a looser reading order such as Russian or Japanese.

The ability to swap the variable scope in our framework is similar to the *internal dynamic* in dynamic semantics [27, p. 9]. A connective has an internal dynamic property if it can pass a variable from one formula to another. In our interpretation, it is the rewritten process in the environment. According to [27], the property of keeping a variable for the yet-to-come formula in a connective is called an *external dynamic*. In our interpretation, it is the first and last equivalence that lift a variable to the environment and transforms it back to the formula.

Traditionally, according to [42], the donkey phenomenon sentence

Every farmer who owns a donkey beats it.

has a truth condition or static interpretation in first-order logic

$\forall x.(\mathbf{farmer}(x).\exists y.(\mathbf{donkey}(y).\mathbf{own}(x,y)) \rightarrow \mathbf{beat}(x,y))$

However, we formalise the sentence in our framework as

$\forall x.(\mathbf{farmer}(x); \exists y.\mathbf{donkey}(y); \mathbf{own}(x,y) \Rightarrow \mathbf{beat}(x,y))$.

Hence, by swapping the quantifications and rewriting the $\exists$ quantification to $\forall$ as discussed in [43][p. 12], we have another equivalent dynamic interpretation that solves the problem of variable binding

$\forall x.(\forall y.(\mathbf{farmer}(x).(\mathbf{donkey}(y).\mathbf{own}(x,y) \Rightarrow \mathbf{beat}(x,y)))$

The first-order logic interpretation is problematic in that we do not know where the variable y is in the clause $\mathbf{beat}(x,y)$. This confusing arises because the logical implication $\rightarrow$ lacks the internal dynamic property. Hence, this formalisation demonstrates a mismatch between the logical interpretation and natural-language interpretation.

In our interpretation, we shift the first order logic implication, $\rightarrow$ to the dynamic implication $\Rightarrow$. Thus, instead of focussing on the truth condition, we are focussing on the changing of states during the interpretation. The verbs, or predicates, are a test on the changing of states rather than contributing to the truth meaning of a sentence. Clearly, the shifting solves the scope problem of the variable y. In other words, the dynamic implication $\Rightarrow$ has an *internal dynamic* property, while the first-order logic implication $\rightarrow$ contributes only to the sentence's truth condition.

From the above reasoning, the parsing process for the donkey sentence,
If Pedro owns a donkey, he beats it. (in short **C**)

is summarised as following, with an emphasis on the state-changing. The process can be seen as an instance of the specifications structure in [39], which is one of the category models of Hoare's logic. In addition, we view the interpretation of the second context in [149, 46, 164] as a special case of this framework. Hence, this interpretation provides an alternative framework to them. The operator of the discourse is highlighted as below.

Alternatively, the parsing process is detailed in states' changing in DPL in [42, p. 246–247]. Since we provide the interpretation of DPL in the framework above, Elbourne's analysis translates to our framework without any difficulty, and is hence omitted here. It is worth noting, however, that, where Elbourne used $F(R)$, we use $I(R)$.

Pedro owns a donkey $: \mathbb{M}\ \Gamma\ ([\Gamma; p : Ind; d : Donkey])\ \mathbf{S}$

Pedro owns a donkey $= \lambda\gamma.\exists p : Ind.\exists d : Donkey.own(p, d)$

he beats it : $\mathbb{M}\ \Delta\ \Delta\ \mathbf{S}$

he beats it = $\lambda\gamma.beat(read(he), read(it))$

$\mathbf{C} : \mathbb{M}\ \Gamma\ ([\Gamma; p : Ind; d : Donkey])\ \mathbf{S}\ \Rightarrow\ \mathbb{M}\ \Delta\ \Delta\ \mathbf{S}$

$\mathbf{C} = \lambda p.\lambda q.p \Rightarrow q$

Hence,

$\mathbf{C} : \mathbb{M}\ \Gamma\ [\Gamma; p : Ind; d : Donkey]\ \mathbf{S}\ \Rightarrow \mathbb{M}\ [\Gamma; p : Ind; d : Donkey]\ [\Gamma; p : ind; d : Donkey]\ \mathbf{S}$ (*)

$\mathbf{C} = \lambda\gamma.(\exists p : Ind.\exists d : Donkey.own(p, d)) \Rightarrow \lambda\delta.beat(read(he), read(it))$

$\mathbf{C} : \mathbb{M}\ \Gamma\ [\Gamma; p : Ind; d : Donkey]\ \mathbf{S}\ \Rightarrow\ \mathbb{M}\ [\Gamma; p : ind; d : Donkey]\ [\Gamma; p : ind; d : Donkey]\ \mathbf{S}$

$\mathbf{C} = \lambda\gamma.(\exists p : Ind.\exists d : Donkey.own(p, d)) \Rightarrow beat(p, d)$(anaphora resolution)

$\mathbf{C} : \mathbb{M}\ \Gamma\ [\Gamma; p : Ind; d : Donkey]\ \mathbf{S}$

$\mathbf{C} = \lambda\gamma.\forall p : Ind.\forall d : Donkey.(own(p, d) \Rightarrow beat(p, d))$ (swapping)

Table 7.7: Interpretation of the donkey sentence: *if Pedro owns a donkey, he beats it.*

We can think of this analysis as a dual for the progressive conjunction in type theoretical grammar by [86] where we analyse the process of contexts instead of analysing the formula. Furthermore, the idea of the secondary context also appeared in [302, 46]. Hence, roughly speaking, the last formula in the Table 7 can be read as a situation-semantics interpretation of the predicate logic in literature such as [42].

The first rule in the table (i.e the * rule) shows a transition from Δ to $[\Gamma; p : Ind; d : Donkey]$. It is called the context lifting in [141, 252, 86], or a specification monad which maps from the postcondition to the precondition in [240]. In general, if we want to transform a context Γ to a context Δ, we say—in a counter-intuitive manner—that Δ extends Γ, and write it formally as:

Definition A context $\Delta = [y_1 : B_1, y_2 : B_2(y_1), \cdots, y_m : B_m(y_1, y_2, \cdots, y_{m-1})]$ extends a context $\Gamma = [x_1 : A_1, x_2 : A_2(x_1), \cdots, x_m : A_m(x_1, x_2, \cdots, x_{n-1})]$ if we have a map $f = (f_1, f_2, \cdots, f_n)$from Δ to Γ such that

$$f_1(y_1, y_2, \cdots, y_m) : A_1$$

$$f_2(y_1, y_2, \cdots, y_m) : A_2(f_1(y_1, y_2, \cdots, y_m))$$

$$\cdots$$

$$\cdots$$

$$f_n(y_1, y_2, \cdots, y_m) : A_n(f_1(y_1, y_2, \cdots, y_m), \cdots, f_{n-1}(y_1, y_2, \cdots, y_m))$$

Related research is in [280] shows the parsing of a simple donkey sentence

If a man knocked, he left.

with the following sequence

$[\![\text{a man knocked}]\!] = [x|]; [|\mathbf{man}(x)]; [|\mathbf{knocked}(x)]$

$= [x|\mathbf{x.knocked}(x)]$

$[\![\text{he left}]\!] = [|\mathbf{left}(x)]$

$[\![\text{if a man knocked, he left}]\!] =$

$[|[x|\mathbf{man}(x).\mathbf{knocked}(x)] \Rightarrow [|\mathbf{left}(x)]]$

$= \lambda ij.i[]j \wedge \exists k.(j[x]k \wedge \mathbf{man}(x(k)) \wedge \mathbf{knocked}(x(k)) \rightarrow \exists l.k[]l \wedge \mathbf{left}(x(l)))$

However, the variable x in the above second clause, viz *he left*, has not been resolved in Charlow's interpretation. We can use the additional step of the swapping technique, or Egli's theorem as above, to rewrite his interpretation from

$[|[x|\mathbf{man}(x).\mathbf{knocked}(x)] \Rightarrow [|\mathbf{left}(x)]]$

to

$(\exists x.\mathbf{man}(x) \wedge \mathbf{knocked}(x)) \Rightarrow [|\mathbf{left}(x)]$

Hence, the swapping technique to translates it into

$(\exists x.(\mathbf{man}(x) \wedge \mathbf{knocked}(x) \Rightarrow [|\mathbf{left}(x)])$

and reduces further into[18]

$(\exists x.(\mathbf{man}(x) \wedge \mathbf{knocked}(x) \Rightarrow \mathbf{left}(x))$

In other words, we shift the variable scope from the first formula to both left and right formula in an implication sentence and the problem of the scope of the variable is resolute. For a further discussion of the scope-taking, see section 2.2.3 or [17, p. 113–116] where various frameworks are compared.

[18] We can employ the weakest-precondition calculus to do so, or the formula simply means adding a situation in situation semantics for the logical formula as discussed in [42]. Since, for example, that there is an equivalent of interpreting in states or in formulae as discussed in [181]

A detailed analysis of the phenomenon in dynamic predicate logic, and a comparison with a static approach, are given in [292]. A recent study of the phenomenon under continuation is pursued by [12, 46]. However, their approach yields a double-negation reading of the sentence as highlighted above. In contrast, this framework and the cDRT provide a direct reading.

7.5 Discussion

This chapter refines the cDRT in [24] by translating it into parameterized monads. It provides a simplifier model in category theory for the cDRT. The research in [24] is based on the λ-calculus and has recently been criticised by [69]. This new fine-grained model would eliminate the critics by them.

Adding the second state in the parameterized monad is crucial to capturing the pragmatics, or environment-related, issues which are essential for a natural-language understanding of the donkey phenomenon. This idea is discussed in [12, p. 200] who concluded that any frameworks that interpret the phenomenon require a delimited, composable continuation. However, the composable continuation has been interpreted in parameterized monads by [23]. In the above example, the phenomenon is only captured in parameterized monad framework by defining the dynamic implication $\Rightarrow$ through associating each evaluation state of the (compositional) evaluation process. The implication is used, for example, to represent the English condition *if*.

We cannot express the dynamic implication in general monads because the state k, which acts as the output of the first monad and input of the second monad, is required. In other words, we must have a declaration $\mathbb{M}\, g\, k \top$ and $\mathbb{M}\, k\, k' \top$ to connect between two monads.

Our idea of including two states in an expression also appeared in the Lamberk–Grishin calculus for the type-logical grammar, as discussed in [12, p. 194]. According to that discussion, an expression of type $(B \oslash C) \obslash A$ is interpreted as locally acting as A within a context of type B; hence it acts as a function transforming B into C.

The direct treatment of the quantifier *every* in the sentence

Every farmer who owns a donkey beats it.

is also studied by [63] with his variable-free dynamic semantics. Shan's analysis may lead to an unwanted reading that every farmer who owns a donkey beats each donkey that he owns. However, Shan, as well as [86], didn't analyse it further in their frameworks.

We are providing a category or type-theoretical interpretation of dynamic semantics in comparison with model theory interpretation in the DPL by [27] and its related systems. In our sense, the dynamic is interpreted as a transition between information states or contexts. The truth condition in type theory entails constructing an element of a given type. It can be studied further as a record type [259] or a database scheme [231, 232]. However, our innovation also involves the *swapping technique*, which shows how we operate the binding in a formula, and its underlying reasoning by the BHK interpretation. This technique is crucial in the general treatment of categories in linguistics since it allows a framework for reasoning about scope in compensation for the classical-logic interpretation of the implication (another version of Yoneda's lemma).

A deeper analysis of the donkey phenomenon in dynamic semantics is given in [43]. For example, according to [43], the dynamic approach to the donkey anaphora can extend to general cross-sentence anaphora. This observation is also a good research direction for examination in our framework.

Adding the described states in parameterized monads complicates the state monad in the conjunction and implication interpretations. There is a question of why do we need to do this? The donkey phenomenon above shows the empirical requirement of the correct sentential reading for the dynamic implication. It has not been captured by the state monads. In the author's view, the parameterized monads may have further applications in situations where the interpretation's correctness is paramount, such as in scientific or legal documents.

The donkey phenomenon is a subproblem of the linguistic context-dependent phenomena with associated linguistic structures such as scope-reading. Examples of those phenomena are the donkey anaphora or domain-restriction phenomena in [303]. In the author's opinion, parameterized monads do not solve the scope-reading problem since the author assumes that these problems are related to pragmatics. Instead, the parameterized monads provide a representation, categories and modularisation for a greater variety of linguistic phenomena. Thus, the monads preserve the linguistic structures of those phenomena more faithfully. In other words, parameterized monads provide a computational framework to capture those linguistic phenomena.

In other words, the parameterized monads provide a framework for a monad morphism between two-state monads and explicit splitting. We thus gain some theoretical advantages: for example, adding the second state (as per [101]) can capture the cataphora phenomenon.Moreover, it is possible to use records types [137] or collections [231] to model information states. In the author's opinion, the fields label or modularizing ability in collections is a proper candidate for modelling plurality in linguistics.

7.5.1 Related research

This chapter provides a dynamic semantics interpretation of the parameterized monads. A brief list of related research is given here. The most closest related research is [16] who uses graded monads from [304] to build a dynamic semantic framework based on [24], and interprets the presupposition phenomenon in linguistics. [53] shows that her theoretical framework, i.e. graded monads, and this framework are similar. Her central notions of dynamic implication $\rightrightarrows$ and monadic notion >>= are similar to our dynamic implication $\Rightarrow$ and the kept monadic notion. In addition, the presupposition projection is a strength of the contextual analysis in type theory in [141]. Her *reset* notion is similar to our *reset* notion defined in Chapter 5 or in [23]. However, it seems that the only distinction between her framework and ours is that the interpretation of Muskens' cDRT is quite natural to us while it is not natural in her framework. In addition, she did not study the donkey sentence in her framework, and she used possible-world semantics to interpret her information states rather than using context in type theory.

Another similar theoretical framework is also the dynamic monad by [280]. This framework and Charlow's have the same objective of interpreting dynamic semantics in monads. However, his framework is not thoroughly theoretically investigated, and is based on monad transformer by [202]. Besides, he tries to manage the information states in the focus presupposition binding rather than through the scope analysis of the pronoun. Thus, we can say that this version is a strong compositional version of Charlow's dynamic monads. We also declare the states in the types rather than in the denotation of linguistic expressions. Doing so is crucial since it makes the presentation clearer, and makes the typing declaration more meaningful.

[63] also provides the semantics for the donkey sentence for his variable-free dynamic semantics. In comparison with Shan, this research explains more detail on the state-changing during the sentential analysis of the phenomenon. Chapter 4 by [6] also modelled the context as a graph. Hence, he lifts the scope of the sentence's quantifiers into the contexts, and performs swapping operators on the context. This idea is similar to our *swapping technique*. More clearly, it is the concretisation of this technique in the categorial grammar. He also used one context to model the change in the contexts, and splits the context into two by the ⊚ operator, with the idea of left and right concatenation from categorial grammar to describe how the contexts interact. Let us consider the context in the derivation in an example by [6, p. 101]

$$\frac{[\Gamma = (x : NP, f : saw, y : NP) \circledcirc 1] \vdash fyx : s}{[\Gamma' = (f : saw, y : NP) \circledcirc (1, x : np)] \vdash fyx : s}$$

In the author's opinion, his idea and ours are similar in the sense that both are describing the change in the context. Thus, the change from pre- to post-state is expressed as the contexts in the premise, and as the conclusion in his derivation. Thus, if he wrote

$$\frac{\Gamma \vdash fyx : s}{\Gamma' \vdash fyx : s}$$

it means

$\mathbb{M}\,\Gamma\,\Gamma'\,fyx : s$

in this framework. In general, this is a special case of the Curry–Howard correspondence between the proof and programming.

Some further related research is by [32] who uses category theory with distributional semantics rather than dynamic semantics to study natural language. This technique is called the Frobenius algebra. According to [157], this algebra has a particular existential quantifier property that $\delta \wedge (\exists x.\phi) \Leftrightarrow \exists x.(\delta \wedge \phi)$. According to [305], it is because the connectives $(\Rightarrow, \wedge, \rightarrow, \times)$ and quantifiers $(\forall, \vee, \exists, +)$ are a duality of the right and left adjoints of a simpler category. In the author's opinion, we can think of Taylor's connectives as being similar to the generalized quantifiers in [108], or to continuation in [12].

The continuation interpretation of linguistics with the donkey sentence analysis is represented in [12, 46, 149, 10]. These investigations are based on continuation, and they delimit control with classical logic to overcome the scope problem of linguistic phenomena. We enrich their study and applications of category theory in linguistics by providing an alternative foundation. Hence, we can interpret the donkey phenomenon and the implication directly rather than having to use the double-negation rule from classical logic.

In comparison with the towering notion in [12], the syntax is replaced by the pre- and post- information states, and we replace the syntax calculus as the discourse structure in our framework. These replacements preserve the continuation idea of keeping the interaction between an expression and its environment, or context, without the pros and cons of syntax. However, we still keep the semantic by Montague's grammar, which is the same as in [12]. Their contextual illustration, and hence continuation, is represented by the introduction of the hole [] which has the λ-calculus interpretation, $\lambda k.k$ is merged to be the pre-state in our framework, i.e. λs_1. Hence, the continuised denotation is the second state s_2.

Readers may wonder, where is the syntax analysis? I would say that it is to be found in the typing declaration. [114] shows that we can interpret syntactic calculus in type theory. In order to encode the swapping technique[19] in type theory. We need nominal type theory from [306, 307] or linear logic in type theory from [30]. The full development, however, is beyond this dissertation's intended scope.

[19] also had an idea similar to ours. The transition technique is also named

[19] In a manner similar to parallel computing.

as hypothetical proof in [19]. Formally, the hypothetical proof rule is

$$\frac{\Gamma, p : P; A, z : B \vdash a : C; D; b : E}{\Gamma \vdash \lambda_p.a : P \to C; A \multimap D; \lambda_z.b : B \to E}$$

[19] have a rich linguistic structure. However, we base our proposal on type theories and logical interpretation.

Finally, [291] develops the analysis of the donkey phenomenon further in the inquisitive semantic framework from [308]. In contrast with that approach, our approach has a strong mathematical background by the research of [35].

7.5.2 The continuation monad

The application of continuation in linguistics is studied in [149, 166, 12]. The lifting operator for continuation monads is represented in [9] as $\eta(o) = (e \to \omega) \to \omega : \mathbb{M}\,\omega\,o$, and it has been studied in linguistics in [17]. In parameterized monads, I propose a new representation for composable continuation as

$$\eta(o) = (o \to \Delta) \to \Gamma : \mathbb{M}\,\Gamma\,\Delta\,o.$$

This means that the states are associating with the return values. A comparison of extensions to the continuation monad is [301]. Both frameworks are based on category theory. However, I based this research on parameterized monads, while Melliès defines his own frameworks.

In a concrete example, [302] used $\Delta = \Gamma \to o$ for the second context. Hence, his typed interpretation of a formula with a type o is

$$\Gamma \to (\Gamma \to o) \to o.$$

On the other hand, if we still keep $\Delta = \Gamma \to o$, then our parameterized monad interpretation is

$$o \to (\Gamma \to o) \to \Gamma.$$

We see that the above two interpretations are identical up to swapping the first and third parameters o and Γ.

