Review[3] of
Type-Logical Semantics
Author of Book: B. Carpenter
Publisher: MIT Press (585 pages)
Author of Review: Pucella and Chong, Dept of CS, Cornell Univ

4 Introduction

One of the many roles of linguistics is to address the semantics of natural languages, that is, the meaning of sentences in natural languages. An important part of the meaning of sentences can be characterized by stating the conditions that need to hold for the sentence to be true. Necessarily, this approach, called truth-conditional semantics, disregards some relevant aspects of meaning, but has been very useful in the analysis of natural languages. Structuralist views of language (the kind held by Saussure, for instance, and later Chomsky) have typically focused on phonology, morphology, and syntax. Little progress, however, has been shown towards the structure of meaning, or content.

A common tool for the study of content, and structure in general for that matter, has been logic. During most of the 20th century, an important role of logic has been to study the structure of content of mathematical languages. Many logicians have moved on to apply the techniques developed to the analysis of natural languages—Frege, Russell, Carnap, Reichenbach, and Montague. An early introduction to such classical approaches can be found in [2].

As an illustration of the kind of problems that need to be addressed, consider the following examples. The following two sentences assert the existence of a man that both walks and talks:

> Some man that walks talks
> Some man that talks walks

The situations with respect to which these two sentences are true are the same, and hence a truth-conditional semantics needs to assign the same meaning to such sentences. Ambiguities arise easily in natural languages:

> Every man loves a woman

There are at least two distinct readings of this sentence. One says that for every man, there exists a woman that he loves, and the other says that there exists a woman that every man loves. Other problems are harder to qualify. Consider the following two sentences:

> Tarzan likes Jane
> Tarzan wants a girlfriend

The first sentence must be false if there is no Jane. On the other hand, the second sentence can be true even if no woman exists.

Those examples are extremely simple, some might even say naive, but they exemplify the issues for which a theory of natural language semantics must account. A guiding principle, apocryphally due to Frege, in the study of semantics is the so-called Fregean principle. Essentially, it can be stated as "the meaning of a complex expression should be a function of the meaning of its parts." Such a principle seems required to explain how natural languages can be learned. Since there is no arbitrary limit on both the length and the number of new sentences human beings can understand, some general principle such as the above must be at play. Moreover, since it would not be helpful

[3]

to require an infinite number of functions to derive the meaning of the whole from the meaning of the parts, a notion such as recursion must be at play as well.

That there is a recursive principle *à la Frege* at play both in syntax and semantics is hardly contested. What is contested is the interplay between the two. The classic work by Chomsky [6] advocated essentially the autonomy of syntax with respect to semantics. Chomsky's grammars are transformational: they transform the "surface" syntax of a sentence to extract its so-called deep structure. The semantics is then derived from the deep structure of the sentence. Some accounts of the Chomsky theory allows for a Fregean principle to apply at the level of the deep structure, while more recent accounts slightly complicate the picture. A different approach is to advocate a close correspondence between syntax and semantics. Essentially, the syntax can be seen as a map showing how the meaning of the parts are to be combined into the meaning of the whole.

The latter approach to semantics relies on two distinct developments. First, it is based on a kind of semantic analysis of language originating mainly with the work of Montague [9]. His was the first work that developed a large scale semantic description of natural languages by translation into a logical language that can be given a semantics using traditional techniques. The second development emerged from a particular syntactic analysis of language. During his analysis of logic, which led to development of the λ-calculus, Curry noticed that the types he was assigning to λ-terms could also be used to denote English word classes [7]. For example, in *John snores loudly*, the word *John* has type n, *snores* has type $n \Rightarrow s$, and *loudly* has type $s \Rightarrow s$. Independently, Lambek introduced a calculus of syntactic types, distinguishing two kinds of implication, reflecting the non-commutativity of concatenation [8]. The idea was to push all the grammar into the dictionary, assigning to each English word one or more types, and using the calculus to decide whether a string of words is a grammatically well-formed sentence. This work derived in part from earlier work by Ajdukiewicz [1] and Bar-Hillel [4].

This book, "Type-Logical Semantics" by Carpenter, explores this particular approach. Essentially, it relies on techniques from type theory: we assign a type (or more than one) to every word in the language, and we can check that a sentence is well-formed by performing what amounts to type-checking. In fact, it turns out that we can take the type-checking derivation proving that a sentence has the right type, and use the derivation to derive the semantics of the sentence. In the next sections, we will introduce the framework, and give simple examples to highlight the ideas. Carpenter pushes these ideas quite far, as we shall see when we cover the table of contents. We conclude with some opinions on the book.

