Lambek Calculus and Montague Grammar

Summary. This chapter discusses one of the important advantages of using (Lambek) categorial grammars: the straightforward correspondence between Lambek calculus proofs and derivations in Montague-style semantics, which extends straightforwardly to modern theories like DRT. In order to keep the exposition simple, we will only briefly discuss the intensional operators of Montague.

3.1 Introduction

The Lambek calculus is a lexicalized formalism: that is, the Lambek calculus consists of a universal set of deduction rules — the logical properties of which we have studied in detail in the previous chapter — and to obtain a Lambek grammar we only add a lexicon, a function $\text{Lex}$ which assigns a finite set of types to each word.

Now we will turn our attention to its relation to Montague semantics, introduced in [Montague, 1970a,b, 1973] which is a very important feature of categorial grammars.

We do not intend to give a lecture on Montague semantics, which is a large research topic in itself — the reader interested in this topic is referred to [Dowty et al., 1981; Gamut, 1991; Partee and Hendriks, 2011] for general introductions, and to [Carpenter, 1996] for many more applications of the Lambek calculus and Montague grammar to different semantic phenomena — but only to illustrate Montague semantics viewed from the perspective of the Lambek calculus. Montague semantics is also a controversial view of semantics. Indeed it has nothing fancy to say about mental representation or the organization of concepts as for instance in [Jackendoff, 1995] or [Pustejovsky, 1995] (though we believe that Montague grammar is at least compatible with a more comprehensive cognitive theory of meaning): the semantics of a sentence is given by formulae of (higher-order) predicate calculus, possibly of intensional logic, and the elementary expression are interpreted by logical constants: the word “Paul” is interpreted by the logical constant $\text{Paul}$ and the word “car” by the logical constant $\text{car}$ (or equivalently, as is more usual in Montague semantics, $\lambda x.e (x)$). Nevertheless it enables a neat and
computational treatment of (co)reference and of quantifiers and this is an important step towards modeling meaning computationally. Interpreting sentences in a logical syntax also allows us to model entailment: a set of sentences $s_1, \ldots, s_n$ entails another sentence $s$ if and only if the truth of all of the sentences $s_1, \ldots, s_n$ implies the truth of $s$. If all of the sentences are interpreted as logical formulae, then this semantic notion of entailment corresponds to the logical notion of entailment, that is to $s_1, \ldots, s_n \vdash s$ where ‘$\vdash$’ is the logical entailment relation of the logic we use for our interpretation.

Although Montague himself thought that one should forget about whatever lies between the sentence and its interpretation in possible world semantics, we shall choose a more intermediate level, which could be called the “syntax of semantics”, as the endpoint of our semantic interpretation: the logical formula itself. We agree with Montague and others that the intermediate steps, being unobservable, are difficult to study (or even to substantiate). The logical form is already less observable than the sentence, though tools like entailment allow us to reason about properties these logical forms must have.

### 3.2 Logic and Lambda Calculus

We will first give an extremely brief introduction to the typed lambda calculus, then see how to model logical formulae in typed lambda calculus.

#### 3.2.1 Typed Lambda Calculus and Intuitionistic Propositional Calculus

We do not pretend to include here a presentation of the typed lambda calculus, many of them exist (Krivine, 1990; Seldin and Hindley, 1980; Girard et al., 1988) but we provide a brief reminder of the minimal background necessary to follow a presentation of Montague semantics. There are at least two unrelated ways to consider the relation between the typed lambda calculus and logic:

[Church]. Typed lambda terms as logical formulae disregarding their truth and provability. Provided the type systems includes a type $e$ for entities (also called individuals) and one, $t$, for truth values (or propositions), constants for logical connectives and constants for the predicates, functions and constants, every closed formula correspond to a normal lambda term of type $t$, and conversely, every closed normal lambda term (with constants as indicated) of type $t$ corresponds to a formula (this will be made precise in next section).

---

1 We consider these phenomena part of semantics. However, in generative grammar, they are considered part of syntax. For a good introduction to semantics from the point of view of generative grammar, see (Heim and Kratzer, 1997).
2 There are, of course, many caveats to using logical entailment to model semantic entailment: semantic entailment depends, in many cases, on complex world knowledge and there has been much debate about the difference between the logical implication “$A \Rightarrow B$” and the construction “if $A$ then $B$” in natural language.