7.5.3 The state monad

The parameterised state-manipulation has been extensively discussed as Monoidal Typed Command Calculus in [23, p. 32]. However, we still can extend this calculus by adding further rules. For example, we can govern the changing of the state from s_1 to s_2 by adding additional $C - Conseq$ rules as implicit state-manipulation as in [23, p. 34]:

$$\frac{s_1 \Rightarrow s_1^{'} \quad \Gamma; s_1^{'} \vdash^c c : A; s_2^{'} \quad s_2^{'} \Rightarrow s_2}{\Gamma; s_1 \vdash^c c : A; s_2}$$

For example, the state-changing rule $\hat{}$ in [15] can be interpreted as the $C - Conseq$ rule for the implicit addition of a new entity in the state with

$s_1^{'} = s_1, s_2^{'} = s_2, s_2 = \hat{}(c, s_1)$

Finally, in the author's opinion, the discourse representation $\oplus$ in [15] can be represented as the merging relation $\bowtie$ in [23, p. 31–32]. The stack manipulations *push, top, add* are represented as the *StkPrg* program for a simple stack machine program with defining rules in [23, p. 7–8, 28].

Chapter 8

Imperatives phenomenon in parameterized state monads

This chapter shows another application of parameterized monads to interpret the imperative phenomenon. This interpretation is characterized as a computational approach to the phenomenon. Monads, in our sense, are the computational types by [8], and types and logics are equivalent by the Curry–Howard correspondence as discussed in section 2.1.2.5. This research is based on the previous one by [49] using Hoare's logic [51]. The precondition and postcondition in Hoare's logic are captured by the first and second states in parameterized monads, respectively.

The basic literature review of this phenomenon is given as follows. The philosophical approach to the phenomenon can be found in [309, 310, 311, 312, 313] with recent developments in [314, 315, 316, 317, 318, 319]. Linguistic study of the phenomenon includes [145, 320, 321, 77, 322, 323, 76]. However, we are approaching the phenomenon in the logical aspect, so the related literature is the logical and computational approach to the phenomenon as studied by [311, 313, 324, 49, 139, 55].

According to [49, p. 1], the definition of imperatives is

> An imperative is a tenseless and subjectless sentence typically used to ask someone to do something or not to do something, and which does not denote a truth-value.

According to [325, 115], despite its formation as an incomplete sentence as indicative one, imperative can be an object of scientific reasoning. The basic idea of the reasoning is illustrated as: do this, hence if we do not do that, we cannot do this. For example, let us illustrate this reasoning point by an example in [324] as below

Open the door!

However, the door cannot be opened unless it is first unlocked. Thus, the imperative also means
Unlock the door!

Hence, we can reason about an imperative by its inferences. In the above example, a request to open the door has an implicit meaning associated with unlocking the door. The reasoning of the imperative is more essential, for example, in the case when we talk to a robot and say

Clean the house!

Then the robot should know what to do, such as taking the vacuum, plugging it into the power, starting the work, finishing the work, etc. Practically, according to [55], any dialog theories have to interpret imperatives.

However, there is a challenging question: how to build an inference system for imperatives? This problem is challenging since we are reasoning with verbs, or with predicates rather than nouns. Besides, the interpretation of an imperatives is heavily influenced by the environment in which it is uttered. According to [324, 309], the meaning of imperatives can be interpreted by Dubislav's analogy. The analogy states that there is an equivalence between imperatives and declared sentences. This analogy was discussed originally in [309] and further developed in [310, 311, 324].

Basically, to interpret an imperative sentence, we translate it to the equivalent indicative sentence, then we use inference rules on the indicative sentence before transferring it back to the imperative form. This process is illustrated by the following diagram, where I_1 and I_2 are imperative sentences, S_1 and S_2 are interpreted indicative sentences. The horizontal arrows show the translation, and the vertical arrow shows the inference rules.

$$\begin{tikzcd}
I_1 \arrow[r] & S_1 \arrow[d] \\
I_2 & S_2 \arrow[l]
\end{tikzcd}$$

Table 8.1: Dubislav's analogy.

However, this approach poses a problem called Ross' paradox, which is raised by the interpretation of the disjunction to the phenomenon.

8.1 The Ross' paradox

[311] follows [310] to provide an answer the question:

Can imperative be a part of logical inference?

Then he questions the validity of [309] and [310] in the case of disjunction. Validity, in Ross' sense, means that, if all the premises are correct, then the conclusion is correct.

$$\begin{tikzcd}
I(a) \arrow[r] & S(a) \arrow[d] \\
I(a \vee b) & S(a \vee b) \arrow[l]
\end{tikzcd}$$

A natural-language example is

Slip the letter into the letter-box!

This sentence could be inferred as

Slip the letter into the letter-box! Or burn it!

That is an undesirable implication. Thus, [311] argues that, in order to avoid the paradox, the context or *holding good* of the imperative should be presumed. It is similar to the idea in [309] that the inference is valid if the subject is presupposed.

8.1.1 Logic in imperatives

According to [49, 313, 139], there is a logic for the imperative despite its incomplete formation in comparison with an indicative sentence. In their observation, the basic logical operators in the phenomenon are

Direct imperative: $\rho =$ Come here!

Negative imperative: $\rho_1 =$ Don't do that!

Conjunction: $(\rho_1; \rho_2) =$ Sit down and listen carefully!

Disjunction: $(\rho_1 + \rho_2) =$ Shut up or get out of here!

Conditional imperative: $(\mathrm{C}\ )\ \rho) =$ If it is raining, close the window!

8.1.2 Properties

According to [49, p. 4], interpreting an imperative requires a state of affairs or the speaker's context. Understanding the context allows us to avoid inappropriate speaker expectations, thus avoiding inappropriate inference such as Ross' paradox. The context is called a situation in [326]. In addition, using illocutionary forces allows us to deduce whether an imperative is satisfiable or not [326].

Thus, an inappropriate order such as *Have three arms!* is identified and rejected. To explain it, we may say that there is a precondition of pragmatics for each imperative. To make sense of an imperative, the precondition must satisfied. An imperative that overcomes the conditions is called *satisfiable* by Fox [49].

We also have a post-condition for verifying the actions of the addressee, to update the state of affairs, or to record the pragmatic issues which relate to speaker and hearer performance. For example, if the speaker utters the order

Close the door!

In order to satisfy the above order, we verify that the door is closed after the action of the hearer. If the door is closed, an imperative has a post condition that can be used for further reasoning. If the door is not closed, the imperative is not satisfied, which Vranas calls *violated* [316]. Thus, the post-condition can be extended to include complex reasoning on common grounds and pragmatic issues.

Besides contextual dependency, imperatives have both subjective and objective properties. Imperatives have the objective property because they explain an imperative. This is the demand from a speaker to a hearer. The demand must be understood and interpretable by both parties. Therefore, it must be objective in order to be analysed. Imperatives are subjective because they depends on the situation such as whether a speaker has authority over a hearer.

According to [311], an imperative cannot provide a truth value of true or false in the sense used in propositional logic. In place of true and false, he proposed the terms, validity and invalidity. The corresponding term *correctness* is used in [49]. If an imperative satisfies its pre-condition, then it is called validity.

In the author's opinion, Ross has classical or propositional logic in mind when he discusses the concept of an imperative's validity. That means a proposition must be either true or false. Thus, he cannot categorise the imperative to that frame. If we view modern logics in which a proposition is not just true or false (e.g. in [93, 131]), the author supposes that we can provide a logic for imperatives in Ross' sense. Then, validity would mean a true proposition, while an imperative without classification as validity or invalidity would mean a hypothetic judgement in the sense of [123].

Thus, Nanevski and Pfenning and Pientka leads to an explanation of non-applicable of classical logic in imperative as discussed in [311, 49]. It illustrated by an example in [49]: if an employee utters, *Do your work!* to the boss, then it is invalid. It is not implied that its negation, *Don't do your work!* is a valid imperative either.

Practically, according to [311, p. 36], defining the validity of an imperative is impossible, for example in association with feelings or morality [325], in comparison with defining validity for an indicative sentence. To arrive at a practical logical interpretation, Ross also proposed using Dubislav's analogy and defining the term, *satisfaction*. If an imperative **I** to be *satisfaction*, then its corresponding indicative sentence **S** has a *truth value* in classical logic. Consequently, we can build a logical interpretation of the phenomenon.

In addition, according to [49], interpreting the satisfaction of an imperative also provides a means to verify it. For example, an imperative *Close the door!* is satisfied if we can check that the post-condition has a property that the door is closed. If the door is not closed, then the imperative is not satisfied.

8.1.3 Specifications

There are various definitions for the objective meaning of an imperative. For example, it is called a *requirement* [49] and *prescriptions* [316]. Ross [311][p. 33] calls it *theme of demand*:

> An imperative expresses a demand for action and must therefore of

> necessity contain a statement of the nature of the thing demanded. It is impossible to demand without demanding something. This "something" I propose to call the "theme of demand". The theme of demand consists of a certain fact, or a state, or an activity, which is assumed not to exist at the moment of the demand; but the realisation of which is requested by the demand through the action of the one to whom the demand is directed. Every imperative may therefore be conceived to be resolved into two factors, the properly imperative factor, expressing that something is demanded, and the indicative factor, describing the theme of demand. It is now possible to segregate the indicative factor and give it an independent formulation in a sentence which describes the theme of demand, and which will therefore be true in case the demand is complied with. For example, in the imperative, “Peter, close the door” the theme of demand is described in the sentence “Peter closes the door”. It may then be laid down that to an imperative

In this dissertation, I adopt the terminology from Hoare’s logic and call it a *specification.* A specification is a list of indicative sentences which express the meaning of an imperative. According to [49, p. 13], we can verify a specification and avoid the Ross paradox. This is because the specification does not infer any new utterance. If we describe the specification more clearly, such as the to-do list by [145], the verification task is made easier. An imperative is always verifiable as accomplished, or not accomplished, by pragmatic factors, even if the demanded task is in a specific time in the future, such as

Run tomorrow!

Or even just in a vague near future:

Get well soon!

I call the translation process from an imperative to a descriptive specification, a specification process, recalling Dubislav's analogy between imperative and indicative sentences. The process is discussed in [310] as follows:

> According to this method the command "Shut the door" corresponds in a certain sense to the indicative sentence "The door is to be closed", or more explicitly, "The action of closing the door is belonging to the class of actions which are to be performed". Or generally, there is a syntactic rule according to which an imperative sentence of the form "Do so and so" may be transformed into an indicative sentence of the form "Such and such action is to be performed, resp. such and such state of affairs is to be produced".

Hence, by adding the context to an imperative expression, I propose modifying the Dubislay analogy to associate with an evaluation context as

$$\begin{tikzcd}
\Gamma \vdash I_1 \arrow[r] & \Delta \vdash S_1 \arrow[d] \\
\Gamma \vdash I_2 & \Delta \vdash S_2 \arrow[l]
\end{tikzcd}$$

Table 8.2: Revised Dubislav analogy.

In our parameterized monad approach, Γ or Δ is represented by a pair of the pre- and post-states $\langle G, H \rangle$. The analysis of context in imperatives, in the author's opinion, is essential for the interpretation of the phenomenon as discussed in the literature.

8.1.4 Monadic approaches to the semantics of imperatives

Our monadic approach to imperatives is a dynamic semantic one. The related research are [320, 139, 326, 55, 49, 313] in parallel with the modality approach by [322, 77].

Furthermore, I propose that, using the evaluation order in monads by [50, 189], we can tackle the order-dependence of the conjunction problem in [49, p. 6]. It has a distinguished theoretical foundation with the continuation evaluation order in [6], and a similar, but not identical, concept is also discussed in Chapter 11 of [12]. The order-dependence of conjunctions means that a later imperative in a conjunction can be satisfied only after the previous imperative has been satisfied. Thus, for example *Open the envelope! And read the letter!* differs from *Read the letter! And open the envelope!*. That difference demonstrates that the conjunction in an imperative is not commutative as it is in classical logics.

Basically, evaluation order or scope-taking in section 2.2.3, with detail in

[12, 75], study and assert the practical issue that the order of terms in a formula contributes to the meaning of that formula. Thus, this technique is better at analysing quantifiers, than is the edge treatment of quantifiers in typed logic grammar in [18]. We can employ this technique to interpret the order-dependent conjunction in [49]. However, we should keep in mind that monads, instead of continuations, provide the theoretical framework. Either of the two techniques would be adequate in this dissertation's research scope. It is not, however, achievable in classical logics, but it is quite straightforward in monads if we interpret the conjunction as a binding operator as below.

8.2 The interpretation in monads

We base our interpretation of Hoare logic in monads via [52] with a similar study of Hoare's type theory by [237]. The differences between Hoare logic and monads originate in the principles used to express them. Hoare logic is expressed in imperative programming languages, while monads are expressed in functional or compositional principles.

8.2.1 Hoare state monads

Let us assume that the pre- and post-conditions are collections of propositions $\mathbb{P}$ as defined in [49]. So, the precondition P is a subset of $\mathbb{P}$, i.e. $P \subseteq \mathbb{P}$. In order to define Hoare state monads, we need to define the unit and binding operators for general monads as per [8]. Let a be a type, then let us follow [52] to define the return unit as

$\eta(a) = \mathbb{M}P\ P\ a$,

now, let us investigate the binding operator which is, by default,

$\mathbb{M}P_1\ a\ Q_1 \to (a \to \mathbb{M}P_2\ b\ Q_2) \to \mathbb{M}\cdots b\cdots$.

Thus, the question is: how we define the final monad in the binding operator? I assume that the imperatives are normal and reasonable; we need not be concerned with absurdities here.

My proposal is that, in order to make sense of the binding, the second precondition, P_2, should be stronger than P_1. Thus, post-condition Q_2 is weaker than post-condition Q_1, since the first imperative is added by making a second order. Thus, the pre-condition of the second order should be stronger than the pre-condition of the first because it is to happen only after the first order has been accomplished. Thus, the post-condition, or the final goal of the second order, should be weaker than the first one since the goal is more clarified by the action in the first-order imperative. Formally, we define that strength-ordering as

$P_1 \sqsubseteq P_2$, and $Q_2 \sqsubseteq Q_1$. The $\sqsupseteq$ is the subsumption notion, i.e. if we write $P \sqsubseteq Q$, it means that there is a function from P to Q.

And so, the binding operator returns

$\mathbb{M}P_1\ a\ Q_1 \to (a \to \mathbb{M}P_2\ b\ Q_2) \to \mathbb{M}\{P_1 \wedge (P_1 \sqsubseteq P_2)\}b\{Q_2 \wedge (Q_2 \sqsubseteq Q_1)\}$.

Were we to argue further, we should let the conditions: $(P_1 \sqsubseteq P_2), (Q_2 \sqsubseteq Q_1)$ manifest on another dimension since we just want the result to have a representation as: $\mathbb{M}P_1\ b\ Q_2$. For further discussion, the author refers to the topic of adaptation in Hoare logic as discussed in [327], and to separation logic in

[244].

8.2.2 An interpretation

This monadic interpretation captures the evaluation order by the binding order, which is problematic for conjunctions in [49]. The detailed interpretation is illustrated below.

1) A single imperative such as Portner's imperative

Feed the bird!

has a direct representation as

P $\{\rho\}$ Q $= \mathbb{M}P\ Q\ \rho$

2) The negation imperative, such as the imperative in [49]

Don't close the window!

returns the η operator of absurdity,

i.e. $\neg\phi = \eta(\phi) = \mathbb{M}P\ (\phi \rightarrow \bot)\ P$

3) The conjunction imperative, such as

Find the key under the carpet! And open the door!

is the above binding operator

$$\rho_1 \wedge \rho_2 = \mathbb{M}\ P_1\ \rho_1\ Q_1 \gg \mathbb{M}P_2\ \rho_2\ Q_2.$$

4) The condition imperative is interpreted as a dynamic implication and explained in the donkey anaphora section.

The conditional imperative is one case of hypothetical reasoning. It is represented as a condition in an imperative programming language by [49]. Formally, it is written as

$\{P\}\phi \Rightarrow \rho\{Q\}$

For example, ϕ is *The cat, Felix, is hungry*, and ρ is *Feed it!*.

This relationship means that the conclusion imperative ρ depends on the conditional imperative ϕ. I use the same method as the dynamic implication for the donkey phenomenon in the Chapter 7 in the general treatment of conditional imperatives. Thus, we are checking the pre- and post-conditions P and Q to verify the correctness or satisfaction of the conditional imperatives. This approach is dynamic one, and continues the research in [49] by incorporating with dynamic predicate logic from [27].

This approach has an advantage that we do not need an excluded-middle law to interpret the imperative. For example, in [49], the hearer is required either to take an action if the condition is met, or not to take that action. Their example is

ρ = *Close the door!*

ϕ = *It is raining.*

P = *The door is open.*

Q = *The door is closed.*

Thus, their interpretation means either: if the condition is met, then take the action as

$(P \wedge \phi)\{\rho\}(Q)$

or, if the condition is not met, then do not take the action as

$(P \wedge \neg\phi)\_(\neg Q)$

That interpretation is not desirable because we do not know what is meant by "do not take an action ρ", and also because ρ can be performed without first meeting the condition ϕ. On the other hand, the dynamic approach means that a listener takes the action only if the condition is met. Formally, it is written as

$(P)\phi \Rightarrow \rho(Q)$

Hence, we lift the condition ϕ to the environment via the BHK interpretation, and it has the following equivalent interpretation:

$(P \wedge \phi)\rho(Q)$

5) I provide the continuation and nondeterministic approaches to the interpretation of imperatives with disjunction as below. An example of a disjunction imperative is

Stand here! Or don't eat the cake!

The continuation approach

In general, the disjunction operator differs from the conjunction operator in that its result is nondeterministic. In the case of a conjunction and a condition, the result of an action is determined by a commander, whereas in the disjunction case, the final result provides the listener a choice of action. in the case of *Close the door or close the window*, a listener can choose to close the door, or close the window, or close both.

The nondeterministic problem is vastly studied in computer science field. To solve the problem, I suggest using continuation as a technique to interpret the nondeterministic problem. Continuation lifts a function so that its evaluation depends on its environment. The continuation is formulated as: given a function f, its continuation is $\lambda c.f$, where λ is an arbitrary function. In other words, there is an arbitrary function λ from an environment c to interpret f. Thus, we derive the rule for disjunction as follows.

given : P = ¬(the door is closed ∧ the window is closed),

Q_1 = the door is closed.

Q_2 = the window is closed.

$$\frac{P\ \{\rho_1\}\ Q_1 \quad P\ \{\rho_2\} Q_2}{P\ \{\rho_1 \vee \rho_2\}\ (\lambda c.if c = \rho_1\ \ Q_1\ else\ \ Q_2)}$$

If we took an alternative semantics, the post-condition is a subset of the powerset of {door closed, window closed} (i.e. {door closed}, {window closed}, {door closed, window closed}). This post-condition can be written more intuitively as

$\lambda c.c \subseteq \mathcal{P}(\{doorclosed, windowclosed\})$

where $\mathcal{P}$ stands for the powerset notion.