5 To semantics...

The first problem we need to address is how to describe the semantics of language. We will follow in the truth-conditional tradition and model-theoretic ideas and we start with first-order logic. Roughly speaking, first-order logic provides one with constants denoting individuals, and predicates over such individuals. Simple example should illustrate this. Consider the sentence *Tarzan likes Jane*. Assuming constants **tarzan** and **jane**, and a predicate **like**, this sentence corresponds to the first-order logic formula $\mathbf{like}(\mathbf{tarzan}, \mathbf{jane})$. The sentence *Everyone likes Jane* can be expressed as $\forall x.\mathbf{like}(x, \mathbf{jane})$. This approach of using first-order logic to give semantics is quite straightforward. Unfortunately, for our purposes, it is also quite deficient. Let us see two reasons why that is. First, recall that we want a compositional principle at work in semantics. In other words, we want to be able to derive the meaning of *Tarzan likes Jane* from the meaning of *Tarzan* and *Jane*, and the meaning of *likes*. This sounds straightforward. However, the same principle should apply to the sentence *Tarzan and Kala like Jane*, corresponding to the first-order

formula $\mathbf{like}(\mathbf{tarzan}, \mathbf{jane}) \wedge \mathbf{like}(\mathbf{kala}, \mathbf{jane})$. Giving a compositional semantics seems to require giving a semantics to the extract *like Jane*. What is the semantics of such a part of speech? First-order logic cannot answer this easily. Informally, *like Jane* should have as semantics something that expects an individual (say x) and gives back the formula $\mathbf{like}(x, \mathbf{jane})$. A second problem is that the grammatical structure of a sentence can be lost during translation. This can lead to wild differences in semantics for similar sentences. For instance, consider the following sentences:

Tarzan likes Jane.
An apeman likes Jane.
Every apeman likes Jane.
No apeman likes Jane.

These sentences can be formalized as such in first-order logic, respectively:

$\mathbf{like}(\mathbf{tarzan}, \mathbf{jane})$
$(\exists x)(\mathbf{apeman}(x) \wedge \mathbf{like}(x, \mathbf{jane}))$
$(\forall x)(\mathbf{apeman}(x) \Rightarrow \mathbf{like}(x, \mathbf{jane}))$
$\neg(\exists x)(\mathbf{apeman}(x) \wedge \mathbf{like}(x, \mathbf{jane}))$, or equivalently, $(\forall x)(\mathbf{apeman}(x) \Rightarrow \neg\mathbf{like}(x, \mathbf{jane}))$

There seems to be a discrepancy among the logical contributions of the subjects in the above sentences. There is a distinction in the first-order logic translation of these sentences that is not expressed by their grammatical form.

It turns out that there is a way to solve those problems, by looking at an extension of first-order logic, known as higher-order logic [3]. Let us give enough theory of higher-order logic to see how it can be used to assign semantics to (a subset of) a natural language. This presentation presupposes a familiarity with both first-order logic and the λ-calculus [5].[4] There is a slight difference in our approach to first-order logic and our approach to higher-order logic. In the former, formulas, which represents properties of the world and its individuals, are the basic units of the logic. In higher-order logics, terms are the basic units, including constants and functions, with formulas explicitly represented as boolean-valued functions.

We start by defining a set of types that will be used to characterize the well-formedness of formulas, as well as derive the models. We assume a set of basic types $\mathsf{BasTyp} = \{Bool, Ind\}$, where *Bool* is the type of boolean values, and *Ind* is the type of individuals. (In first-order logic, the type *Bool* is not made explicit.) The set of types Typ is the smallest set such that $\mathsf{BasTyp} \subseteq \mathsf{Typ}$, and $(\sigma \to \tau) \in \mathsf{Typ}$ if $\sigma, \tau \in \mathsf{Typ}$. A type of the form $\sigma \to \tau$ is a functional (or higher-order) type, the elements of which map objects of type σ to objects of type τ.

The syntax of higher-order logic is defined as follows. Assume for each type $\tau \in \mathsf{Typ}$ a set Var_τ of variables and a set Con_τ of constants of that type. The set Term_τ of terms of type τ is defined as the smallest set such that:

$\mathsf{Var}_\tau \subseteq \mathsf{Term}_\tau$,
$\mathsf{Con}_\tau \subseteq \mathsf{Term}_\tau$,
$\alpha\beta \in \mathsf{Term}_\tau$ if $\alpha \in \mathsf{Term}_{\sigma\to\tau}$ and $\beta \in \mathsf{Term}_\sigma$, and
$\lambda x.\alpha \in \mathsf{Term}_\tau$ if $\tau = \sigma \to \rho$, $x \in \mathsf{Var}_\sigma$, and $\alpha \in \mathsf{Term}_\rho$.