Basically, the continuation approach requires a further, and external, clarification. In general, disjunction is a nondeterministic phenomenon. The post-condition in the above interpretation is the weakest post-condition in the sense that it returns either Q_1 or Q_2. A listener may perform both actions, leading to the actual post-condition being that both the door and the window are closed. There are several uses of continuation in linguistics as discussed in section 4.3.

In the field of program verification, continuation is also used as a lifting function to combined two Hoare statements [328]. There is also the possibility [329] of taking post-conditions as primitives rather than as pre-conditions in general Hoare logic. Doing so enables us to trace back the pre-conditions from the post-condition and the program's return value. While it would be worth seeing how this approach would enhance the research in [139], we should bear in mind that human imperatives are more intuitive, direct, ambiguous than programming languages.

The guarded nondeterministic approach

This approach is similar to the above continuation approach. However, it provides explicitly where the further clarification is. It is restricted by the LEFT and RIGHT rules as described below. These rules shift the choice operator from the post-condition in a continuation approach to the formula's formation. Hence, we can control, or guard, the choice more properly. It is a simplified version of the delay-guarantee method for parallel programming by [330, 329].

8.3 The imperative logic

I present here an imperative logic which is an improved version of the logic in [49], with handling rules for disjunctions. For an imperative ρ, we say $\rho : A$, for a clause type A as discussed in [145]. The clause types can be further characterised to forces such as ORDER, COMMAND, REQUEST, ADVICE, WARNING, INSTRUCTION, THREAT, DARE, WISH, PERMISSION, etc [207, 321]. The clause types differ from basic types in formal semantics in the sense that the latter's types, constructed in [4], have only basic categories $e, e \to t, e \to t \to t, \cdots$ (where e stands for entities, or individuals, t for truth, or proposition).

In the author's opinion, Portner's types are types in the sense of the typed theoretical semantics and grammar in [26, 112]. More particularly, they are the hypothetic judgement in the sense of [123]. Providing an imperative with a type in Portner's sense means setting a type for a hypothetic judgement. Indeed, it is correct since [145] used modality and clause type as his foundational background to study the phenomenon. Thus, we adopt type-theoretic interpretation of natural language as a general syntax and semantics for imperatives, instead of defining it anew as was done in [49], following [312]. Thus, the foundation for the overall framework can be found in the previous chapter.

Furthermore, we can specify, or encapsulate in the definition in [49], the nondeterminism of a disjunction by using reverse Hoare logic from [329]. I am going to represent the logic from [49, 313] in a type-theoretical formulation. This framework is similar to the judgement formulation of the phenomenon in [139]. A related research is the interpretation of dynamic logic by [331], which [49] is based on, in the type theory or natural deduction style of [332].

Suppose that A, B, etc stand for clause types, P, Q, etc stand for condition, and ϕ, ρ stand for a proposition and an imperative respectively. Γ is a context, x is a variable, and $\mathbb{P}$ is the set of propositions. The axioms and inference rules for the logic are described below.

8.3.1 Axioms

When pre- or post-conditions are not mentioned below, it is assumed that they are applied globally. In addition, the judgement

$\Gamma; P \vdash \{\rho\}Q$

is a shorthand notation for

$\Gamma \vdash \mathbb{M}\; P\; Q\; \rho$.

A0. All tautologies in predicate logic are axioms. By the propositions-as-types principle, tautologies can also be expressed by typing rules in [99].

A1

$$\frac{\Gamma; P \vdash \{\phi \Rightarrow \rho\}Q \;\; \phi : \mathbb{P} \;\; \rho : B}{\Gamma; \{P \wedge \phi\} \vdash \{\rho : \Pi(x : \mathbb{P})B(x)\}Q} \Rightarrow elim \frac{\Gamma; P \wedge \phi \vdash \{\rho\}Q \;\; \phi : \mathbb{P},\;\, \rho : B}{\Gamma; P \vdash \{\phi \Rightarrow \rho : \Pi(x : \mathbb{P})B(x)\}Q} \Rightarrow$$

intro

A2

$$\frac{\Gamma; P \vdash \{(\rho_1;\rho_2) : \Sigma(x:A)B(x)\}Q}{\Gamma; P \vdash \{\rho_1 : A\}((\rho_2 : B)Q)};\mathit{elim}\frac{\Gamma; P \vdash \{\rho_1 : A\}((\rho_2 : B)Q)}{\Gamma; P \vdash \{(\rho_1,\rho_2) : \Sigma(x:A)B(x)\}Q};\mathit{intro}$$

A3

$$\frac{\Gamma; P \vdash \{(\rho_1 + \rho_2) : A + B\}Q}{\Gamma; P \vdash \{\rho_1 : A\}Q \vee \{\rho_2 : B\}Q} + \mathit{elim}\frac{\Gamma; P \vdash \{\rho_1 : A\}Q \vee \{\rho_2 : B\}Q}{\Gamma; P \vdash \{(\rho_1 + \rho_2) : A + B\}Q} + \mathit{intro}$$

A4

$\dfrac{\Gamma; P \vdash \forall x(\phi \to \psi) : \Pi(x:\mathbb{P}).\Pi(y:\mathbb{P}).C(x,y)Q \; \phi : \mathbb{P}, \psi : C}{\Gamma; P \vdash \phi \to (\forall x.\psi_x) : \Pi(y:\mathbb{P}).\Pi(x:\mathbb{P}).C(x,y)Q}\mathit{swap}$, x is not free in ϕ

A5

$\dfrac{\Gamma; P \vdash \forall x.\phi(x) : \Pi(x:A).B(x)Q}{\Gamma; P \vdash \phi(t) : \mathbb{P}Q}$, t is free in ϕ,

8.3.2 Rules

We add LEFT and RIGHT rules, which are a simplified version of the top-down rule for a reliable guarantee of specification for the disjunction. Details of the analysis can be found in [330]

Modus Ponens:

$$\frac{\Gamma \vdash \phi : \mathbb{P} \qquad \Gamma \vdash \phi \to \psi : \Pi(x:\mathbb{P})\mathbb{P}}{\Gamma \vdash \psi : \mathbb{P}}$$

Necessitation: I use the contextual modal type theory by [123] to represent the modality, ‘necessary’

$$\frac{\Gamma; \vdash \phi : \mathbb{P}}{\Gamma \wedge (\rho : A) \vdash \phi : \mathbb{P}}$$

where $\rho : A$ means a hypothetic judgement as per [123]. A necessitation introduction and elimination rules are described in [123, p. 7].

Universal Generalisation:

$$\frac{\Gamma \vdash \phi : \mathbb{P}}{\Gamma \vdash \forall x.\phi : \Pi(x : \mathbb{P})\mathbb{P}}, \ x \text{ is not free in } \phi$$

LEFT rule:

$$\frac{\Gamma; P \vdash c_0 : A; Q}{\Gamma; P \vdash c_0 \sqcup c_1 : A + B; Q}$$

RIGHT rule:

$$\frac{\Gamma; P \vdash c_1 : B; Q}{\Gamma; P \vdash c_0 \sqcup c_1 : A + B; Q}$$

SPLIT rule:

$$\frac{\forall l \in L.\Gamma; P \vdash c : A; Q_l}{\Gamma; P \vdash c : A; (\bigsqcup_{l \in L} Q_l)}$$

(AND)

$$\frac{\Gamma \vdash P\{\rho_1 : A\}Q \qquad \Gamma \vdash Q\{\rho_2 : B\}R}{\Gamma \vdash P\{(\rho_1; \rho_2) : \Sigma(A, B)\}R}$$

OR is replaced by LEFT and RIGHT rules.

CONDITIONS

$$\frac{\Gamma \vdash (P \wedge \phi : \mathbb{P})\{\psi : B\}Q \qquad \Gamma \vdash (P \wedge \neg\phi) \rightarrow Q}{\Gamma \vdash P\{\phi \Rightarrow \psi : \Pi(x : \mathbb{P}).B(x)\}Q}$$

Alternatively, we can use dynamic implication for the conditions rule. It is formally written as

$$\frac{P\{\phi \Rightarrow \rho : \Pi(x : \mathbb{P}.B(x)\}Q \ \phi : \mathbb{P}}{\Gamma \vdash \{P \wedge \phi\}\rho : B\{Q\}}$$

WP (weakening post condition)

$$\frac{\Gamma \vdash P\{\rho : A\}Q \qquad Q \rightarrow R}{\Gamma \vdash P\{\rho : A\}R}$$

SP (strengthening precondition)

$$\frac{O \rightarrow P \qquad \Gamma \vdash P\{\rho : A\}Q}{\Gamma \vdash O\{\rho : A\}Q}$$

CR(lifting)

$$\frac{P' \rightarrow P \qquad Q \rightarrow Q' \qquad \Gamma \vdash P\{\rho : A\}Q}{\Gamma \vdash P'\{\rho : A\}Q'}$$

(CDR1)

$$\frac{\Gamma \vdash P\{\rho : A\}Q \qquad \Gamma \vdash P'\{\rho : A\}Q'}{\Gamma \vdash (P \wedge P')\{\rho : A\}(Q \wedge Q')}$$

CDR2 is replaced by LEFT and RIGHT rules. Furthermore, in the author's opinion, The + type, which represents the disjunction type, can be viewed as the dot type • [170].

8.4 Discussion

The relation between the imperative logic and Hoare's state monad is that we are using the Hoare state (or parameterized) monad as a general framework and the logic substance the application of the framework. It is similar to the definition of typed command calculus inside parameterized monads as discussed in section 5.5.

This research considers category as a dynamic approach to the phenomena examined in the sections above. Hence, interpreting [49] in parameterized monads means that we solve their open problem of combining their axiomatized system of imperative with dynamic predicate logics in Sections 2 and 3. Thus, this research, in a manner similar to the dynamic approach in [55], has a potential practical implication to NLP and other fields. Indeed, for example, according to [76], the two main approaches to the phenomenon in linguistics are the dynamic approach [55] and the modality approach [77]. However, the distinction between our research and [55] is that the latter is based on DRT. On the other hand, we can interpret cDRT in our system. Hence, we can provide a compositional approach to their system.

The results thus far open several future research prospects:

- It seems quite straightforward to formalize to-do lists in [145] as the writer monad. Furthermore, it is also a prominent future research to see how update monad by [229] can be used to model the interaction between common grounds, the to-do list in Portner's sense, in this framework.
- Subjects of imperatives are not mentioned in [49], while they are a third solution to interpret an imperative in [311] besides the satisfaction approach. Taking a type-theoretic approach, the author suggests representing subjects as metavariables in type theories as in [121, 138]. Doing so may result in a difference from the imperatives and promises as discussed in [145]. Hence, we would be able to use dependent types or subtypes to formalize an inference process for the phenomenon.

Chapter 9

Additional linguistic phenomena in parameterized monads

This chapter studies the formalization of additional linguistic phenomena: definite descriptions, demonstrative, and conventional implicature phenomena in parameterized monads.

9.1 Definite descriptions in $\mathbb{IO}$ monads

There is a vast and traditional literature on the topic of definite descriptions in linguistics and logics. This section is selective, for a brief introduction, see [333, 71]. Instead, a formalization of definite descriptions as $\mathbb{IO}$ monads, in section 3.6 in the sense of [38, 187], is given in this section. Informally, the section first attempt to interpret the English definite description in the sentence

$$\textit{the F is G}$$

in $\mathbb{IO}$ monads:

$$\mathbf{the\ F} : \mathbb{IO} \quad \mathbf{F}$$

$$\textbf{the}\ \ \mathbf{F}{=}\ \lambda x : \mathbf{F}.\mathbf{G}(x) \wedge \mathbf{sing}(x).$$

where **sing**(x) means x is a singular object. This formalization can be developed further by synthesising the ι operator to interpret the definite descriptions in [216, 71, 334, 335] as the μ operator for recursive definitions, or as $\mu v\lambda 2$-calculus, in [38] which is studied in [336, 337, 338]. Thus, an alternative interpretation of the definite description in the above sentence is

$$\textbf{The}\ F : \mathbb{IO} \quad \mathbf{F}$$

$$\textbf{The}\ \ \mathrm{F} = \mu x.\lambda c.x : \mathbf{F}\ \text{in c}.\mathbf{G}(x) \wedge \mathbf{sing}(x)^{1}.$$

A related study for research is given below. The most closed one is the research by [63] which also derives a novel definition of input and output types as $\triangleright$ and $\ltimes$, respectively. The types are described in λ terms and similar to the first attempt, rather than to the second attempt, to derive them from μ and v terms. A recent related research is by [83] and a broader related research is an ongoing definition of the input/output logic in by [339] which can be linked to the deontic logic.

[54] is also another related study with a proposed calculus for the definite description with exceptions. However, he did not spell out the exception in the case of plurality as in [334]. The difference between this proposed solution and [54] is only in language expressions since both parameterized

[1] in the functional programming language Haskell, μx means the initial function, or an input function, such as *getchar*. The terminal or output function *putchar* by [38] is the vx

monads and their framework are based on Hoare's logic. This language choice also helps us to reveal the rich structure of monads with the compositional principle. Notably, the presupposition projection is able to be parsed in both the research by [54] and a recent one by [16].

In addition, [83] studied the possessive definite description phenomenon, such as *the rabbit in the hat*, in the dependent type theory. Alternatively, its interpretation in $\mathbb{IO}$ monads is given as follows

$$\text{the rabbit in the hat} : \mathbb{IO}\ \textbf{Rabbit}$$
$$\text{the rabbit in the hat} = \lambda c.\mu\ \textbf{r: Rabbit}.\mu\ \textbf{h: Hat.in\_the\_hat(c,r,h)}$$

If we characterize this phenomenon as the domain restriction phenomenon, its formalization in parameterized $\mathbb{IO}$ monads is similar to the demonstrative one in the next section. Besides, the composition principle for the above possessive definite description is performed as usual with additional syntactic rules. For example, the sentence *The rabbit in the hat is bald* has a derived interpretation as follow

$$\left\{\begin{array}{c}
\text{the rabbit in the hat} : \mathbb{IO}\ \textbf{Rabbit}\\
\text{the rabbit in the hat} = \lambda c.\mu\ \textbf{r: Rabbit}.\mu\ \textbf{h: Hat}.\textbf{in\_the\_hat}(\textbf{c,r,h})\\
\textbf{is\_bald: Rabbit} \to \textbf{Boolean}\\
\textbf{is\_ bald}(x) = \lambda c.\text{if } (\text{x} = \cdots) \text{ in c then } \top \text{ else } \bot \text{ (i.e pure function)}\\
\textbf{the rabbit in the hat is bald} : \mathbb{IO}\ \mathit{Boolean}\\
\textbf{the rabbit in the hat is bald} = \mathit{do}\\
r \leftarrow \text{the rabbit in the hat}\\
v \leftarrow \textbf{is\_bald}(r)\\
\text{return } v
\end{array}\right.$$

[2]

Table 9.1: Possessive definite description in $\mathbb{IO}$ monads.

In the author's opinion, the ι notion in [334, p. 360] can be regarded as the $\mathbb{IO}$ monads. Hence, the interpretation of the definite description **the** in [334, p. 380] is synthesised to the above interpretation. Kinds, in Chierchia's sense, are close to types in this framework. However, they are not the same because each object in a type in type theories requires a canonical object formation, as discussed in [86]. On another hand, this restriction is not required in Chierchia's framework.

The denotational semantics for $\mathbb{IO}$ monads is discussed in [38]. According to [187], this semantics lacks strength on concurrency. However, I argue that it is a good candidate for further development for reasoning

[2] v is similar to the output function in [38], which is the *putchar* function in Haskell.

with the $\mathbb{IO}$ monads. For example, recent research attempts for the reasoning of a computer program with $\mathbb{IO}$ monads can be found in [50, 249, 240].

Accordingly, the operational semantics approach to $\mathbb{IO}$ monads in [187], with its description in type as

world -> (a,world)

is a special case of an abductive reasoning with an inference rule

$$\frac{\Omega}{(a, \Omega)}$$

This line of research has a reasoning framework of $\mathbb{IO}$ monads if we view the *update* function as an abduction inference by Plotkin's research [268].

Finally, parameterized $\mathbb{IO}$ monads are similar to the teletype $\mathbb{IO}$ in [38, 50]. The denotational semantics of the $\mathbb{IO}$ monads is based on denotational semantics on $\mu v \lambda 2$ calculus by Gordon [38]. On the other hand, the operational semantics is studied by Jones [187]. In a short comparison, the denotational semantics is a good approach for reasoning with the monads, while the operational is proper for concurrency. Linguistically, denotational semantics is suitable for semantic interpretations, while operational semantics is better at capturing plurality.

9.2 The complex demonstrative in parameterized $\mathbb{IO}$ monads

9.2.1 Introduction to the complex demonstratives

9.2.1.1 The definition of the complex demonstrative

The recent research by [340] provides the cutting edge perspectives on the demonstrative. This section overviews the definition of the English complex demonstrative in the literature. The definition, according to [341], is a linguistic expression which has a formation *that* N or *this* N where N is a common noun. Common nouns have an identity property which enables comparison, recognition, or quantification. This definition is similar to the observation of Bennett in [203, p. 527] that only places are being demonstrated, and an explicit or implicit common noun phrase is required for the demonstration's actions.

For example, let us given a situation in which we are talking to a robot, or translating between languages, and we utter

That car is moving to us.

The complex demonstrative is *that car*, and its meaning must be understood to interpret the semantics. The difficulty is that its meaning depends on the context of the utterance. Therefore, building a formal representation for demonstratives is an aim worthy achieving. It is problematic because its semantics is not fixed and depends on time, location, social situations, and other parameters.

[342] define the notions of nominal-policing and appropriation to interpret complex demonstratives. Their examples of complex demonstratives are *this cat* or *that glove with a hole*, and the associated nouns behind the demonstratives *this* and *that* are called nominals, viz. *cat* and *glove with a hole*. The term *nominal* is perhaps similar to that in [107], which has a broader meaning than *common nouns*. However, its identity properties have not been discussed.

Their idea is similar to ours which, focusses on common nouns rather than on the demonstratives themselves (*this, that*). Their nominal-policing functionality is similar to the recognition property in a common noun. So, instead of saying *nominal-policing*, we say *common-noun recognition*. However, they advocate the semantics of nominal-policing more strongly than we do, since it guards the semantics property of the complex demonstratives. More clearly, if the object σ fails to be policed in *that* F, then the complex demonstrative *that* F has no semantic value despite it having been uttered.

The role of nominals in complex demonstratives has been discussed in [342]. Their philosophical viewpoint is similar to [58, 343], and ours in an opinion that nominals play a central role for the complex demonstrative in natural language contexts. Philosophically, it may not be necessary; however, in daily conversations, the ability for object recognition is crucial for efficient language use.

Besides the *nominal-policing* definition, [342] introduced the notion of *appropriate*. Intuitively, it is similar to the notion of *similarity* in [163]. For example, if a speaker utters *that car*, and points to a car, by demonstration gestures, or by intentions, then an object *car* is *appropriated*. If (s)he says *that car* and points to a *boat*, then it is not appropriated and predicts that the utterance will lead to semantic failures. Thus, they emphasize the role of *appropriation* as a fundamental linguistic notion.

Notably, Grudzińska [343] presents arguments that demonstrative description, i.e. common nouns in the complex demonstrative, should not be analyzed in semantics. It is a pragmatics problem, and is analyzed in the conventional implicature aspect. She criticises the semantics approach to interpret complex demonstratives by interpreting the word *that*, and proposed that the complex demonstrative should be analysed via the conventional implicature. The conventional implicature or a pragmatic perspective is similar to our opinion if it is the familiarity process of finding the object in a complex demonstrative. In addition, Grudzińska declares 5 criteria for the complex demonstrative:

- (I) reference, or rigid designators, in Kripke's sense: e is a rigid designator if, in every possible world, it designates the same object; and it is a nonrigid or accidental designator if that is not the case.

- (II) vacuous use: The gaps are cases where there is a reference failure.
- (III) no negation: The utterance, "That F is not F" is infelicity, i.e. nominal-policing.
- (IV) entailment: "Necessarily, if that F is G, then something is F".
- (V) scope-reading: no descriptive entailment in "Necessarily, if that F exists, it is F".

Grudzińska analyses the semantical approaches such as quantification by [344] and [345], modality by [346, 347], by these 5 criteria. Then she proposes that the conventional implicature approach by [348] satisfies these criteria, and is feasible to use as an alternative approach besides the semantic one.

9.2.1.2 The problem with direct references

The pioneering research on the logic of demonstratives in [203] has significantly shaped the topic. However, there are limitations of the theoretical framework because it is based on the first-order logic. The criticism focuses on the semantic representations of objects in direct references. An example of such limitation is explored in the work on complex demonstratives in [345], or on covariation in situation semantics in [292].

In the logic of demonstratives, [203] follows [62] to assert that linguistic expressions have both a sense and a reference. A sense and a reference are contents and their referred physical entities in the real world, respectively. However, there are several expressions that have only senses and no references, or the references are vague. Perhaps one of the reasons is that reference-resolution is time-consuming and not relevant to a conversation. One case of that reason is explained as a vacuous use in [343]. Her example is the following sentence:

That murderer of Smith is insane.

A detective can make that utterance despite the fact that (s)he does not know exactly who performed the action. Another example is the case when a woman has the title *Mrs*; people assume that the lady has a husband. However, we do not need to know exactly what his name is. Another example is when we are travelling and see a beautiful historic building, we may utter

The person who designed this building was a genius.

In this situation, we do not need to know who the designer is to express our respect to them. [292, p. 2] also proposes an example of a covarying phenomenon that is problematic for direct references:

Mary talked to no $[\textit{senator}]_1$ *before* $[\textit{that senator}]_1$ *was lobbied.*

[that senator]$_1$ is referred to a senator in [senator]$_1$. However, we cannot find any physical entity that represents a senator in [senator]$_1$ since it is a conditional sentence. It is a problematic example of the interpretation of the direct reference approach in [203]. Indeed, Kaplan also criticizes his approach in [349]:

> Lately, I have been thinking that it may be a mistake to follow Frege in trying to account for differences in cognitive values strictly in terms of semantic values. Can distinctions in cognitive value be made in terms of the message without taking account of the medium? Or does the medium play a central role? On my view, the message—the content—of a proper name is just the referent. But the medium is the name itself.

However, the author argues that there is nothing wrong with following [62]. If we understand Kaplan's research as discussed in [106], the problem is centred on there not being enough flexibility in Kaplan's context analysis; it is a gap between philosophy of languages and pragmatic linguistics. In the philosophy of languages, the contexts are possible worlds and physical worlds, while in pragmatics, the context of conversations is information. The physical worlds are hardly changed while the information does change during the conversation. An example of an attempt to improve Kaplan's research in that direction is [274].

9.2.1.3 Previous approaches

While complex demonstratives are widely used, there are only a few theories to provide a complete semantic analysis to the phenomenon. According to [292], besides his situation-semantic interpretation, there are only two theories for unifying the semantics of English demonstratives. They are theories of King [345] and Robert [350] which use generalized quantifiers and dynamic semantics. In the author's opinion, they are the static and dynamic approaches to the phenomenon, respectively.

Dynamic semantics regards the meaning of conversations as information exchanges in the conversations' contexts. Indeed, the research by [111, 56, 27] show that linguistic phenomena in static semantics can be captured in dynamic semantics . The idea is to change the entities' properties in the static case, to be the discourse properties. This works because we are analysing a sentence according to the situations in which it is uttered rather than for its universal characteristics. For example, the complex NP *the rich man* is evaluated at the context of the time of evaluation, not over the whole of human history, nor the whole population. The dynamic approach is not limited by its theoretical framework in the same way as the static approach. By using the *update* function, the dynamic approach captures the pragmatic phenomena adequately. In general, it is the BHK interpretation, as discussed in [351, 297, 295, 296], which states that the interpretation of a term in a logical formulae is equivalent to its interpretation in the contexts.

The problems discussed above can be seen as the improper handling of information in [62]. It is improper in the sense of making an assertion that every linguistic expression has a direct reference. While this is true in a perfect situation of semantic analysis, how relevant it is to real-life situations? This matters because human conversation exchanges information, shares feeling, or even does nothing at all, rather than finding truth values of an utterance. For example, we can take a sentence from [163] to illustrate the point

Bill saw a lion on the street. He claim that the lion had escaped from the zoo.

Of course, the speaker and listener are both referring to a lion but it does not have a concrete identity. To resolve the problem, we can employ the dynamic semantics viewpoint. It has been applied to the phenomenon by [163] who used context-change potential as a dynamic semantic framework to develop her theory of the demonstrative. She interpreted a demonstrative as definite descriptions, and associated the meaning of a demonstrative to its presupposition. Then, a presupposition is formalized in the context-change potential in [111], which is based on the possible-world semantics in [180]. Roberts uses familiarity and uniqueness properties of objects as primary presuppositions in accordance with the demonstration act.

Her theory also develops a taxonomy of the familiarity process. She divides familiarity into a strong familiarity which appears in the preceding utterance, and a weak familiarity, which is variance by salient variables. The familiarity and uniqueness properties of definite descriptions are discussed in [92] as the existence and uniqueness properties of definite descriptions. Her innovation is associated them with the context in [163, p. 31] as follows:

> The existence and uniqueness presuppositions of a definite descriptions are not claims about an individual in a model, but about a discourse referent in the domain of discourse. Thus, the existence presupposition amounts to (a variation on) [111] Familiarity Presupposition for definite NPs, and the uniqueness presupposition is about the status of the familiar discourse referent in the Domain of the Context of discourse.

In addition to Robert's theory, there have been other noteworthy modifications. The theoretical framework in [292] reflects Robert's theory with a reinterpretation in situation semantics by [47], rather than in possible world semantics. Wolter's theory [57] goes further by asserting that demonstratives are different from definite descriptions as the former affects and changes the context.

Besides the main dynamic approach to the phenomenon is the static approach by [345] [3]. His hypothesis is that demonstratives are generalized

[3] Basically, it is the higher order logics

quantifiers in the sense of [108]. His interpretation of a demonstrative *that* is

______ *and* ______ *are uniquely* ______ *in an object* x *and* x *is* ______

The first and last slots are for NP and VP, respectively. The second and third slots are for the recognition or familiarity in Robert's sense, and reference or perception function, respectively. His innovation stays at expressing the referencing function through the speaker's perceptual. For example, Elbourne's sentence

That animal [Pointing at Flossy] is a donkey.

has an interpretation in generalized quantifiers with possible-world semantics as

animal and $=$ *Flossy* are uniquely jointly instantiated in w, t in an object x and x is a *donkey*.

The detailed analysis of the problems that each theory poses is covered in [292]. For the generalized quantifiers theory, that is a rejection of Russell's idea that definite descriptions are existence objects. It seems true, especially in the view of Hilbert's ϵ-calculus which is described in [142, 283]. In dynamic semantics, it is the contradiction reading between sentences or subsentences.

9.2.2 The context domain restriction interpretation of the single complex demonstrative

This interpretation defends the dynamic approach to the demonstrative in conjunction with [163, 57, 292]. To align with it, I provide an alternative dynamic semantic framework as discussed in Chapter 7. In this section, I will explain how this dynamic framework and Wolter's hypothesis of the distinction between the demonstrative and definite descriptions is synthesised through the parameterized $\mathbb{IO}$ monads in section 5.3.6. Wolter's hypothesis [57] is that the demonstrative is a context-domain restriction phenomenon. Hence the demonstrative narrows the context while other phenomena, such as definite descriptions, do not.

According to [203], demonstratives are interpreted as a direct reference as $dthat[\delta]$. The problem with this interpretation is the direct reference as discussed above. However, we can have a semantic interpretation of the complex demonstrative in the sentence *That F is G* in [58] as follows:

$\mathbf{that}(\mathbf{F}) \coloneqq \mathbf{the}(x).(\mathbf{F}(x) \wedge \mathbf{G}(x))$

Where **the** is an interpretation of a definite description. According to [292], definite descriptions have the following interpretation in situation theory

$[\![\mathbf{the\ cat}]\!] \coloneqq \iota x.\lambda P.\mathbf{cat}(x) \wedge P(x).$

The definite descriptions is interpreted in the $\mathbb{IO}$ monads in the previous section 9.1. On the other hand, I propose a solution for the demonstratives, which is based on the research by [57], in parameterized $\mathbb{IO}$ monads as the derived formulae

$$\textbf{that } \mathbf{F} : \mathbb{IO} \quad (M) \quad (N \wedge M \neq N) \quad \mathbf{F}^{4}$$

$$\textbf{that } \mathbf{F} = \mu x : \mathbf{F} \wedge \mathbf{G}(x) \wedge \textbf{sing}(x).$$

Where **F** is a familiar object of **that F**. The dynamic condition $M \neq N$ is derived from Wolter [57]. Her research states that demonstratives are different from definite descriptions, as the former trigger the changing of the context, while the latter do not. Formally, demonstratives differ from definite descriptions in the sense that the latter require no change in the state. Explicitly, an interpretation of definite description **the** in parameterized $\mathbb{IO}$ monads is

$$\textbf{the } \mathbf{F} : \mathbb{IO} \quad (M) \quad (M) \quad \mathbf{F}$$

$$\textbf{the } \mathbf{F} = \mu x : \mathbf{F} \wedge \mathbf{G}(x) \wedge \textbf{sing}(x).$$

In the previous section with an interpretation of the definite description in

[4] In the current state of the art, the states have a limited formation or apply in limited circumstances. The states are the teletype $\mathbb{IO}$ in by [50, 38] or 25 situations in according to [352]

$\mathbb{IO}$ monads, we only saw a type $\mathbb{IO}$ $\mathbf{F}$ without pre- and post-states. In addition, the above formula also differs from the formulation of definite descriptions in [333], at the point of stating the dynamic condition. The dynamic condition asserts that the context is not changing with inserted definite descriptions.

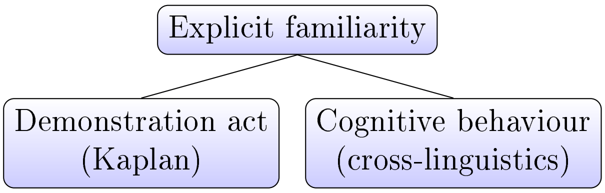
9.2.2.1 The difference between demonstratives and definite descriptions

There is a close link between the research on demonstratives and the research on definite descriptions because they are both referential expressions. Researchers such as [292] support the idea that demonstratives are definite descriptions. Elbourne argues that English demonstratives 'this' and 'that' are definite articles in the sense of providing existence and unique presuppositions.

In the author's opinion, the difference between demonstratives and definite descriptions is that the former is the *explicit substitution* by the demonstration action of the latter. In other words, the substitution is an *explicit similarity* or a familiarity in Robert's sense. A related approach by [353] also provides that the demonstrative refers to the most salient object through the descriptive content in the context, while definite descriptions refer to the unique object by their descriptive content. This idea differs from [350] and [292] where demonstratives are classified as definite descriptions. In the author's opinion, the explicit substitution provides a distinguished characterization between two phenomena: demonstratives and definite descriptions. they are the same at the conceptual level but different at the expressing level.

The objection to treating demonstratives as definite descriptions is also supported by Nowak [58] and Wolter [57]. According to Wolter [57], the demonstrative narrows the scope of familiarity objects, while definite descriptions infer the universal property.

In the author's opinion, explicit familiarity is similar to anchoring in [163, p. 42]. The explicit familiarity is represented as a demonstration, i.e. an act of demonstrations in the logic of demonstratives in [203]. Furthermore, explicit familiarity can be expressed through other forms of cognitive behaviours.


Explicit familiarity
Demonstration act (Kaplan)
Cognitive behaviour (cross-linguistics)

There is also another interesting viewpoint in cross-linguistics on the difference. Universally, we may agree that both demonstratives and definite descriptions are both referential words. However, the demonstrative may have different word order constraints from those of definite descriptions. Thus, if we identify demonstratives with definite descriptions, they are still subject to different word-order interpretations. For example, in Swahili, as recorded in [354]

a. yule mtu (Swahili)

the man

'the man'

b. mtu yule

man that

'that man'

9.3 The conventional implicature in parameterized monads

The literature review of the conventional implicature phenomenon has been discussed in section 2.2.4. [20, 79] proposed theoretical frameworks to parse the phenomenon by following the principle that the at-issue and the dimension issue do not interact. However, the recent research by AnderBois et al. [80, 355] provided counterexamples to that principle. Hence, this part is going to discuss the phenomenon at an alternative perspective and the next section provides an alternative framework to capture the new empirical observation in parameterized monads.

The definition of conventional implicature in this research was provided in section 2.2.4. For example, in the below sentence

Jake, who almost killed a woman with his car, visited her in the hospital.

[78] proposed that the phrase *who almost killed a woman* is being analysed in a separate dimension rather than the usual analysing context of the sentence. Indeed, the phrase is not a necessary contextual presupposition nor a prerequisite needed for an interpreter to understand the sentence. Its functionality is to provide *additional information* to the sentence. Therefore, he suggested that the additional information should be analysed in another context rather than in the traditional context. The new context is called a *conventional implicature* (CI) context, and it is parallel with the context of the ordinary sentence, which now changes to become an *at-issue* context.

The interesting point, according to [78], is that the *at-issue* dimension can be used in the CI dimension, but not the opposite direction. However, the interaction between the two contexts are not that separated. The pragmatic examples below from [355] show cases in which the CI and the *at-issue* dimension interact by discourse phenomena-related issues such as presuppositions, anaphora, VP ellipsis, and nominal ellipsis. They cross, at their boundary, from $CI \to$ *at-issue* and *at-issue* $\to CI$:

1) **Presupposition**

a) *John, who wouldn't talk to Mary, wouldn't talk to* ***Susan either****.*

b) *John, who wouldn't talk to Mary, wouldn't talk to* ***him either****.*

2) **Anaphora**

a) *John, who had been kissed by Mary, kissed* ***her too****.*

b) *John kissed Mary, who kissed* ***him too****.*

3) **Ellipsis**

a) *Melinda, who won three games of tennis, lost because Betty won* ***six****.*

b) *Melinda lost three games of tennis to Betty, who lost* ***six*** *to Jane.*

4) **VP ellipsis**

a) *Mr. Gore at first believed the president, and even defended him to Tipper and his daughters,* ***who did not****.*

b) *So Lalonde, who was the one person who could deliver Trudeau,* ***did****.*

In the above sentences, the phrases inside the commas are called the *side issues* or conventional implicature, as they provide *additional information* for the readers/hearers rather than the fact at the *believe level* of the speaker. According to Potts [78], we separate the sentence into multiple dimensions rather than interpreting them in an single dimension. For example, we separate sentence 2a into two parts:

John kissed her too (at issue) and *who had been kissed by Mary.* (CI)

Potts defined the principle that the *at-issue* is reusable in CI but not vice versa. On the other hand, [355] argues against this principle by providing the above examples as a crossed boundary between two dimensions of *at-issue* and CI. They propose to interpret the sentences as a single dimension rather than on multiple dimensions. However, this approach also leads to an ambiguous example in [20], as below.

Luke Skywalker is so gullible that he believes that Jabba the Hutt, a notorious scammer, is a trustworthy business partner.

If we interpret the sentence in a single dimension, it would lead to a contradicting semantics that *Luke Skywalker is so gullible that he believes Jabba the Hutt is a trustworthy business partner* and *he is a notorious scammer.* The contradiction comes from the semantic meaning that Luke Skywalker is both a trustful business partner and being a notorious scammer. This evidence show a strong case that we should separate the propositions into two distinct discourse segments.

In the next session, we will use session types in section 5.3.6 to model the interaction between the two dimensions. We view these dimensions as an interaction channel between the client and server computers. Session types provide a channel with an explicit type for exchanging data between the two dimensions. The type of the data is not able to be captured by general state monads.

9.3.1 The conventional implicature in session types

This section proposes to use *session types* in [23, 247] as a mechanism to model the conventional implicature's phenomenon in [78]. This proposal synthesises the research results in [78] and [355]. Specifically, we keep the multidimensional analysis by [78], while allowing resources to be exchangeable between the CI and *at issue* dimensions as in the analysis by [355] during the sentence composition. Thus, we do not leave it as a discourse phenomenon, nor as post-compositional process as done in [20].

Firstly, let us reintroduce a session type in section 5.3.6.2 and from [23]. Its original version is described as the π calculus by [246]. Let X, X_1, X_2 be a collection of sets of values for input/output (IO). The states are abstract traces of a program's possible IO behaviour; we call these sessions. Their regular grammar is:

$$\mathcal{S} = ?X | !X | S_1 + S_2 | S_1.S_2 | \circ$$

A session $?X$ means that the program must input, or read, a value in X. $!X$ means that the program must output, or write, a value in X. $S_1 + S_2$ means a choice operator in which a program performs either session S_1 or session S_2. A conjunction operator $S_1.S_2$ means that the program must perform session S_1 before next performing session S_2. $\circ$ means the program does nothing.

The state relations, or arrows in $\mathcal{S}$, are the combination of the above grammar, with atoms (or formulae) being $?X, !X, \circ$, a coordinator $S_1 + S_2$ and an implication $S_1.S_2$.

From the above session type, we have infinitely many sessions with states described as sessions. Consider a function *sum* from [23], which takes an input of two integers and outputs their sum if it is greater than 10, or outputs nothing otherwise. It has the declared type and denotation as below.

$sum : \mathbb{M}\ (?Int.?Int.((!Int.\circ) + \circ))\ (\circ)\ 1$

This parameterized monad *sum* has a consequence reading. Firstly, we are taking a pre-state or a session of type $(?Int.?Int.(!Int.\circ +\circ))$. It receives, or reads, two integers, outputs one integer, and performs a choice operator + of a function without any input or output, i.e. the pure calculation plus $(+)$. A post-state, or a final session, has a type $\circ$ with the return value being a unit, i.e. a termination if the computation successes.