What are we doing here? We are defining a term language. First-order logic introduces special syntax for its logical connectives ($\wedge$, $\vee$, $\neg$, $\Rightarrow$). It turns out, for higher-order logic, that we do not

[4]Higher-order logic is interesting in that it can either be viewed as a generalization of first-order logic, or as a particular instance of the simply-typed λ-calculus.

need to do that, we can simply define constants for those operators. (We will call these logical constants, because they will have the same interpretation in all models.) We will assume the following constants, at the following types: a constant **not** of type $Bool \to Bool$ and a constant **and** of type $Bool \to Bool \to Bool$, the interpretation of which should be clear, a family of constants $\mathbf{eq}_\tau$ each of type $\tau \to \tau \to Bool$, which checks for equality of two elements of type τ, and a family of constants $\mathbf{every}_\tau$ each of type $(\tau \to Bool) \to Bool$, used to capture universal quantification. The idea is that $\mathbf{every}_\tau$ quantifies over objects of type τ. In first-order logic, $\forall x.\varphi$ is true if for every possible individual i, replacing x by i in φ yields a true formula. Note that $\forall x$ binds the variable x in first-order logic. In higher-order logic, where there is only λ as a binder, we write the above as $\mathbf{every}_\tau(\lambda x.\varphi)$, which is true if φ is true for all objects of type τ.

This defines the syntax of higher-order logic. The models of higher-order logic are generalizations of the relational structures used to model first-order logic. A (standard) frame for higher-order logic is specified by giving for each basic type $\tau \in \mathsf{BasTyp}$ a domain Dom_τ of values of that type. These extend to functional types inductively: for a type $(\sigma \to \tau) \in \mathsf{Typ}$, $\mathsf{Dom}_{(\sigma\to\tau)} = \{f \mid f : \mathsf{Dom}_\sigma \to \mathsf{Dom}_\tau\}$, that is, the set of all functions from elements of Dom_σ to elements of Dom_τ. Let $\mathsf{Dom} = \bigcup_{\tau\in\mathsf{Typ}} \mathsf{Dom}_\tau$. We also need to given an interpretation for all the constants, via a function $[\![-]\!]_\tau : \mathsf{Con}_\tau \to \mathsf{Dom}_\tau$ assigning to every constant of type τ an object of type τ. (We simply write $[\![-]\!]$ when the type is clear from the context.) Hence, a model for higher-order logic is of the form $M = (\mathsf{Dom}, [\![-]\!])$. We extend the interpretation $[\![-]\!]$ to all the terms of the language. To deal with variables, we define an assignment to be a function $\theta : \mathsf{Var} \to \mathsf{Dom}$ such that $\theta(x) \in \mathsf{Dom}_\tau$ if $x \in \mathsf{Var}_\tau$. We denote $\theta[x := a]$ the assignment that maps x to a and $y \neq x$ to $\theta(y)$. We define the denotation $[\![\alpha]\!]^\theta_M$ of the term α with respect to the model $M = \langle \mathsf{Dom}, [\![-]\!]\rangle$ and assignment θ as:

$[\![x]\!]^\theta_M = \theta(x)$ if $x \in \mathsf{Var}$,
$[\![c]\!]^\theta_M = [\![c]\!]$ if $c \in \mathsf{Con}$,
$[\![\alpha(\beta)]\!]^\theta_M = [\![\alpha]\!]^\theta_M([\![\beta]\!]^\theta_M)$, and
$[\![\lambda x.\alpha]\!]^\theta_M = f$ such that $f(a) = [\![\alpha]\!]^{\theta[x:=a]}_M$.

Standard frames are subject to restrictions. For instance, the domain corresponding to boolean values must be a two-element domain, such as $\mathsf{Dom}_{Bool} = \{\mathbf{true}, \mathbf{false}\}$. Moreover, they must give a fixed interpretation to the logical constants (i.e., the conjunction operator should actually behave as a conjunction operator). Hence, we require:

$$[\![\mathbf{not}]\!](b) = \begin{cases} \mathbf{false} & \text{if } b = \mathbf{true} \\ \mathbf{true} & \text{if } b = \mathbf{false} \end{cases}$$

$$[\![\mathbf{and}]\!](b_1)(b_2) = \begin{cases} \mathbf{true} & \text{if } b_1 = \mathbf{true} \text{ and } b_2 = \mathbf{true} \\ \mathbf{false} & \text{otherwise} \end{cases}$$

$$[\![\mathbf{eq}_\tau]\!](v_1)(v_2) = \begin{cases} \mathbf{true} & \text{if } v_1 = v_2 \\ \mathbf{false} & \text{otherwise} \end{cases}$$

$$[\![\mathbf{every}_\tau]\!](f) = \begin{cases} \mathbf{true} & \text{if } f(x) = \mathbf{true} \text{ for all } x \in \mathsf{Dom}_\tau \\ \mathbf{false} & \text{otherwise} \end{cases}$$

One can check that if we define $\mathbf{some}_\tau$ as $\lambda P.\mathbf{not}(\mathbf{every}_\tau(\lambda x.\mathbf{not}(P(x))))$, it has the expected interpretation. Note that we will often use the abbreviations $\varphi \wedge \psi$ for $\mathbf{and}(\varphi, \psi)$, and $\neg\varphi$ for $\mathbf{not}(\varphi)$.