From the declared type for the session type of *sum*, its operation under the servers' perspective is illustrated as following sum : $\mathbb{M}\ (?Int.?Int.((!Int.\circ) + \circ))\ (\circ)\ 1$

sum =

do$x \leftarrow sent_{Int}, ?Int.((!Int.\circ) + \circ)$

do $y \leftarrow sent_{Int}, (!Int.\circ) + \circ$

do $z \leftarrow (x + y; \star)$

if $z > 10$ then

do $\star \leftarrow (\odot; (!Int.\circ) + \circ \Rightarrow !Int.\circ)$

$receive_{Int,\circ} z$

else $(\odot; (!Int.\circ) + \circ \Rightarrow \circ)$

$1; \circ$

Table 9.2: An example of an operational operator in session types.

where *sent, receive* are primitive functions of the *read* and *write* operators with associated types

$sent_{X,S} : 1 \mapsto \mathbb{M}\ ?X.S\ S\ X$

$receive_{X,S} : X \mapsto \mathbb{M}\ !X.S\ A\ 1$

The *if-then-else* is the usual logical structure. The $\Rightarrow$ performs the choice operator between two choices of returning $(!Int.\circ)$ or $\circ$. in the case of $\star$, the choice is $(!Int.\circ)$, otherwise, it is $\circ$.

Alternatively, we will elaborate the idea more with an example from [247] that takes place in an internet shopping context. The shop provides two operations: giving a list of books, or checking out by collecting credit card numbers and their associated addresses:

Shop = & ⟨add: !book.Shop, checkout: !Card.! Address. ∘⟩

The syntax structure &⟨⟩ explains that the shop has two sub-functions: adding a book, or checking-out to buy. the & symbol means a *branching* with 2 options *add* and *checkout*. The adding function is processed as follows: we receive a book name and then return to the main shop function to continue to perform the choosing between two options. Otherwise, we will checkout by receiving the credit card number and addressing and finishing our shopping session.

We can perform these actions under the assumption that we have shoppers or users over the internet. As clients, they would provide the names of interested books, followed by a checkout process in which they provide personal card numbers and addresses. Our example correspondence is

Shopper = ⊕ ⟨add: ? book.Shopper, checkout: ? Card.? Address. ∘⟩

⊕ means *choice* between two options. The two above protocols are working if we ensure some additional properties of the processes: they are *compatible* as both shop and shopper are operate on an agreement assumption, and will not terminate prematurely by small mismatches between two parties. In addition, the protocol is terminated at the end, and the shopper can only perform on *selected* actions. The process of selecting books should be performed before checkout and, during the checkout process, both card number and address are given, in that order. If the shopper chooses an option of adding a book, she sends a book's name; and after accepting the adding option, a shop expects a book's name. The requirements are similar for checkout process.

The π calculus in the above examples, described in [23] and [246], provides a model to resolve the controversy in conventional implicature posed by [78] and [355]. Firstly, Potts strictly divided the sentence into multiple dimensions and forbade the interaction between the CI and at-issue dimensions. Anderbois and Brasoveanu and Henderson, on the other hand, gave counterexamples and suggest that we should analyse a sentence in a single-dimension version. We provide a solution to unify their analyses into the information flow section of π calculus, in which each phenomenon, such as anaphora resolution, is a task of exchanging data. Hence, we keep the interesting and insightful multi-dimensional analysis while still capturing crossed-boundary linguistic phenomena. In our view, it relates to the problem of modelling the relation between two asymmetric parties such as servers and clients or database's query.

We can directly use session types to model the interaction between the *at-issue* and CI dimensions. Analogically, the CI dimension is the server side, and the *at-issue* dimension is the client side. We use *commas* to separate between dimensions. For example, we are going to demonstrate the above example by [20]

John, who likes cats, also likes dogs.

The CI dimension reads an input as a man, which is *John*, and returns a proposition which is *John like cats*. The *at-issue* dimension reads the proposition and checks the semantics for the word *also*.

$$comma : \mathbb{M}\ (\mathbf{!j}.\circ)\ (\mathbf{?like\ j\ c}.\circ)\ j$$

$$comma = receive \gg x.\lambda l.l \star \lambda f.sent(f\ x) \star \lambda\_.\eta(x)$$

where the *sent*, or **?** operator translates the state from $?x.q$ to q after sending the value x over the communication channel as per [356]. Its formalization in parameterized monads is

$$sent : \alpha \rightarrow \mathbb{M}\ (?x.r)\ r\ \top$$

$$sent\ a = \lambda c.write\ c\ a$$

where $a : \alpha$ and *write* is the writer monad. Similarly, the definition of receive, or **!**, is

$receive : \mathbb{M}\ (\mathbf{!}a.r)\ r\ \alpha$

$receive = \lambda c.sel\ c$

where $a : \alpha$.

$also : \mathbb{M}\ (\mathbf{!like\ j\ c}.\circ)\ (\circ)\ ((d \to j \to l) \to d \to j \to l)$

$also = \lambda v.\lambda o.\lambda s.s \star \lambda x.v \star \lambda f.o \star \lambda y.receive \gg A \wedge check(\exists z \in A.f\ z\ x \wedge z \neq y) \star \lambda\_.\eta(f\ y\ x)$

$John : \mathbb{M}\ \circ\ (\mathbf{?j}.\circ)\ j$

$John = \eta(j) \gg sent(j) \gg \eta(j)$

We perform the lifting operator on the proper name, *John* twice. One performance is to lift the name to the parameterized monads for an usage by the *sent* operator. However, the *sent* operator just returns the truth value in $\top$ and we want to reuse the name *John* again in the *check* function at the *at issue* dimension. Therefore we lift the name again.

$who : \mathbb{M}\ \circ\ \circ\ ((j \to l) \to (j \to l))$

$who = \eta(\lambda P.P)$

$like : \mathbb{M}\ \circ\ \circ\ (c \to j \to l)$

$likes = \eta(\lambda y.\lambda x.like(x,y))$

$cats : \mathbb{M} \circ \circ (c)$

$cats = \eta(\iota x.cat^{*}(x))$

$likes : \mathbb{M} \circ \circ (d \to j \to l)$

$like = \eta(\lambda y \lambda x.like(x,y))$

$dogs : \mathbb{M} \circ \circ d$

$dogs = \eta(\iota x.dog^{*}(x))$

There are two duality performances between *comma* and *john*, *comma* and *also*. We take a duality as an underlying assumption for session types in [357]. The duality means that, for each sent and receive operator in a session channel, there is a corresponding received and sent from the opposite side of the channel. We also assume that the channel or session is closed after the data is interchanged. Hence, the derivation is demonstrated below where the data exchanged are *john* and a proposition *John likes cats.*

$$\dfrac{\dfrac{\dfrac{\mathit{John}}{\mathbb{M} \circ (?\mathbf{j}.\circ)\, j}\,?\,\dfrac{\mathit{comma}}{\bullet\mathbb{M}\, (!\mathbf{j}.\circ)\, (?\mathbf{like\ j\ c}.\circ) j}}{\mathbb{M} \circ \circ j \bullet \mathbb{M} \circ (?\mathbf{like\ j\ c}.\circ)\, (j)}\,!\quad \dfrac{\dfrac{\mathit{who}}{\mathbb{M} \circ \circ ((j \to l) \to (j \to l))}\quad \dfrac{\dfrac{\mathit{likes}}{\mathbb{M} \circ \circ (c \to j \to l)}\;\dfrac{\mathit{cats}}{\mathbb{M} \circ \circ c}}{\mathbb{M} \circ \circ (j \to l)} \to E}{\mathbb{M} \circ \circ (j \to l)} \to E}{\mathbb{M} \circ \circ (j \oplus \mathbf{like\ j\ c}) \bullet \mathbb{M} \circ \circ l}\,?$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathit{also}}{\mathbb{M} \circ (!\mathbf{like\ j\ c}.\circ)\, (d \to j \to l) \to d \to j \to l}\quad \dfrac{\mathit{likes}}{\mathbb{M} \circ \circ (d \to j \to l)}}{\mathbb{M} \circ (!\mathbf{like\ j\ c}.\circ)\, (d \to j \to l)} \to E\quad \dfrac{\mathit{dogs}}{\mathbb{M} \circ \circ d}}{\mathbb{M} \circ (!\mathbf{like\ j\ c}.\circ)\, (j \to l)} \to E\quad \dfrac{\bullet}{\mathbb{M} \circ \circ (j \oplus \mathbf{like\ j\ c})}}{\mathbb{M} \circ \circ (\mathbf{like\ j\ c} \wedge ((j \to l)\ \ j)) \equiv \mathbb{M} \circ \circ (\mathbf{like\ j\ c} \wedge l) \equiv \mathbb{M} \circ \circ (\mathbf{like\ j\ c} \wedge \mathbf{like\ j\ d})}\,!$$

Table 9.3: Parsed conventional implicature sentence in session types.

•, in Potts' sense, is the separation between two dimensions. The left of the • is an *at-issue* dimension while the right is the CI dimension. In other words, the left is the client and the right is the server. The interaction of *John* and *comma* is the parameterised monads composition, whereas the composition between *comma* and *who likes cats* is the normal composition. $a \oplus b$ is an object which is both a and b.

The detailed implementation of session types in parameterized monads is available in [356]. The other properties of session types, such as duality, recursion, have not been explored in linguistics. There are still limitations on this approach such as exception-handling or error-handling, as the interactions between dimensions are not yet properly handled.

9.4 Discussion

This chapter has used parameterized monads to parse the definite descriptions, demonstrative, and conventional implicature phenomena. The parsing of the demonstrative phenomenon shows an expressive power and dynamic semantic requirements for a theoretical framework in which the parameterized $\mathbb{IO}$ monads have. The rich expressive power is required to express Wolter's hypothesis [57]. Her hypothesis means that the demonstrative differ from the definite description in the sense that the former is the contextual restriction phenomenon. Finally, the conventional implicature phenomenon shows an application of parameterized monads through session types.

In addition, Grove [16] shows how to interpret presuppositions in graded monads. If we also take Roberts' [163] viewpoint of demonstratives as the definite description and presuppositions, then synthesising [163, 16] and section 9.1 also provides another approach to parse demonstratives in $\mathbb{IO}$ monads. Finally, if a state is being viewed as a situation in situation semantics as in [47], this framework is similar to the situation semantics approach to the phenomenon in [292].

The research of [358] is related to us and it shows how to interpret the phenomenon in type theories with a dialog system from [118]. However, we do not have the interpretation of a dialog system in monads yet. Instead, the advantage of our approach is an ability to have a clear expression of how contexts change during an utterance. Finally, Nowak [58] also has a similar perspective to ours. However, we extend his research by providing a framework to express his idea. Since we use the typing to express the similarity, the typing formula $x : \mathbf{F}$ expressed a familiar process of finding object $\mathbf{F}$ rather than stating it plainly as $\mathbf{F}(\mathrm{x})$.

[359] also provided a database approach to the phenomenon. Our approach is different from theirs in the sense that our representation is more dynamic while their approach is more static-oriented. We are focusing on the changing of the situations during the use of the demonstrative. We can use their study of linguistic features of the demonstrative, such as formality, gender, and number to enhance our framework by *predicting* a specific

transition of the state (or situation). Furthermore, the syntax of the complex demonstrative, such as *that book*, could change according to the language that is being used. For example, Wilkins records 25 situations in which the demonstrative changes [352]. The usage of the demonstratives, hence, is to restrict the state/information for the interpreter belong to the referential functionality.

Furthermore, we improve the interpretation of the conventional implicature by using writer monads in [20] as session types in parameterized monads. The advantage of this move lies in the ability to exchange data between two separated dimensions, which fits with the empirical observation of the phenomenon in linguistics. Observation shows that the data is indeed exchanged between the CI and at-issue dimensions of the phenomenon [355]. The data, for example, are common linguistic phenomena such as pronouns, presuppositions, anaphoric expressions, etc. This cannot be handled by the writer monad approach in [20]. The writer monad separates dimensions but does not allow exchanging data.

A related study for the interpretation of the conventional implicature is by Marsik [10, p. 134-137]. He used the effects and handlers framework to parse the phenomenon. However, he was not concerned about information-exchange in his framework.

Finally, substantiating the study of the linguistic structures of states in Chapter 6 with the interpretation of demonstratives in section 9.2 can brings new perspectives on other topics in demonstratives such as multi-occurrences or plural demonstratives. The interaction of properties of sessions types, such as duality, recursion, and exception handling with linguistic exchanged data phenomena in the conventional implicature phenomenon are also prominent research directions. Alternatively, it is good to see how the research of information flows, a well known application of session types in computing, flourish in linguistics.

Chapter 10

Conclusion

The theoretical background of this dissertation is category theory by [200, 30, 31, 29] with recent developments in monads by [33, 8] and parameterized monads by [23, 34, 35]. Hence, this dissertation applied the theory in linguistics as a new field of study to parse and analyze linguistic phenomena under the intersection between semantics and pragmatics such as the donkey anaphora, the conventional implicatures, the demonstratives, and the imperatives phenomena. Notably, this dissertation regards the parameterized monads as a dynamic approach in formal semantics that re-interprets the cDRT framework by [24]. Thus, the summary of the research is given below with the limitation and future research in section 10.1

It is well known that λ-calculus is not an adequate linguistic semantic framework. Hence, for its semantic interpretations, traditional research in formal semantics usually combines the calculus with other theories, such as model theory with possible world semantics by [180]. An example of this framework is the intentional semantics by [4]. However, some questions remain, which are neglected by formal semanticists, in the background theories. For example, set theory, which model theory is based on, has been recently criticised for its linguistic applications by [68, 28, 69].

In this dissertation, the author proposes category theory as an alternative for the traditional theories for parsed natural languages [1] based on parsing as a deduction hypothesis stating in chapter 2. In a similar manner, [30] proposed category theory as an alternative to set theory in philosophy. Previous research, such as [182, 29, 30], show that it has a natural and compatible treatment with λ-calculus through the Curry–Howard–Lambek correspondence.

Since there is a duality between category and type theories; category theory has a potential to provide a framework to modularise a variety of linguistic semantic models in λ-calculus into a single framework. In the author's opinion, the strength of type theories is that they provide general frameworks, while category theories provide a more phenomena-oriented approach to linguistics.

[11, 17, 16, 12, 15, 21] are regarded as the main stepping stones leading to this dissertation. In particular, the parameterised monads are oriented towards Hoare's logic, and can interpret the composable continuation by

[1] This idea has also circulated in literature since Lambek. However, ours is more substantive and based on recent critics of the use of the set theories as a foundation for linguistic theories

[7] in monads based on previous research by [9, 53]. Hence, parameterised monads provide an alternative framework in comparison with the composable continuation framework by [6, 75, 12].[2]

By studying the donkey anaphora, we showed that parameterized monads can handle sentential dependency. Thus, this framework has expressive power equal to related frameworks such as typed logical grammar by [26], dynamic predicate logic by [27], and typed predicate logic by [28].[3] This framework also supports multi-level sentencing analysis, and proposes the *swapping technique* in Section 7.4.2 to handle the scope of an existence variable. This technique is based on the BHK interpretation; hence it proposes an analogy between scope-taking in linguistics and proof-search in logics.

The framework in this dissertation is capable of interpreting the cDRT framework (Sections 7.1 and 7.2). Hence, it provides a compositional approach to dynamic semantics with a strong mathematical foundation. This foundation has not been studied in the cDRT, and it uses Hoare-style logic

[2] The generic notion of continuation has been introduced to linguistics in [220, 149, 166], and its relation with monads is discussed in [360]. In categorical grammar, the equivalent notion of parameterized monads is the Lambek–Griffin calculus as discussed in chapter 11 of [12].

[3] Other formalisms such as type logical grammar in [113] and [6, p. 150] have not been studied by the author. A brief view is that they provide better formalization of context as highlighted in [141, 82]. The presupposition phenomenon is a special case of the context and formulae relations.

rather than Dijkstra's weakest-precondition calculus[4]. Furthermore, it uses session types to model the interaction between the at-issue and conventional implicature dimensions in the conventional implicature phenomenon in Section 9.3. Previous versions of the writer monad framework [20, 21] could not handle this property. Finally, it also follows and provides an alternative dynamic semantics framework to interpret the demonstratives (Sections 9.1 and 9.2) in parallel with previous research by [163, 292].

This dissertation also shed light on the logical study of the imperative phenomenon. This analysis is based on previous studies by Pérez-Ramírez and Fox [49], and its contribution in interpreting their framework in parameterized monads with an additional handle of disjunction. Translating their research into parameterized monads provides a dynamic approach to the phenomenon. The phenomenon has also been interpreted using DRT [55], but, by interpreting the cDRT in parameterized monads, we have provided a compositional approach to their research. That is essential since, according to [76], the two main theoretical frameworks used to analyse the phenomenon in linguistics are the dynamic approach by [55] and the modality approach by [77].

Related research includes [16, 10, 46, 17, 6, 15, 21, 19]. [16][5] studied the

[4] See chapter 7

[5] Her research is conducted in parallel with this one and the author was not aware of it when conducting this research.

graded monads and its application in the presupposition phenomenon. The graded monads and parameterized monads are equal [53]. However, the strength of parameterized monads is their capability of interpreting the continuation while this is the weakness found in the grade monads. [6] uses continuation to parse phenomena in linguistic side effects and he criticises other frameworks such as monads as being less expressive than his approach. However, a recent study by [13] rejects this claim and declares that it is still a conjecture. [10] used the algebraic effects and handler by [64]. This is another framework used in the study of effects in computing. Besides that, [28] takes the logical approach. In the author's opinion, this approach has as much expressive power as the contextual modal type theory by [123].

Finally, another related research is the dynamic Montague grammar(DMG) by [275]. This framework is more generalized than DMG since DMG is based on Montague's grammar. The interpretation of DMG in parameterised monads is quite straight forward as this framework is open for grammatical implementation. A closer discussion is given by [12, p. 25–29] which compares DMG with the continuation monad. The research by [122] and [304] also relates to ours. However, our research is distinct from theirs in both the linguistic phenomena studied and in the choice of the theoretical background.

10.1 Future work and limitations of the research

The parameterized monads follow the dynamic approach to interpret natural languages; hence, they bring both the strengths and weaknesses of dynamic semantics. According to [109], the dynamic semantics focuses on the unstructural properties of natural language. Thus, it lacks the rigorous analysis of the static semantics, such as type theories or predicate logic. In the currently accepted research, an analysis of the donkey anaphora phenomenon by [292], shows that static analysis is preferred over a dynamic one. However, in the author's opinion, other phenomena, such as demonstratives, are better understood through dynamic semantics.

Despite its compositional rule inside a monad, monads in general show a limit of combining different monads together. The general approach to solve that problem is the monad transformers by [202], which is used and explored as the background theory for the research in [17]. A further pragmatic study is needed to see how monad transformers and parameterized monads are better at parsing natural languages.

In the parameterized monads, or categories theory, the author's studies suggest that it is feasible to employ category theories as an alternative to type theories or model theory to interpret natural languages within the studies of linguistics. Prominent open problems are listed in the discussion section of chapter 4.

A potential next step is to study the context-changing during the sentential analysis phenomena such as state-switch reference [276] or dynamic pragmatics [323]. In addition, the interaction of imperative and indicative sentences have not yet been studied. Another further research is to investigate the domain restriction phenomena in [303] in the framework, such as the generalized quantifiers in the Chapter 1 of [159]. Others are interpreting the states and related phenomena as collections in [36] rather than a simple treatment of the context as a list of individuals, as was done in this dissertation. Finally, the study and characterization of the ambiguity as an essential and spurious one in compiling techniques in computing has not been explored in linguistics.

Future work should extend the linguistic phenomena studied in this research such as

- investigating the interaction of this framework with maxims in Grice's sense in [361], or Cooper's storage in [304].
- detailed analysis of conditional reasoning.
- context shifting by [73].
- multiple occurrences of demonstratives by [362].

- dynamic pragmatics by [323].
- linguistic phenomena that change the context during sentence analysis, such as switch reference, by [276, 277].
- representation of Nunberg's demonstratives by [363] in [292].

Alternatively, additional potential research directions include

- natural language parsing and its applications in machine translations.
- natural language parsing for speech acts and daily conversations with potential applications in computer assisted writer applications.
- a unifying framework to interpret anaphora resolution. It would open up research directions for indexical and linguistic contextual analysis.
- promote categorical semantics to linguistics.

Bibliography

[1] M. Dummett. *The logical basis of metaphysics.* Harvard university press, 1991.

[2] N. Francez. *Proof-Theoretic Semantics.* College Publications, 2015.

[3] A. Church. A formulation of the simple theory of types. *The journal of symbolic logic*, 5(2):56–68, 1940.

[4] R. Montague. The proper treatment of quantification in ordinary english. In J. Kulas, J.H Fetzer, and T.L Rankin, editors, *Philosophy, Language, and Artificial Intelligence*, volume 2 of *Studies in Cognitive Systems.* Cambridge University Press, 1973.

[5] P. Portner and B. Partee, editors. *Formal Semantics: The Essential Readings.* Wiley-Blackwell, Sep 2002.

[6] C.C. Shan. *Linguistics side effects.* PhD thesis, Harvard university, 2005.

[7] O. Danvy and A. Filinski. Abstracting control. In *ACM Conference on LISP and Functional Programming*, pages 151–160. ACM, Jun 1990.

[8] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[9] P. Wadler. Monads and composable continuations. In *LISP and symbolic computation*, volume 7, pages 39–56. Kluwer academic publishers,

1993.

[10] J. Maršík. *Effects and handlers in natural language.* PhD thesis, de l'Université de Lorraine, 2016.

[11] C.C. Shan. Monads for natural language semantics. In Kristina Striegnitz, editor, *European Summer School in Logic, Language and Information Student Session*, pages 285–298, 2001.

[12] C. Barker and C.C. Shan. *Continuations and Natural Language.* The Oxford University Press, 2014.

[13] Y. Forster, O. Kammar, S. Lindley, and M. Pretnar. On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control. *Functional Programming*, 29, 2019.

[14] I. Heim and A. Kratzer. *Semantics in Generative Grammar.* Blackwell, 1998.

[15] C. Unger. Dynamic semantics as monadic computation. In *8th International Workshop on Logic and Engineering of Natural Language Semantics*, volume 9980, 2012. (LENLS 8).

[16] J. Grove. *Scope-taking and presupposition satisfaction.* PhD thesis, The University of Chicago, 2019.

[17] S. Charlow. *On the Semantics of Exceptional Scope.* PhD thesis, the department of linguistics, NewYork university, 2014.

[18] G. Jäger. *Anaphora and type logical grammar.* Springer, 2005.

[19] S. Martin and C. Pollard. A dynamic categorial grammar. *International Conference on Formal Grammar*, pages 138–154, Aug 2014.

[20] G. Giorgolo and A. Asudeh. Monad for conventional implicatures. In *Sinn und Bedeutung*, volume 16, 2012.

[21] A. Asudeh and G. Giorgolo. *Enriched Meanings: Natural Language Semantics with Category Theory.* Oxford University Press, Sep 2020.

[22] M. White, S. Charlow, J. Needle, and D. Bumford. Parsing with dynamic continuized CCG. In *Proceedings of TAG 13, Association for Computational Linguistics*, pages 71–83, 2017.

[23] R. Atkey. Parameterised notions of computation. *Functional Programming*, 19(3-4):335–376, Jul 2009.

[24] R. Muskens. Combining montague semantics and discourse representation. *Linguistics and philosophy*, 19(2):143–186, 1996.

[25] J. van Eijck and C. Unger. *Computational Semantics with Functional Programming.* Cambridge University press, Sep 2010.

[26] A. Ranta. Gf: A multilingual grammar formalism. *Language and Linguistics Compass*, 3, 2009.

[27] J. Groenendijk and M. Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14(1):39–100, Feb 1991.

[28] C. Fox. Curry-typed semantics in typed predicate logic. In *The Logica Yearbook 2013*, pages 35–48. College Publications, 2014.

[29] B.C. Pierce. *Basic Category Theory for Computer Scientists (Foundations of Computing).* The MIT Press, 1991.

[30] S. Abramsky and N. Tzevelekos. Introduction to categories and categorical logic. In B. Coecke, editor, *New structures for physics*, pages

3–94. Springer, Berlin, Heidelberg, 2010.

[31] D.I. Spivak. *Category Theory for the Sciences.* The MIT Press, Nov 2014.

[32] D. Kartsaklis, M. Sadrzadeh, S. Pulman, and B. Coecke. Reasoning about meaning in natural language with compact closed categories and Frobenius algebras. In J. Chubb, A. Eskandarian, and V. Harizanov, editors, *Logic and Algebraic Structures in Quantum Computing and Information.* Cambridge University Press, 2013.

[33] N. Benton, J. Hughes, and E. Moggi. Monads and effects. In G. Barthe, P. Dybjer, L. Pinto, and J. Saraiva, editors, *Applied Semantics. APPSEM 2000. Lecture Notes in Computer Science, vol 2395.* Springer, Berlin, Heidelberg, 2002.

[34] R. Atkey. *Substructural Simple Type Theories for Separation and Inplace Update.* PhD thesis, Laboratory for Foundations of Computer Science, School of Informatics, University of Edinburgh, 2006.

[35] R. Atkey. Algebras for parameterised monads. In A. Kurz, M. Lenisa, and A. Tarlecki, editors, *Algebra and Coalgebra in Computer Science. CALCO 2009*, volume 5728 of *Lecture Notes in Computer Science*, pages 3–17, 2009.

[36] P.M. Martins. *Context-Oriented Functional Programming.* PhD thesis, Imperial College London, Department of Computing, 2014.

[37] C. McBride and R. Paterson. Applicative programming with effects. *Journal of Functional Programming*, 18(1):1–13, 2008.

[38] A.D. Gordon. *Functional Programming Input/Output (Distinguished Dissertations in Computer Science)*. Cambridge University Press, 1995.

[39] S. Abramsky, S. Gay, and R. Nagarajan. Specification structures and propositions-as-types for concurrency. In *Logics for Concurrency*, pages 5–40. Springer, Berlin, Heidelberg, 1996.

[40] M. Materzok and D. Biernacki. Subtyping delimited continuations. In *ICFP '11: Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*, page 81–93, Sep 2011.

[41] P. Elbourne. Weather predicates, binding, and radical contextualism. *Mind & Language*, 2019.

[42] P. Elbourne. *Situations and Individuals*. Current Studies in Linguistics. MIT press, 2005.

[43] A. Brasoveanu and J. Dotlačil. Donkey anaphora: Farmers and bishops. In L. Matthewson, C. Meier, H. Rullmann, and T.E. Zimmerman, editors, *Wiley's Linguistics Companion (Companion to Semantics)*. Wiley, 2018.

[44] P. Dekker. *Dynamic Semantics*. Studies in Linguistics and Philosophy. Springer, 2012.

[45] C. Barker and C.C. Shan. Donkey anaphora is in-scope binding. *Semantics & Pragmatics*, 1:1–46, 2008.

[46] E. Lebedeva. *Expression de la dynamique du discours a laide de continuations*. PhD thesis, l'Université de Lorraine, 2012.

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

[48] J. Barwise. *The situation in logic.* Number 17 in CSLI. Stanford University, 1989.

[49] M. Pérez-Ramírez and C. Fox. An axiomatisation of imperatives using hoare logic. In H. Bunt, I.D. Sluis, and R. Morante, editors, *Fifth International Workshop on Computational Semantics (IWCS-5)*, pages 303–320, 2003.

[50] W. Swierstra. *A Functional Specification of Effects.* PhD thesis, Nottingham University, 2009.

[51] C.A.R Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(12):576–583, 1969.

[52] W. Swierstra. A Hoare logic for the state monad. In *TPHOLs 2009: Theorem Proving in Higher Order Logics*, International Conference on Theorem Proving in Higher Order Logics, pages 440–451, 2009.

[53] D. Orchard, P. Wadler, and H. Eades. Unifying graded and parameterised monads. *MSFP@ETAPS*, pages 18–38, 2020.

[54] J. van Eijck and F.D. Vries. Dynamic interpretation and Hoare deduction. *Journal of Logic, Language, and Information*, 1(1):1–44, 1992.

[55] A. Lascarides and N. Asher. Imperatives in dialog. In P. Kühnlein, H. Riesser, and H. Zeevat, editors, *Perspectives on Dialogue in the New Millenium.* John Benjamins Publishing Company, Amsterdam/Philadelphia, 2003.

[56] H. Kamp and U. Reyle. *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic, and Discoure*

Representation Theory. Springer-Science+Business Media,B.V., 1993.

[57] L.K. Wolter. *That's That: the Semantics and Pragmatics of Demonstratives Noun Phrases.* PhD thesis, The University of California, Santa Cruz, 2006.

[58] E.P. Nowak. *Two dogmas about demonstratives.* PhD thesis, University of California, Berkeley, 2016.

[59] I. Heim. File change semantics and the familiarity theory of definiteness. In P. Portner and B. Partee, editors, *Formal Semantics: The Essential Readings.* Blackwell Publishers Ltd, 2002.

[60] A. Ranta. Grammatical framework: A type-theoretical grammar formalism. *Functional Programming*, 14(2):145–189, 2004.

[61] S. Liang, P. Hudak, and M.P. Jones. Monad transformers and modular interpreters. In *POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, San Francisco, CA, Jan 1995.

[62] G. Frege. Über sinn und bedeutung. In *Zeitschrift für Philosophie und philosophische Kritik*, pages 25–50. NF 100, 1892.

[63] C.C. Shan. A variable-free dynamic semantics. In Robert van Rooy and Martin Stokhof, editors, *Proceedings of the 13th Amsterdam colloquium*, pages 204–209, 2001.

[64] O. Kiselyov, A. Sabry, and C. Swords. Extensible effects: an alternative to monad transformers. *Haskell '13: Proceedings of the 2013 ACM SIGPLAN symposium on Haskell*, pages 59–70, Sep 2013.

[65] A. Brasoveanu. *Structured Nominal and Modal Reference.* PhD thesis,

the Graduate School-New Brunswick

Rutgers, The State University of New Jersey, 2007.

[66] L. Joachim. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958.

[67] D.J. Dougherty. Closed categories and categorial grammar. *Notre Dame Journal of Formal Logic*, 34(1), 1993.

[68] C. Fox. The meaning of formal semantics. In P Stalmaszczyk, editor, *Semantics and Beyond Philosophical and Linguistic Inquiries*, pages 85– 108. De Gruyter, 2014.

[69] R. Moot and C. Retoré. Natural language semantics and computability. *Journal of Logic, Language and Information*, 28(2):287–307, 2019.

[70] G. Giorgolo and C. Unger. Coreference without discourse referents a non-representational drt-like discourse semantics. In Erik Tjong Kim Sang Barbara Plank and Tim Van de Cruys, editors, *19th Meeting of Computational Linguistics in the Netherlands*, 2009.

[71] P. Elbourne. *Definite Descriptions*. Oxford Studies in Semantics and Pragmatics. Oxford University Press, 2013.

[72] J. Stanley and Z.G. Szabó. On quantifier domain restriction. *Mind & Language*, 15(2-3):219–261, 2000.

[73] F. Recanati. Indexicality and context-shift. *Workshop on Indexicals, Speech Acts and Logophors*, Nov 2004.

[74] A. Filinski. Representing layered monads. In *26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL*

'99). ACM, pages 175–188. ACM, Jan 1999.

[75] C.C. Shan. Linguistic side effects. In C. Barker and P. Jacobson, editors, *Direct compositionality*, pages 132–163. Oxford University Press, 2007.

[76] M. Jary and M. Kissine. *Imperatives.* Key topics in Semantics and Pragmatics. Cambridge University Press, 2014.

[77] M. Kaufmann. *Interpreting Imperatives.* Studies in Linguistics and Philosophy. Springer, 2011.

[78] C. Potts. *The Logic of Conventional Implicatures.* Oxford, Oxford university press, 2005.

[79] C. Barker, R. Bernardi, and C.C. Shan. Principles of interdimensional meaning interaction. *Proceedings of SALT 20*, pages 109–127, 2010.

[80] S. Anderbois, A. Brasoveanu, and R. Henderson. At-issue proposals and appositive impositions in discourse. *Journal of Semantics*, 32(1):93–138, 2015.

[81] P. Wadler. Propositions as types. *Communications of the ACM*, 58(12):75–84, Dec 2015.

[82] K. Mineshima. A presuppositional analysis of definite descriptions in proof theory. In *Annual Conference of the Japanese Society for Artificial Intelligence*, pages 214–227. Springer, Berlin, Heidelber, Jun 2007.

[83] J. Grudzi´ska and M. Zawadowski. Inverse linking, possessive weak definites and haddock descriptions: A unified dependent type account. *Journal of Logic, Language and Information*, 28:239–260, 2019.

[84] R. Moot and C. Retoré. *The Logic of Categorial Grammars: A Deductive Account of Natural Language Syntax and Semantics.* Springer, 2012.

[85] D. Grune and C.J.H. Jacobs. *Parsing Technique: A Practical Guide.* Springer, 2008.

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

[87] D.R. Dowty, L. Karttunen, and A.M. Zwicky. *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives.* Cambridge University Press, 2005.

[88] B.C. Pierce. *Types and Programming Languages.* The MIT Press, 2002.

[89] E. Moggi. An abstract view of programming languages, 1989.

[90] R. Bernardi. Scope ambiguities through the mirror. In M.B.H. Everaert, T. Lentz, H.N.M.D. Mulder, ∅. Nilsen, and A. Zondervan, editors, *The Linguistics Enterprise: From knowledge of language to knowledge in linguistics*, pages 11–54. John Benjamins publishing company, 2010.

[91] J. Barnes, editor. *Complete Works of Aristotle: Volume 1 The Revised Oxford Translation.* Bollingen. The Princeton University Press, 1984.

[92] B. Russell. *Principles of Mathematics.* New York: Norton, 1903.

[93] P. Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[94] S. Feferman. Typical ambiguity: trying to have your cake and eat it

too. In *One hundred years of Russell's paradox*, pages 135–151. De Gruyter, 2004.

[95] M. Ganesalingam. *The Language of Mathematics: A Linguistic and Philosophical Investigation.* Springer-Verlag Berlin Heidelberg, 2013.

[96] C. Barker. Continuations: in-situ quantifcation without storage or type shifting. In R. Hastings, B. Jackson, and Z. Zvolensky, editors, *SALT Semantics and Linguistic Theory*, volume XI. Cornell university press, 2001.

[97] K. Kearns. *Semantics.* Palgrave Macmillan, 2011.

[98] N. Chomsky. *Aspects of the theory of syntax.* Cambridge, Massachusetts: MIT Press, 1965.

[99] P. Martin-Löf. *Intuitionistic Type Theory.* Sambin, Giovanni, Napoli: Bibliopolis, 1984.

[100] B. Partee, editor. *Compositionality in Formal Semantics.* Blackwell Publishing, 2004.

[101] P.D. Groote. Towards a montagovian account of dynamics. In *SALT Semantics and Linguistic Theory*, volume XVI, pages 1–16. LSA Linguistic Society of America, Jan 2006.

[102] C. Fox and S. Lappin. *Foundations of Intensional Semantics.* John Wiley & Sons, 2008.

[103] C. Fox. *The Ontology of Language Properties, Individuals and Discourse.* Center for the Study of Language and Information Publications., 2000.

[104] H. Barendregt. *Lambda Calculi with Types*, volume II of *Handbook of Logic in Computer Science*. Oxford University Press, 1993.

[105] R. Carnap. *Meaning and Necessity: a Study in Semantics and Modal Logic*. The university of Chicago Press, 1947.

[106] S. Soames. *Philosophy of Language*. Princeton university press, 2010.

[107] M.J. Cresswell. *Semantical Essays: Possible Worlds and Their Rivals*. SPRINGER-SCIENCE+BUSINESS MEDIA, B.V., 1988.

[108] J. Barwise and R. Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4(2):159–219, 1981.

[109] J. van Benthem. *Logical Dynamics of Information and Interaction*. Cambridge University Press, 2010.

[110] M. Aloni, A. Buler, and P. Dekker, editors. *Questions in Dynamic Semantics*. Current research in Semantics Pragmatics Interface. Elsevier, 2007.

[111] I. Heim. *The Semantics of Definite and Indefinite Noun Phrases*. PhD thesis, University of Massachusetts, 1982.

[112] S. Chatzikyriakidis and Z. Luo, editors. *Modern Perspectives in Type Theoretical Semantics*. Studies in Linguistics and Philosophy. Springer, 2017.

[113] G.V. Morill. *Type logical grammar, categorial logic of signs*. Kluwer Academic Publishers, 1994.

[114] A. Ranta. Syntactic calculus with dependent types. *Journal of Logic, Language, and Information*, 7(4):413–431, Oct 1998.

[115] H. Poincaré. *Mathematics & Science Last Essays Paperback – 1913*. Dover Publications, 2010.

[116] J. van Benthem and A. ter Meulen, editors. *Handbook of Logic & Language*. North Holland, 1997.

[117] M.H.V.D. Berg. *Some aspects of the internal structure of discourse. The dynamics of nominal anaphora*. PhD thesis, Universiteit van Amsterdam, 1996.

[118] J. Ginzburg. *The interactive Stance: meaning for conversation*. Oxford, 2012.

[119] W. Hodges. *A shorter model theory*. Cambridge University Press, 1997.

[120] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics (Studies in Logic and the Foundations of Mathematics 103)*. 2nd edition, Amsterdam: North-Holland, 1985.

[121] U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, 2007.

[122] D. Bekki. Monad and meta-lambda calculus. In R. Hastings, B. Jackson, and Z. Zvolensky, editors, *SALT XI Semantics and Linguistic Theory*, volume LNAI 5447 of *New Frontiers in Artificial Intelligence*, pages 193–208, 2009. JSAI 2008, LENLS5.