A formula of higher-order logic is a term of type *Bool*. We say that a model M satisfies a formula φ if $[\![\varphi]\!]_M = \mathbf{true}$ in the model. Two terms are said to be logically equivalent if they have the same interpretation in all models. One can check, for instance, that $\lambda x.r(x)$ and r are logically equivalent, as are $(\lambda x.\alpha)\beta$ and $\alpha\{\beta/x\}$ (that is, α where every occurrence of x is replaced by β).

For example, consider the following simple three individual model M, with constants **tarzan**, **jane**, **kala** and **like**.

$$\mathsf{Dom}_{Ind} = \{t, j, k\}$$

$$[\![\mathbf{tarzan}]\!] = t \quad [\![\mathbf{jane}]\!] = j \quad [\![\mathbf{kala}]\!] = k$$

$$[\![\mathbf{like}]\!] = \left[\begin{array}{lcl} t & \mapsto & \left[\begin{array}{lcl} t & \mapsto & \mathbf{false} \\ j & \mapsto & \mathbf{true} \\ k & \mapsto & \mathbf{true} \end{array}\right] \\ j & \mapsto & \left[\begin{array}{lcl} t & \mapsto & \mathbf{true} \\ j & \mapsto & \mathbf{false} \\ k & \mapsto & \mathbf{false} \end{array}\right] \\ k & \mapsto & \left[\begin{array}{lcl} t & \mapsto & \mathbf{true} \\ j & \mapsto & \mathbf{false} \\ k & \mapsto & \mathbf{true} \end{array}\right] \end{array}\right]$$

This model M satisfies the term $\mathbf{like}(\mathbf{kala})(\mathbf{tarzan})$ (*Kala likes Tarzan*) as $[\![\mathbf{like}(\mathbf{kala})(\mathbf{tarzan})]\!] = [\![\mathbf{like}]\!](k)(t) = \mathbf{true}$. It also satisfies the term $\mathbf{some}_{Ind}(\mathbf{like}(\mathbf{jane}))$ (*There is someone Jane likes*). It does not satisfy the term $\mathbf{every}_{Ind}(\lambda x.\mathbf{like}(x)(x))$ (*Everyone likes himself/herself*).

We will use higher-order logic to express our semantics. The idea is to associate with every sentence (or part of speech) a higher-order logic term. We can then use the semantics of higher-order logic to derive the truth value of the sentence. Consider the examples at the beginning of the section. We assume constants **tarzan**, **kala** and **jane** of type *Ind*, and a constant **like** of type $Ind \to Ind \to Bool$. We can translate the sentence *Tarzan likes Jane* as $\mathbf{like}(\mathbf{tarzan})(\mathbf{jane})$, as in first-order logic. But now we can also translate the part of speech *like Jane* independently as $\lambda x.\mathbf{like}(x)(\mathbf{jane})$.

For a more interesting example, consider the treatment of noun phrases as given at the beginning of the section. The solution to the problem of losing the grammatical structure was solved by Russell by treating all noun phrases as though they were functions over their verb phrases. This is analogous to what is already happening with the definition of $\mathbf{every}_{Ind}$ in higher-order logic, which has type $(Ind \to Bool) \to Bool$. Such generalized quantifier takes a property of an individual (a property has type $Ind \to Bool$) and produces a truth value—in the case of **every**, the truth value is true if every individual has the supplied property. A similar abstraction can be applied to a noun position. We define a generalized determiner as a function taking a property stating a restriction on the quantified individuals, and returning a generalized quantifier obeying that restriction. Hence, a generalized determiner has type $(Ind \to Bool) \to (Ind \to Bool) \to Bool$. Consider the following generalized determiners, used above:

$$\mathbf{some}^2 = \lambda P.\lambda Q.\mathbf{some}(\lambda x.P(x) \wedge Q(x))$$
$$\mathbf{every}^2 = \lambda P.\lambda Q.\mathbf{every}(\lambda x.P(x) \Rightarrow Q(x))$$
$$\mathbf{no}^2 = \lambda P.\lambda Q.\neg\mathbf{some}(\lambda x.P(x) \wedge Q(x))$$