[123] A. Nanevski, F. Pfenning, and B. Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic (TOCL)*, 9(3), 2008.

[124] T. Altenkirch and B. Reus. Monadic presentations of lambda terms using generalized inductive types. In *International Workshop on*

Computer Science Logic, pages 453–468. Springer, Berlin, Heidelberg, September 1999.

[125] R. Muskens. Language, lambdas, and logic. In *Resource-sensitivity, binding and anaphora*, pages 23–54. Springer, Dordrecht, 2003.

[126] A. Chlipala. *Certified programming with dependent types.* MIT press, 2008.

[127] J.A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, January 1965.

[128] R. Milner. A theory of type polymorphism in programming. *Computer and System Sciences*, 17(3):348–374, 1978.

[129] R. Constable et al. *Implementing Mathematics with The Nuprl Proof Development System.* Englewood Cliffs, NJ: Prentice-Hall, 1986.

[130] P. Dybjer. Internal type theory. In *Types for Proofs and Programs, Lecture Notes in Computer Science*, volume 1158, pages 120–134, 1996.

[131] P. Aczel and N. Gambino. Collection principles in dependent type theory. In *TYPES '00 Selected papers from the International Workshop on Types for Proofs and Programs*, pages 1–23, 2000.

[132] Z. Luo. *Computation and reasoning: A type theory for computer science.* Clarendon press, Oxford, 1994.

[133] A. Nanevski, G. Morrisett, and L. Birkedal. Hoare type theory, polymorphism and separation. *Functional Programming*, 18(5-6):865–911, Sep 2008.

[134] The Univalent Foundations Program. *Homotopy Type Theory: Uni-*

valent Foundations of Mathematics. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

[135] R.T. Oehrle. Multi-modal type-logical grammar. In R. Dorsley and K. Börjars, editors, *Non-transformation syntax: formal and explicit models of grammar*. Wiley, Blackwell, 2011.

[136] R. Cooper. Type-theoretical approaches to lexical semantics. *Journal of Language Modelling*, 5(2), 2017.

[137] R. Cooper. Records and record types in semantic theory. *Journal of Logic and Computation*, 15(2):99–112, Apr 2005.

[138] D. Bekki. Representing anaphora with dependent types. In *LACL 2014: Logical Aspects of Computational Linguistics*, International Conference on Logical Aspects of Computational Linguistics, pages 14–29, 2014.

[139] C. Fox and R. Turner. In defense of axiomatic semantics. In P Stalmaszczyk, editor, *Philosophical and Formal Approaches to Linguistic Analysis*. Ontos Verlag, 2012.

[140] Y. Bar-Hillel. Indexical expressions. *Mind*, LXIII:359–379, Jul 1954.

[141] P. Boldini. Formalizing context in intuitionistic type theory. *Fundamenta Informaticae*, 42(2):105–127, May 2000.

[142] J. Carlström. Interpreting descriptions in intensional type theory. *The Journal of Symbolic Logic*, 70(2), Jun 2005.

[143] K. Mineshima. A presuppositional analysis of definite descriptions in proof theory. In *New Frontiers in Artificial Intelligence, JSAI 2007 Conference andWorkshops, Miyazaki, Japan*, pages 18–22, Jun 2007.

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

[145] P. Portner. The semantics of imperatives within a theory of clause types. In R Young, editor, *SALT XIV*, pages 235–252, Ithaca, NY: Cornell University, 2004.

[146] Z. Luo. Coercive subtyping. *Logic and computation*, 9(1), 1999.

[147] M.P. Jones, P. Hudak, and S. Shaumyan. Using types to parse natural language. *Proceedings of the 1995 Glasgow Workshop on Functional Programming*, pages 1–11, Jul 1995.

[148] G. Morill and O. Valentín. Spurious ambiguity and focalization. *Computational Linguistics*, 44(2):285–327, 2018.

[149] P.D. Groote. Type raising, continuations, and classical logic. *Thirteenth Amsterdam Colloquium*, pages 97–101, 2001.

[150] D. Westerståhl. Compositionality and ambiguity. *Philosophical Communications, web series no. 46*, 2007.

[151] J. McCarthy. Notes on formalizing context. In *IJCAI'93 Proceedings of the 13th international joint conference on Artifical intelligence*, volume 1, pages 555–560, 1995.

[152] J. Hinkita. *the Principles of Mathematics Revisited.* Cambridge University Press, 1996.

[153] J. Väänänen. *Dependence Logic: a New Approach to Independence Friendly Logic.* London Mathematical Society Student Texts. Cam-

bridge University Press, 2007.

[154] P.R Sutton. *Vagueness, communication, and semantic information.* PhD thesis, King's College London, 2013.

[155] A. Abel and J.P. Bernardy. A unified view of modalities in type systems. *Proc. ACM Program. Lang, ICFP*, 2020.

[156] M.V. Aldridge. *The elements of mathematical semantics.* De Gruyter, 1992.

[157] B. Lawvere. Adjointness in foundations. *Dialectica*, (23):281–296, 1969.

[158] A. Szabolsci. *Quantification.* Cambridge University Press, 2010.

[159] S. Lappin and C. Fox. *The handbook of contemporary semantic theory.* John Wiley & Sons, 2015.

[160] J. van Benthem. Questions about quantifiers. *Journal of Symbolic Logic*, 49(2), 1984.

[161] J. van Benthem. Polyadic quantifiers. *Linguistics and Philosophy*, 12:437–464, 1989.

[162] P. Dekker. A guide to dynamic semantics. 2008.

[163] C. Roberts. Demonstratives as definites. In K.Van Deemter and K. Kibble, editors, *Information Sharing: Reference and Presupposition in Language Generation and Interpretation*, pages 89–196. CSLI Press, 2002.

[164] S. Qian, P.D. Groote, and M. Amblard. Modal Subordination in Type Theoretic Dynamic Logic. *Linguistic Issues in Language Technology*, 14((1)):1–39, 2016.

[165] H. Hendriks. *Studied Flexibility: Categories and Types in Syntax and Semantics.* PhD thesis, University of Amsterdam, 1993.

[166] C. Barker. Continuations and the nature of quantification. *Natural Language Semantics*, 10(3):211–242, Sep 2002.

[167] A. Oliver and T. Smiley. *Plural Logic.* Oxford University Press, 2012.

[168] G.R. Berta. *From plurals to superplurals: in defence of higher-level plural logic.* PhD thesis, University of Glasgow, 2018.

[169] P. Amaral, C. Roberts, and E.A Smith. Review of the logic of conventional implicatures by Chris Potts. *Linguistics and Philosophy*, 30(6):707–749, 2007.

[170] T. Xue and Z. Luo. Dot-types and their implementation. In *LACL'12, LNCS 7351*, 2012.

[171] H. Bahramian, N. Nematollahi, and A. Sabry. Copredication in homotopy type theory: A homotopical approach to formal semantics of natural languages. *Fourth Workshop on Natural Language and Computer Science, NLCS*, 2016.

[172] J. Putejovsky. *The Generative Lexicon.* MIT Press, 1995.

[173] J. Putejovsky. A survey of dot objects, manuscript. 2005.

[174] P. Dybjer. Category theory and programming language semantics: an overview. *Proceedings of a tutorial and workshop on Category theory and computer programming*, pages 165–181, Nov 1986.

[175] R.A.G. Seely. Locally cartesian closed categories and type theory. *Math. Proc. Camb. Phil. Soc*, 95(33), 1984.

[176] M. Hofmann. On the interpretation of type theory in locally cartesian closed categories. In *Proceedings of Computer Science Logic, Lecture Notes in Computer Science*, pages 427–441. Springer, 1994.

[177] P. Fu, K. Kishida, and P. Selinger. Linear dependent type theory for quantum programming languages: Extended abstract. *LICS '20: Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2020.

[178] M. Vákár. A categorical semantics for linear logical frameworks. *FoSSaCS 2015: International Conference on Foundations of Software Science and Computation Structures*, pages 102–116, 2015.

[179] A. Brasoveanu. Donkey pluralities: plural information states versus non-atomic individuals. *Linguistics and Philosophy*, 31(2):129–209, Apr 2008.

[180] S. Kripke. *Naming and necessity.* Harvard University Press, 1972.

[181] P. Galliani. *The dynamics of imperfect information.* PhD thesis, University of Amsterdam, 2012.

[182] A. Asperti and G. Longo. *Categories, Types, and Structures.* MIT Press, 1991.

[183] J.C. Mitchell and E. Moggi. Kripke-style models for typed lambda calculus. *Annals of Pure and Applied Logic*, 51:99–124, 1991.

[184] E. Moggi. Computational lambda calculus and monads, 1988.

[185] R. Bird. *Introduction to Functional Programming using Haskell.* Prentice Hall, 1998.

[186] P. Walder. Monads for functional programming. In J. Jeuring and E. Meijer, editors, *Advanced Functional Programming, Proceedings of the Bastad Spring School*, Lecture Notes in Computer Science 925. Springer Verlag, May 1995.

[187] S.P. Jones. Tackling the awkward squad monadic input/output, concurrency, execptions and foreign-language calls. *Lecture Notes for a tutorial given at Mark-toberdorf Summer School*, 2002.

[188] N. Benton. Categorical monads and computer programming. *Impact150 - LMS 150 Year Impact Stories. London Mathematical Society*, Nov 2015.

[189] T. Petricek. What we talk about when we talk about monads. *The Art, Science, and Engineering of Programming*, 2(3), 2018.

[190] A.A Ivanova, S. Srikant, Y. Sueoka, H.H Kean, R. Dhamala, U.M. O'reilly, M.U. Bers, and E. Fedorenko. Comprehension of computer code relies primarily on domain-general executive resources. *BioRxiv*, 2020.

[191] P. Wadler. The essence of functional programming. In *Proceedings of the 19th ACM Symposium on Principles of Programming Languages.* ACM Press, 1992.

[192] S.P. Jones and P. Wadler. Imperative functional programming. *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 71–84, Mar 1993.

[193] A.D. Gordon and K. Hammond. Monadic i/o in haskell 1.3. In *Pro-*

ceedings of the Haskell Workshop, Jun 1995.

[194] G. Giorgolo and A. Asudeh. Monads as a solution for generalized opacity. In *EACL 2014 Workshop on Type Theory and Natural Language Semantics (TTNLS)*, pages 19–27, Apr 2014.

[195] A. Ścibior, Z. Ghahramani, and A.D Gordon. Practical probabilistic programming with monads. *In Proceedings of the 2015 ACM SIGPLAN Symposium on Haskell*, pages 165–176, Aug 2015.

[196] N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 154–165, Jan 2002.

[197] M. Giry. A categorical approach to probability theory. *Categorical aspects of topology and analysis, Proc. int. Conf., Ottawa 1981, Lect. Notes Math*, 915:68–85, 1982.

[198] R. O'Connor. A monadic, functional implementation of real numbers. *Mathematical Structures in Computer Science*, 1(17):129–159, 2007.

[199] D. Ahman and T. Uustalu. Update monads: cointerpreting directed containers. *In Proc. of 19th Int. Conf. on Types for Proofs and Programs, TYPES*, 13:1–23, Jul 2014.

[200] S. Mac Lane. *categories for the working mathematician.* Springer, 1997.

[201] A. Pitts. Categorical logic. In *Handbook of Logic in Computer Science*, volume 5, pages 39–128. Oxford University Press, 2001.

[202] S. Liang, P. Hudak, and M.P. Jones. Monad transformers and modular

interpreters. In *POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, San Francisco, CA, Jan 1995.

[203] D. Kaplan. Demonstratives: an essay on the semantics, logic, metaphysics and epistemology of demonstratives and other indexicals. In *Summer Institute in the Philosophy of Language*, 1977. Los Angeles, s.n.

[204] G. Link. The logical analysis of plurals and mass terms: a lattice-theoretical approach. In P. Portner and B. Partee, editors, *Formal Semantics: the Essential Readings.* Wiley Press, Jan 2008.

[205] J. Grudzi´ska and M. Zawadowski. Scope ambiguities, monads and strengths. *Journal of Language Modelling*, 5(2):179–227, 2017.

[206] G. Giorgolo and A. Asudeh. One semiring to rule them all. *Cognitive Science*, 2014.

[207] C.L. Hamblin. Questions in Montague English. *Foundations of Language*, 10(1):41–53, 1973.

[208] M. Rooth. *Association with Focus.* PhD thesis, University of Massachusetts, Amherst, 1985.

[209] C.C. Shan and C. Barker. Explaining crossover and superiority as left-to-right evaluation. *Linguistics and Philosophy*, 29(1):91–134, 2006.

[210] J.P. Bernardy, S. Chatzikyriakidis, and A. Maskharashvili. A computational treatment of anaphora and its algorithmic implementation. *Logic, Language and Information*, 2020.

[211] D. Bekki and M. Masuko. Meta-lambda calculus and linguistic monads.

In E. McCready, K. Yabushita, and K. Yoshimoto, editors, *Formal Approaches to Semantics and PragmaticsJapanese and Beyond*, Studies in Linguistics and Philosophy 95, pages 31–64. Springer, 2014.

[212] D. Bekki and K. Asai. Representing covert movements by delimited continuations. In K. Nakakoji, Y. Murakami, and E. McCready, editors, *New Frontiers in Artificial Intelligence (JSAI-isAI Workshops, Selected Papers from LENLS6*, pages 161–180. Springer, Heidelberg, Nov 2010.

[213] J. Maršík. *Effects and Handlers in Natural Language.* PhD thesis, The university of de Lorraine, 2016.

[214] M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. *In International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 190–201, Jul 1992.

[215] J.C. Reynolds. The discoveries of continuations. *LISP AND SYMBOLIC COMPUTATION: An InternationM Journal*, 6:233–248, 1993.

[216] B. Partee. Noun phrase interpretation and type shifting principles. In *Formal semantics: The essential readings*, pages 357–381. Wiley, 2002.

[217] O. Danvy and A. Filinski. A functional abstraction of typed contexts. Technical report, Computer Science Department, University of Copenhagen., 1989.

[218] A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Lisp and symbolic computation 6, no. 3-4*, pages 289–360, 1993.

[219] D. Biernacki. *The Theory and Practice of Programming Languages*

with Delimited Continuations. PhD thesis, Department of Computer Science, University of Aarhus, 2005.

[220] N. Leslie. *Continuations and Martin-Löf's type theory*. PhD thesis, Massey University,Albany,New Zealand, 2000.

[221] G.D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(2):125–159, Dec 1975.

[222] R. Clark. Number sense and quantifier interpretation. *Topoi*, 26(1):51–62, 3 2007.

[223] O. Kiselyov. Parameterized extensible effects and session types. In *Proceedings of the 1st International Workshop on Type-Driven Development*, pages 41–42, 2016.

[224] G. Giorgolo and A. Asudeh. Monad for conventional implicatures. In *Sinn und Bedeutung*, volume 16, 2012.

[225] O. Kiselyov and H. Ishii. Freer monads, more extensible effects. *ACM SIGPLAN Notices*, 50(12):94–105, 2015.

[226] C. Reuben. Monad transformers for natural language: Combining monads to model effect interaction.

[227] L. Burke. P-hype: A monadic situation semantics for hyperintensional side effects. *Proceedings of Sinn und Bedeutung 23*, 23(1), 2019.

[228] C.S Leong and M.Y. Erlewine. Long-distance dependencies in continuation grammar. In *Proceedings of the 33rd Pacific Asia Conference on Language, Information and Computation (ACL Anthology).*, 2019.

[229] D. Ahman and T. Uustalu. Update monads: Cointerpreting directed

containers. In R. Matthes and A. Schubert, editors, *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, LIPIcs–Leibniz International Proceedings in Informatics, pages 1–23, 2013.

[230] J. van Eijck. Incremental dynamics. *Journal of Logic, Language and Information*, 10(3):319–351, 2001.

[231] O. Kiselyov, R. Lämmel, and K. Schupke. Strongly typed heterogeneous collections. *Haskell '04: Proceedings of the 2004 ACM SIGPLAN workshop on Haskell*, Sep 2004.

[232] J. Gibbons, F. Henglein, R. Hinze, and N. Wu. Relational algebra by way of adjunctions. *Proceedings of the ACM on Programming Languages, 2(ICFP)*, pages 1–28, 2018.

[233] L. Champollion. Covert distributivity in algebraic event semantics. *Semantics and Pragmatics*, 9(15):1–65, 2016.

[234] L. Champollion. Overt distributivity in algebraic event semantics. *Semantics and Pragmatics*, 9(16):1–65, 2016.

[235] L. Champollion, J. Bledin, and H. Li. Rigid and flexible quantification in plural predicate logic. In *SALT 27*, pages 418–437, 2017.

[236] J. Bracker and H. Nilsson. Supermonads and superapplicatives. *Functional Programming*, 103, 2018.

[237] A. Nanevski, G. Morrisett, and M. Birkedal. Hoare type theory, polymorphism and separation. *Journal of functional programming*, 8(5-6), Sep 2008.

[238] A. Nanevski, A. Banerjee, G.A Delbianco, and I. Fábregas. Specifying concurrent programs in separation logic: morphisms and simulations. In *Proceedings of the ACM on Programming Languages, 3(OOPSLA)*, pages 1–30, 2019.

[239] C. McBride. Kleisli arrows of outrageous fortune. *(submitted to) Functional Programming*, pages 1–24, Mar 2011.

[240] K. Maillard, D. Ahman, R. Atkey, G. Martínez, C. Hritcu, E. Rivas, and É. Tanter. Dijkstra monads for all. *Proceedings of the ACM on Programming Languages (PACMPL), 3(ICFP)*, 2019.

[241] O. Kiselyov and C.C. Shan. Lightweight monadic regions. *Proceedings of the first ACM SIGPLAN symposium on Haskell*, pages 1–12, Sep 2008.

[242] O. Kammar. *An Algebraic Theory of Type-and-Effect Systems*. PhD thesis, University of Edinburgh, Oct 2014.

[243] J. Bracker. *Unified notions of generalised monads and applicative functors*. PhD thesis, University of Nottingham, 2018.

[244] P. O'hearn. Separation logic. *Communications of the ACM*, 62(2), Feb 2019.

[245] R.K. Dybvig, S.P. Jones, and A. Sabry. A monadic framework for delimited continuations. *Functional Programming*, 17(6):687–730, 2007.

[246] R. Milner. *Communicating and Mobile Systems: The π Calculus*. Information and Computation. Cambridge University Press, 1999. ISBN 10: 0521658691 / ISBN 13: 9780521658690.

[247] S.J. Gay and V.T. Vasconcelos. Linear type theory for asynchronous session types. *Functional Programming*, 20(1):19–50, 2010.

[248] M.J. Jaskelioff. *Lifting of operations in modular monadic semantics.* PhD thesis, University of Nottingham, 2009.

[249] W. Swierstra and T. Baanen. A predicate transformer semantics for effects (functional pearl). *Proceedings of the ACM on Programming Languages, 3(ICFP)*, pages 1–26, 2019.