One can check that the sentence *An apeman likes Jane* becomes $\mathbf{some}^2(\mathbf{apeman})(\lambda x.\mathbf{like}(x)(\mathbf{jane}))$, that *Every apeman likes Jane* becomes $\mathbf{every}^2(\mathbf{apeman})(\lambda x.\mathbf{like}(x)(\mathbf{jane}))$, and that *No apeman likes Jane* becomes $\mathbf{no}^2(\mathbf{apeman})(\lambda x.\mathbf{like}(x)(\mathbf{jane}))$. The subject is interpreted as $\mathbf{some}^2(\mathbf{apeman})$, $\mathbf{every}^2(\mathbf{apeman})$ and $\mathbf{no}^2(\mathbf{apeman})$ respectively. The verb phrase *likes Jane* is given the expected semantics $\lambda x.\mathbf{like}(x)(\mathbf{jane})$. What about the original sentence *Tarzan likes Jane*. According to the above, we should be able to give a semantics to *Tarzan* (when used as a subject) with a type

$(Ind \to Bool) \to Bool$. One can check that if we interpret *Tarzan* as $\lambda P.P(\mathbf{tarzan})$, we indeed get the required behavior. Hence, we see that the noun phrase can be given the uniform type $(Ind \to Bool) \to Bool$, and that higher-order logic can be used to derive a uniform, compositional semantics.

6 ... from syntax

We have seen in the previous section how we can associate to sentences a semantics in higher-order logic. More importantly, we have seen how we can assign a semantics to sentence extracts, in a way that does capture the intuitive meaning of the sentences. The question at this point is how to derive the higher-order logic term corresponding to a given sentence or sentence extract.

The grammatical theory we use to achieve this is *categorial grammars*, originally developed by Ajdukiewicz [1] and later Bar-Hillel [4]. In fact, we will use a generalization of their approach due to Lambek [8]. The idea behind categorial grammars is simple. We start with a set of *categories*, each category representing a grammatical function. For instance, we can start with the simple categories np representing noun phrases, n representing nouns, and s representing sentences. Given categories A and B, we can form the *functor* categories A/B and $B\backslash A$. The category A/B represents the category of syntactic units that take a syntactic unit of category B to their right to form a syntactic unit of category A. Similarly, the category $B\backslash A$ represents the category of syntactic units that take a syntactic unit of category B to their left to form a syntactic unit of category A. Consider some examples. The category n/n represents the category of prenominal modifiers, such as adjectives: they take a noun on their right and form a noun. The category $n\backslash n$ represents the category of postnominal modifiers. The category $np\backslash s$ is the category of intransitive verbs: they take a noun phrase on their left to form a sentence. Similarly, the category $(np\backslash s)/np$ represents the category of transitive verbs: they take a noun phrase on their right to then expect a noun phrase on their left to form a sentence.

Before deriving semantics, let's first discuss well-formedness, as this was the original goal for such grammars. The idea was to associate to every word (or complex sequence of words that constitute a single lexical entry) one or more categories. We will call this the dictionary, or lexicon. The approach described by Lambek [8] is to prescribe a calculus of categories so that if a sequence of words can be assigned a category A according to the rules, then the sequence of words is deemed a well-formed syntactic unit of category A. Hence, a sequence of words is a well-formed sentence if it can be shown in the calculus that it has category s. As an example of reduction, we see that if σ_1 has category A and σ_2 has category $A\backslash B$, then $\sigma_1\ \sigma_2$ has category B. Schematically, $A, A\backslash B \Rightarrow B$. Moreover, this goes both ways, that is, if $\sigma_1\ \sigma_2$ has category B and σ_1 can be shown to have category A, then we can derive that σ_2 has category $A\backslash B$.

It was the realization of van Benthem [12] that this calculus could be used to assign a semantics to terms and use the derivation of categories to derive the semantics. The semantic will be given in some higher-order logic as we saw above. We assume that to every basic category corresponds a higher-order logic type. Such a type assignment T can be extended to functor categories by putting $T(A/B) = T(B\backslash A) = T(B) \to T(A)$. We extend the dictionary so that we associate with every word one or more categories, and a corresponding term of higher-order logic. We stipulate that the term α corresponding to a word in category A should have a type corresponding to the category, i.e. $\alpha \in \mathsf{Term}_{T(A)}$.

We will use the following notation (called a sequent) $\alpha_1 : A_1, \ldots, \alpha_n : A_n \Rightarrow \alpha : A$ to mean that expressions $\alpha_1, \ldots, \alpha_n$ of categories $A_1, \ldots, A_n$ can be concatenated to form an expression α of category A. We will use capital Greek letters (Γ, Δ,...) to represent sequences of expressions and

categories. We now give rules that allow us to derive new sequents from other sequents:

$$\frac{}{\alpha : A \Rightarrow \alpha : A} \qquad \frac{\Gamma_2 \Rightarrow \beta : B \quad \Gamma_1, \beta : B, \Gamma_3 \Rightarrow \alpha : A}{\Gamma_1, \Gamma_2, \Gamma_3 \Rightarrow \alpha : A}$$

In other words, if Γ_2 can concatenate into an expression β with category B, and if $\Gamma_1, \beta : B, \Gamma_3$ can concatenate into an expression α with category A, then $\Gamma_1, \Gamma_2, \Gamma_3$ can concatenate into α with category A.

$$\frac{\Delta \Rightarrow \beta : B \quad \Gamma_1, \alpha(\beta) : A, \Gamma_2 \Rightarrow \gamma : C}{\Gamma_1, \alpha : A/B, \Delta, \Gamma_2 \Rightarrow \gamma : C} \qquad \frac{\Delta \Rightarrow \beta : B \quad \Gamma_1, \alpha(\beta) : A, \Gamma_2 \Rightarrow \gamma : C}{\Gamma_1, \Delta, \alpha : B\backslash A, \Gamma_2 \Rightarrow \gamma : C}$$

$$\frac{\Gamma, x : A \Rightarrow \alpha : B}{\Gamma \Rightarrow \lambda x.\alpha : B/A} \qquad \frac{x : A, \Gamma \Rightarrow \alpha : B}{\Gamma \Rightarrow \lambda x.\alpha : A\backslash B}$$

For example, the following is a derivation of *Tarzan likes Jane*.

$$\dfrac{\dfrac{}{\mathbf{tarzan} : np \Rightarrow \mathbf{tarzan} : np} \qquad \dfrac{\dfrac{}{\mathbf{jane} : np \Rightarrow \mathbf{jane} : np} \quad \dfrac{}{\mathbf{like}(\mathbf{tarzan})(\mathbf{jane}) : s \Rightarrow \mathbf{like}(\mathbf{tarzan})(\mathbf{jane}) : s}}{\mathbf{like}(\mathbf{tarzan}) : s/np, \mathbf{jane} : np \Rightarrow \mathbf{like}(\mathbf{tarzan})(\mathbf{jane}) : s}}{\mathbf{tarzan} : np, \mathbf{like} : np\backslash s/np, \mathbf{jane} : np \Rightarrow \mathbf{like}(\mathbf{tarzan})(\mathbf{jane}) : s}$$

For example, the following derivation of the sentence fragment *Tarzan likes* shows that it is of the type s/np—it is an expression that expects a noun phrase to the right to form a complete sentence.

$$\dfrac{\dfrac{\dfrac{}{\mathbf{tarzan} : np \Rightarrow \mathbf{tarzan} : np} \qquad \dfrac{\dfrac{}{x : np \Rightarrow x : np} \quad \dfrac{}{\mathbf{like}(\mathbf{tarzan})(x) : s \Rightarrow \mathbf{like}(\mathbf{tarzan})(x) : s}}{\mathbf{like}(\mathbf{tarzan}) : s/np, x : np \Rightarrow \mathbf{like}(\mathbf{tarzan})(x) : s}}{\mathbf{tarzan} : np, \mathbf{like} : np\backslash s/np, x : np \Rightarrow \mathbf{like}(\mathbf{tarzan})(x) : s}}{\mathbf{tarzan} : np, \mathbf{like} : np\backslash s/np \Rightarrow \lambda x.\mathbf{like}(\mathbf{tarzan})(x) : s/np}$$

A look at the theory underlying type-logical approaches to linguistics reveals some fairly deep mathematics at work. The fact that we can derive the semantics in parallel with a derivation of the categories associated with the sequence of words is not an accident. In fact, it is a phenomenon known as a Curry-Howard isomorphism. The original Curry-Howard isomorphism was a correspondence between intuitionistic propositional logic and the simply-typed λ-calculus: every valid formula of intuitionistic propositional logic corresponds to a type in the simply-typed λ-calculus, in such a way that a proof of the formula corresponds to a λ-term of the corresponding type. Such a correspondence exists between the Lambek calculus (which can be seen as a substructural logic, namely intuitionistic bilinear logic) and an appropriate instance of the λ-calculus, namely higher-order logic.

We have in this review merely sketched the basics of the type-logic approach, a merciless summary of the first few chapters of the book. Carpenter investigates more advanced linguistic phenomena by extending the Lambek calculus with more categorial constructions, and deriving the corresponding semantics. For instance, he deals with generalized quantifiers, deriving the semantics we hinted at earlier through a syntactic derivation, as well as plural forms, and modalities such as belief. The latter requires a move to a modal form of higher-order logic known as intensional logic.

7 The book

The book naturally divides in three parts. The first part, the first five chapters (as well as an appendix on mathematical preliminaries), introduces the technical machinery required to deal with linguistic issues, namely the higher-order logic used to express the semantics, and the Lambek calculus to derive the semantics.