[250] P.B. Levy, J. Power, and H. Thielecke. Modelling environments in call-by-value programming languages. *Information and Computation*, 185(2):182–210, 2003.

[251] A. Asudeh and R. Crouch. Coordination and parallelism in glue semantics: Integrating discourse cohesion and the element constraint. In *proceeding of the LFG02 conference.* CSLI Publications, 2002.

[252] R.V. Guha. *Contexts : A Formalization and Some Applications.* PhD thesis, Standord University, Feb 1995.

[253] J. Malakhovski. *On the Expressive Power of Indexed Applicative and Monadic Structures.* PhD thesis, IRIT, University of Toulouse-3 Paul Sabatier and Saint Petersburg National Research University of Information Technologies, Mechanics and Optics, 2019.

[254] M. Sackman and S. Eisenbachs. Safely Speaking in Tongues Statically Checking Domain Specific Languages in Haskell. In J.J. Vinju and T. Ekman, editors, *Proceedings of The Ninth Workshop on Language Descriptions, Tools, and Applications (LDTA 2009).* Elsevier, 2009.

[255] D. Janin. A timed io monad. In *International Symposium on Practical Aspects of Declarative Languages*, pages 131–147. Springer, Cham., Jan 2020.

[256] A. Filinski. *Controlling Effects.* PhD thesis, Carnegie Mellon University, May 1996.

[257] S. Abramsky and J. Väänänen. From IF to BI. *Synthese*, 167(2):207–230, 2009.

[258] Z. Luo. Dependent record types revisited. In *Proceedings of the 1st Workshop on Modules and Libraries for Proof Assistants*, pages 30–37, 2009.

[259] R. Cooper and J. Ginzburg. Type theory with records for natural language semantics. In S. Lappin and C. Fox, editors, *The Handbook of Contemporary Semantic Theory.* John Wiley & Sons, Ltd, 2015.

[260] P. Curien, R. Garne, and M. Hofmann. Revisiting the categorical interpretation of dependent type theory. *Theoretical Computer Science*, 546(21):99–119, Aug 2014.

[261] B. Jacobs. *Categorical Logic and Type Theory.* Elsevier, 1999.

[262] A. Abel and C. Sattler. Normalization by evaluation for call-by-push-value and polarized lambda-calculus. In *21st International Symposium on Principles and Practice of Declarative Programming, PPDP'19*, 2019.

[263] M.A. Warren. *Homotopy Theoretic Aspects of Constructive Type Theory.* PhD thesis, Carnegie Mellon University, Aug 2008.

[264] M. Makkai. First order logic with dependent sorts, with applications to category theory. 1995. Preprint 1995, version November 6. 201 pp. Available from Makkai's webpages.

[265] T. Sider. *Logic for Philosophy.* Oxford University Press, 2009.

[266] N. Gerasimov and E. Pyshkin. Using dynamic predicate logic for pronominal anaphora resolution in russian texts. In *International Workshop on Applications in Information Technology.* The University of Aizu Press, 2015.

[267] J. Carlström. *Partiality and Choice: Foundational Contributions.* PhD thesis, Stockholm University, Faculty of Science, Department of Mathematics, 2005.

[268] G.D. Plotkin and J. Power. Notions of computation determine monads. In *FoSSaCS 2002: Foundations of Software Science and Computation Structures*, International Conference on Foundations of Software Science and Computation Structures, pages 343–356, Mar 2002.

[269] M.G.H. Gotham. *Copredication, Quantification and Individual.* PhD thesis, University College London, Jan 2015.

[270] N. Asher. *Lexical Meaning in Context: a Web of Words.* Cambridge university press, 2011.

[271] N. Asher. Context in content composition. In R. Kempson, T. Fernando, and N. Asher, editors, *Philosophy of Linguistics*, volume 14. North Holland, 2012.

[272] L. Champollion. *Parts of a whole: Distributivity as a bridge between*

aspect and measurement, volume 66 of *Oxford Studies in Theoretical Linguistics*. Oxford University Press, 2017.

[273] N. Ivlieva. *Scalar Implicatures and the Grammar of Plurality and Disjunction*. PhD thesis, MIT, 2013.

[274] A. Radulescu. The logic of indexicals. *Synthese*, 192(6):1839–1860, Jun 2015.

[275] J. Groenendijk and M. Stokhof. Dynamic montague grammar. In *Papers from the Second Symposium on Logic and Language*, pages 3–48. Akademiai Kiadoo, 1989.

[276] A.R. McKenzie. *The Role of Contextual Restriction in Reference Tracking*. PhD thesis, University of Massachusetts Amherst, May 2012.

[277] L. Stirling. *Switch Reference and Discourse Representation*. Number 63 in Cambridge studies in linguistics. Cambridge University Press, 1993.

[278] D. Hardt. Dynamic interpretation of verb phrase ellipsis. *Linguistics and philosophy*, pages 185–219, 1999.

[279] L. Champollion. Homogeneity in donkey sentences. In Ken Turner and Klaus von Heusinger, editors, *Proceedings of SALT XXVI*, page 684–704, 2016.

[280] S. Charlow. Cross categorial donkeys. In *Selected papers from the 18th Amsterdam Colloquium*, LNCS 7218, pages 261–270, 2012.

[281] R. Muskens. Tense and the logic of change. In U. Egli, P.E. Pause, C. Schwarze, A.V. Stechow, and G. Wienold, editors, *Lexical Knowledge in the Organization of Language*, pages 147–183. Benjamins, Am-

sterdam, 1995.

[282] M.J Gordon. Mechanizing programming logics in higher order logic. *Current trends in hardware verification and automated theorem proving*, pages 387–439, 1989.

[283] S. Chatzikyriakis, F. Pasquali, and C. Retore, editors. *Ifcolog Journal of Logics and their Applications. Hilbert's epsilon and tau in Logic, Informatics and Linguistics.* Volume 4, Number 2. College Publications, Mar 2017.

[284] R. Atkey and P. Johann. Interleaving data and effects. *Functional Programming*, 25, 2015. Cambridge University Press.

[285] J.C.L. Ralha. *A multidimeonsional dynamic framework for handling simple interruption phenomena, anaphoric pronouns and definite descriptions.* PhD thesis, The University of Leeds, 1998.

[286] P. Galliani. Inclusion and exclusion dependencies in team semantics - on some logics of imperfect information. *Annals of Pure and Applied Logic*, 163(1):68–84, Jan 2012.

[287] M. Kracht. Dynamic semantics. *Linguistische Berichte*, pages 217–241, 2002.

[288] R.W.F. Nouwen. *Plural Pronominal Anaphora in Context: Dynamic Aspects of Quantification.* PhD thesis, The Utrecht University, 2003.

[289] E. Manes. Monads of sets. In M. Hazewinkel, editor, *Handbook of Algebra*, volume 3, pages 67 – 153. North-Holland, 2003.

[290] P. Geach. *Reference and Generality: An Examination of Some Me-*

dieval and Modern Theories. Ithaca, New York: Cornell University Press, 1962.

[291] J. Dotlačil and F. Roelofsen. Dynamic inquisitive semantics: anaphora and questions. In *Proceedings of Sinn und Bedeutung 23*, 2019.

[292] P. Elbourne. Demonstratives as individual concepts. *Linguistics and Philosophy*, 31(4):409–466, Aug 2008.

[293] O. Kiselyov and C.C. Shan. Continuation hierarchy and quantifier scope. In EMcCready, K Yabushita, and K Yoshimoto, editors, *Formal Approaches to Semantics and Pragmatics*, Studies in Linguistics and Philosophy. Springer, Dordrecht, 2014.

[294] P. Dekker. Predicate logic with anaphora. *Semantics and Linguistic Theory*, 4:79–95, Nov 1994.

[295] A. Heyting. Die intuitionistische grundlegung der mathematik. *Erkenntnis*, 2:106–115, 1931.

[296] L.E.J. Brouwer. Points and spaces. *Canadian Journal of Mathematics*, 6:1–17, 1954.

[297] A. Kolmogoroff. Zur deutung der intuitionistischen logik. In K Knoff, E Schmidt, and I Schur, editors, *Mathematische Zeitschrift*, volume 35, pages 58–65. Verlag Von Julius Springer, 1932.

[298] R. Mitkov. *Anaphora resolution.* Longman, 2002.

[299] A. Abel, T. Coquand, and P. Dybjer. Normalization by evaluation for Martin-Löf type theory with typed equality judgements. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007,*

pages 3–12. IEEE, Jul 2007.

[300] M. Satoh and D. Bekki. Calculating projections via type checking. In *the Proceedings of TYpe Theory and LExical Semantics (TYTLES) in the 27th European Summer School in Logic, Language and Information (ESSLLI 2015)*, 2015.

[301] P. Melliés. The parametric continuation monad. *Mathematical Structures in Computer Science*, 27(5), 2017.

[302] P.D. Groote. Towards a Montagovian account of dynamics. *Semantics and Linguistic Theory*, 16:1–16, Aug 2006.

[303] K.Von Fintel. *Restriction on Quantifiers Domains.* PhD thesis, University of Massachusetts, May 1994.

[304] G.M. Kobele. The Cooper storage idiom. *logic, Language and Informatics*, 27(2), 2018.

[305] P. Taylor. *Practical Foundations of Mathematics.* Number 59 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1999.

[306] A. Pitts. Nominal logic: first order theory of names and binding. In N Kobayashi and B. C. Pierce, editors, *Fourth International Symposium on Theoretical Aspects of Computer Software (TACS2001), LNCS Vol. 2215*, pages 219–242, 2001.

[307] J. Cheney. A dependent nominal type theory. *Logical Methods in Computer Science*, 8(1):1–29, 2012.

[308] I. Ciardelli, J. Groenendijk, and F. Roelofsen. *Inquisitive Semantics.*

Oxford Surveys in Semantics and Pragmatics. Oxford University Press, 2019.

[309] W. Dubislav. Zur unbegründbarkeit der forderungssätze. *Theoria*, 3:330–342, 1937.

[310] J. Jörgensen. Imperatives and logic. *Erkenntnis*, 7:288–296, 1937.

[311] A. Ross. Imperative and logic. *Philosophy of Science*, 11(1):30–46, Jan 1944.

[312] C.L. Hamblin. *Imperatives.* Basil Blackwell, 1987.

[313] K. Segerberg. Validity and satisfaction in imperative logic. *Notre Dame Journal of Formal Logic*, 31(2), 1990.

[314] H. Clark-Younger. *Imperatives and Logical Consequence.* PhD thesis, The University of Otago, 2014.

[315] H. Clark-Younger. Imperatives and the more generalised Tarski thesis. *Thought: A Journal of Philosophy*, 3(4):314–320, 2014.

[316] P.B.M. Vranas. New foundations for imperative logic i: Logical connectives, consistency, and quantifiers. *Noûs*, 42(4):529–572, 2008.

[317] P.B.M. Vranas. New foundations for imperative logic: Pure imperative inference. *Mind*, 120(478):369–446, 2011.

[318] P.B.M. Vranas. Logic of imperative. In *International Encycopedia of Ethics*. Wiley Online Library, 2015.

[319] P.B.M. Vranas. New foundations for imperative logic iii: A general definition of argument validity. *Synthese*, 193(6):1703–1753, Jun 2016.

[320] P. Portner. Imperatives. In M. Aloni and P. Dekker, editors, *The*

Cambridge Handbook of Formal Semantics, pages 593–626. Cambridge University Press, 2016.

[321] C.H. Han. *The Structure and Interpretation of Imperatives:Mood and Force in Universal Grammar.* PhD thesis, University of Pennsylvania, Dec 1998.

[322] K.V Fintel and S. Iatridou. A modest proposal for the meaning of imperatives. In Ana Arregui, María Luisa Rivero, and Andrés Salanova, editors, *Modality across Syntactic Categories*, pages 288–319. Oxford Scholarship Online, 2017.

[323] S. Lauer. *Towards a Dynamic Pragmatics.* PhD thesis, Standford University, Aug 2013.

[324] J. Hansen. *Imperatives and Deontic Logic On the Semantic Foundations of Deontic Logic.* PhD thesis, Universität Leipzig, 2008.

[325] H. Poincaré. *Dernières Pensées.* Ernest Flammarion, Paris, 1913.

[326] C.H. Han. Imperatives. In K.Von. Heusinger, C. Maienborn, and P. Portner, editors, *Semantics: An International Handbook of Natural Language Meaning. Handbooks of Linguistics and Communication Science (HSK)*, pages 1785–1804. Berlin: Mouton de Gruyter, 2011.

[327] T. Kleymann. Hoare logic and auxiliary variables. *Formal Aspects of Computing*, 11(5):541–566, Dec 1999.

[328] D. Cock, G. Klein, and T. Sewell. Secure microkernels, state monads and scalable refinement. In *TPHOLs 2008: Theorem Proving in Higher Order Logics*, International Conference on Theorem Proving in Higher

Order Logics, pages 167–182, 2008.

[329] E.D. Vries and V. Koutavas. Reverse hoare logic. In *SEFM 2011: Software Engineering and Formal Methods*, International Conference on Software Engineering and Formal Methods, pages 155–171, 2011.

[330] C. Stirling. A generalization of owicki-gries's hoare logic for a concurrent while language. *Theoretical Computer Science*, 58:347–359, 1988.

[331] D. Harel. First-order dynamic logic. In Goos and Hartmanis, editors, *Lecture Notes in Computer Science*, volume 68, 1979.

[332] F. Honsell and M. Miculan. A natural deduction approach to dynamic logic. In *In International Workshop on Types for Proofs and Programs*, pages 165–182, Jun 1995.

[333] B. Russell. On denoting. *Mind*, 14(56):479–493, 1905.

[334] G. Chierchia. Reference to kinds across language. *Natural language semantics*, 6(4):339–405, 1998.

[335] Hans-Martin Gärtner. Naming and economy. In O. Bonami and P. Cabredo Hofherr, editors, *Empirical Issues in Formal Syntax and Semantics*, pages 63–73. CSSP 2005, 2004.

[336] D. Kozen. Results on the propositional μ calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[337] W.P De Roever. *Recursive program schemes: Semantics and proof theory.* PhD thesis, Free University. Amsterdam, 1973.

[338] J. Bradfield and C. Stirling. Modal mu-calculi. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *The Handbook of Modal Logic*, pages

721–756. Elsevier, 2006.

[339] B. Löwe, W. Malzkorn, and T. Räsch, editors. *Foundations of the Formal Sciences II: Applications of Mathematical Logic in Philosophy and Linguistics (Vol. 17).* Springer Science & Business Media, 2003.

[340] M. Coniglio, A. Murphy, E. Schlachter, and T. Veenstra, editors. *Atypical Demonstratives: Syntax, Semantics and Pragmatics.* Linguistische Arbeiten 568. De Gruyter, 2018.

[341] D. Braun. Complex demonstratives and their singular contents. *Linguistics and Philosophy*, 31:57–99, 2008.

[342] M. Glanzberg and S. Siegel. Presupposition and policing in complex demonstratives. *Noûs*, 40(1):1–42, 2006.

[343] J. Grudzińska. Demonstrative descriptions and conventional implicatures. *Semiotica*, 188(1/4):333 – 345, 2012.

[344] E. Leopre and K. Ludwig. The semantics and pragmatics of complex demonstratives. *Mind*, 109(434), Apr 2000.

[345] J.C. King. *Complex Demonstratives: a Quantificational Account.* The MIT press, 2001.

[346] D. Braun. Structured characters and complex demonstratives. *Philosophical Studies: an International Journal for Philosophy in the Analytic Tradition*, 74(2):193–219, May 1994.

[347] E. Borg. Complex demonstratives. *philosophical Studies: an International Journal for Philosophy in the Analytic Tradition*, 97(2):229–249, Jan 2000.

[348] K. Lauri and P. Stanley. Conventional implicature. In Oh and Dinneen, editors, Syntax and semantics*, Presupposition*, volume 11, pages 1–56. New York: Academic Press, 1979.

[349] D. Kaplan. Words. *Aristotelian Society Supplementary*, 64(1):93–119, 1990.

[350] C. Roberts. Domain restriction in dynamic semantics. In E. Bach, E. Jelinek, A. Kratzer, and B. Partee, editors, *Quantification in Natural Languages*, volume 45 of *Studies in Linguistics and Philosophy.* Kluwer Academic, 1995.

[351] S. Abramsky, J. Kontinen, J. Väänänen, and H. Vollmer, editors. *Dependence Logic: Theory and Applications.* Birkhäuser, 2016.

[352] D. Wilkins. The 1999 demonstrative questionnaire: 'this' and 'that' in comparative perspective. In S.C. Levinson and N.J. Enfield, editors, *Manual for the 2001 Field Season*, pages 149–163, Nijmegen: Max Planck Institute for Psycholinguistics, 2001.

[353] A. Frigerio. Demonstratives and saliency. In C. Penco and M. Vignolo, editors, *WOC 2017 Contexts in Philosophy, Proceedings of the Workshop on Contexts in Philosophy*, 10th International Conference on Modelling and Using Contexts (CONTEXT 2017), Jun 2017.

[354] D. Roehrs. *Demonstratives and Definite Articles as Nominal Auxiliaries.* Linguistik Aktuell/Linguistics Today (LA). John Benjamins publishing company, 2009.

[355] S. Anderbois, A. Brasoveanu, and R. Henderson. Crossing the appos-

tive/ at issue meaning boundary. In N. Li and D. Lutz, editors, *SALT Semantics and Linguistics Theory*, volume 20, pages 328–346. LSA: Linguistic Society of America, 2010. Ithaca, NY, CLC publications.

[356] R. Pucella and A.J. Tov. Haskell session types with (almost) no class. *SIGPLAN Not.*, 44(2):25–36, September 2008.

[357] D. Orchard and N. Yoshida. Session types with linearity in haskell. In S. Gay and A. Ravara, editors, *Behavioural Types: from Theory to Tools*. river publisher, 2017.

[358] A. Lücking. Witness-loaded and witness-free demonstratives. In M. Coniglio, A. Murphy, E. Schlachter, and T. Veenstra, editors, *Atypical Demonstratives: Syntax, Semantics and Pragmatics*. De Gruyter, Aug 2018.

[359] H. Bliss and E. Ritter. Developing a database of personal and demonstrative pronoun paradigms: Conceptual and technical challenges. In S. Bird, P. Bunenman, and M. Liberman, editors, *Proceedings of the IRCS Workshop on Linguistic Database*, pages 38–47, 2001.

[360] M. Felleisen A. Sabry. Reasoning about programs in continuation-passing style. *LISP and Symbolic Computation*, 6:289–360, 1993.

[361] H.P. Grice. Logic and conversation. In Cole et al, editor, *Syntax and Semantics 3: Speech acts*, volume 3, pages 41–58. Elsevier, 1975.

[362] G.B. Georgi. *Demonstratives in Logic and Natural Language*. PhD thesis, University of Southern California, 2011.

[363] G.D. Nunberg. Indexicality and deixis. *Linguistics and Philosophy*,

16(1):1–43, 1993.