Chapter 1, **Introduction**, provides an outline of the role of semantics in linguistic theory. Carpenter discusses the central notions of truth and reference, the latter telling us how linguistic expressions can be linked to objects in the world. He gives a survey of topics that linguistic theories need to address, including synonymy, contradiction, presupposition, ambiguity, vagueness. He also surveys topics in pragmatics, the branch of linguistic concerned with aspects of meaning that involve more than literal interpretation of utterances. Finally, he argues for the methodology of the book, in terms of originality, compositionality, model theory and grammar fragments. Some caveats apply: he studies models of natural language itself, not models of our knowledge or ability to use language; furthermore, these models are not intended to have any metaphysical interpretation, but are only a description and approximation of natural language.

Chapter 2, **Simply Typed λ-Calculus**, lays out the basic theory of the simply typed λ-calculus. The simply typed λ-calculus provides an elegant solution to the problem of giving a denotation for the basic expressions of a language in a compositional manner, as explained in Chapter 3. This chapter concentrates on the basic theory, describing the language of the simply typed λ-calculus, along with a model theory and a proof theory for the logical language, that formalizes whether two λ-calculus expressions are equal (have the same denotation in all models). The standard λ-calculus notions of reductions, normal forms, strong normalization, the Church-Rosser theorem, and combinators are discussed. An extension of the simply typed λ-calculus with sums and products is described.

Chapter 3, **Higher-Order Logic**, introduces a generalization of first-order logic where quantification and abstraction occurs over all the entities of the language, including relations and functions. Higher-order logic is defined as a specific instance of the simply typed λ-calculus, with types capturing both individuals and truth values, and logical constants such as conjunction, negation, and universal quantification. The usefulness of the resulting logic is demonstrated by showing how it can handle quantifiers in natural languages in a uniform way. The proof theory of higher-order logic is discussed.

Chapter 4, **Applicative Categorial Grammar**, is an introduction to the syntactic theory from which the denotation of natural language terms is derived, that of categorial grammars. Categorial grammars are based on the notion of categories representing syntactic functionality, and describe how to syntactically combine entities in different categories to form combined entities in new categories. The framework described in this chapter is the simplest form of applicative categorial grammar, which will be extended in later chapters. After introducing the basic categories, the chapter shows how to assign semantic domains to categories, and how to associate with every basic syntactic entity a term in the corresponding domain, creating a lexicon. The basics of how to derive the semantic meaning of a composition of basic syntactic entities based on the derivation of categories is explored. Finally, a discussion of some of the consequences of this way of assigning semantic meaning is given; mainly, it focuses on ambiguity and vagueness, corresponding respectively to expressions with multiple meanings, and expressions with a single undetermined meaning.

Chapter 5, **The Lambek Calculus**, introduces a logical system that extends the applicative categorial grammar framework of the previous chapter. The Lambek calculus allows for a more flexible description of the possible ways of putting together entities in different categories. The

Lambek calculus is presented both in sequent form and in natural deduction form, the former appropriate for automatic derivations, the latter more palatable for humans. The Lambek calculus is decidable (i.e., the problem of determining whether the calculus can show a given sentence grammatical is decidable). The correspondence between the Lambek calculus and a variant of linear logic is established.

The following four chapters show how to apply the machinery of the first part to different aspects of linguistic analysis.

Chapter 6, **Coordination and Unbounded Dependencies**, studies two well-known linguistic applications of categorial grammars. The first, coordination, corresponds to the use of *and* in sentences. Such a coordination operator can occur on many levels, coordinating two nouns (*Joe and Victoria*), two adjectives (*black and blue*), two sentences, etc. Coordination at any level is achieved by lifting the coordination to the level of sentences, via the introduction of a polymorphic coordination operator in the semantic framework. This operator can be handled in the Lambek calculus via type lifting. The resulting system remains decidable. An extension of the Lambek calculus with conjunction and disjunction is considered, to account for coordinating, for example, unlike complements of a category, such as in *Jack is a good cook and always improving.* The second well-known use of categorial grammars is to account for unbounded dependencies, that is, relationships between distant expressions within an expression, the distance potentially unbounded. This is handled by introducing a new categorial combinator $A \uparrow B$, an element of which can be analyzed as an A with a B missing somewhere within it. The appropriate derivation rules can be added to the Lambek calculus.

Chapter 7, **Quantifiers and Scope**, studies the contribution of quantified noun phrases to the meaning of phrases in which they occur. Such generalized quantifiers, such as *every kid*, or *some toy*, are traditionally problematic because they take semantic scope around an arbitrary amount of material. For instance, *every kid played with some toy* has two readings, depending on the scope of the quantifiers *every* and *some* (is there a single toy with which every kid plays, or does every kid play with a possibly different toy?) Accounting for such readings is the aim of this chapter. Two historically significant approaches to quantifiers are surveyed: Montague's quantifying in approach, and Cooper's storage mechanism. Then, the type-logical solution of Moortgat is described. The idea is to introduce a new category $B \Uparrow A$ of expressions that act locally as B's but take their semantic scope over an embedding expression of category A. Generalized quantifiers are given category $np \Uparrow s$, since they act like a noun phrase (category np) *in situ*, but scope semantically to an embedding sentence (category s). The Lambek calculus is extended with appropriate derivation rules. The issues of quantifier coordination, quantifiers within quantifiers, and the interaction with negation are discussed. Other topics related to quantifiers and determiners in general, such as definite descriptions, possessives (*every kid's toy*), indefinites (*some student*), generics (*italians*), comparatives (*as tall as*), and expletives (*it*, *there*) are analyzed within that context.

Chapter 8, **Plurals**, provides a type-logical account of plurality. First, the notion of group is added to the syntax and semantics. The type *Group* is considered to be a subtype of the type *Ind* and thus the domain of *Group* is a subset of the domain of *Ind*. A relation linking a group to the property that defines membership in the group is defined, and restrictions are imposed to ensure that every group has a unique property that defines membership of that group. With this interpretation, categories for plural noun phrases and plural nouns are studied. The notions of distributors (to view a group as a set of individuals) and collectors (to view a set of individuals as a group) are defined, to handle, for example, verbs that apply only to individuals or only to groups. The issues of coordination and negation are examined in the context of plurals. Further

topics examined include plural quantificatives and, more generally, partitives (*each*, *all*, *most*, or numerical partitives such as *three of*, etc.), nonboolean coordination with *and*, comitative (the use of *with* in *Tarzan climbed the tree with Cheetah*), and mass terms such as *snow* and *water*.

Chapter 9, **Pronouns and Dependency**, analyses the use of non-indexical pronouns such as *him*, *she*, *itself*, especially the dependent use of such pronouns. Dependent pronouns are characterized as having their interpretation depend on the interpretation of some other expression (the antecedent). For example, *he* in *Jody believes he will be famous*. A popular interpretation of pronouns in type-logical frameworks is as variables, although the treatment is subtle, at least for non-reflexive pronouns such as the *he* above. (Admittedly, this topic is an outstanding problem for type-logical grammars.) Reflexive pronouns, such as *himself* in *Everyone likes himself*, can be handled as quantifiers. Topics related to pronomial forms are examined, such as reciprocals (the *each other* in *The three kids like each other*), pied piping (the *which* in *the table the leg of which Jody broke*), ambiguous verb-phrases ellipses (*Jody likes himself and Brett does too*), and interrogatives.

The final part, the last three chapters, extend the framework with modalities to account for intensional aspects of natural languages.

Chapter 10, **Modal Logic**, introduces the logical tools required to deal with intensionality, tense and aspect. The key concept is that of a modal logic, where operators are used to qualify the truth of a statement. The chapter presents both a model theory (Kripke frames) and a proof theory for S5, a particular modal logic of necessity. A brief discussion of how the techniques of modal logic can be used to model indexicality precedes the presentation of a general modal model. First-order tense logics, which extend first order logics with modal operators about the truth of statements in the past and future, are presented in some depth, as they are able to provide a model of tenses in natural language. Time can be regarded as a collection of moments, or as a collection of possibly overlapping intervals. Higher order logic is extended to include modal operators by taking the domains of worlds and time to be basic types, on the same level as the domains of individuals and truth values, yielding a framework referred to as intensional logic. This approach avoids a number of problems associated with simply abstracting the model for higher order logic over possible worlds.

Chapter 11, **Intensionality**, uses modal logic to extend the type-logical framework to cover intensional constructions. In particular, *World* is added as a new basic type, and the assignment of types to basic categories is modified, replacing *Bool* with $World \rightarrow Bool$, i.e. truth values may be different at different worlds. This change facilitates the inclusion of many constructs, such as propositional attitudes (*Frank believes Brooke cheated*), modal adverbs (*possibly*), modal auxiliaries (*should*, *might*), and so-called control verbs (*persuaded*, *promised*), although some constructs remain problematic. The "individual concepts" approach is considered, where the type of a noun phrase is $World \rightarrow Ind$ instead of *Ind*, i.e. the referent of a noun phrase may differ from world to world. Other approaches to intensionality, which do not involve possible worlds, are explained briefly. Finally, the last section returns to the issue of giving a categorization of control verbs, and gives some problematic examples showing the need for more work in this area.

Chapter 12, **Tense and Aspect**, extends the grammar and semantics with a theory of tense. It presents Reichenbach's approach to simple and perfect tenses, how this applies to discourse, and Vendler's verb classes—a semantic classification of verbs that is correlated with their syntactic use. The approach Carpenter adopts for tense and aspect is based on insights derived from these works, and on further development of these works by other authors. To extend the grammar, verbs are subcategorized by classifying them based on whether they are finite or non-finite, and whether they involve simple or perfect tense, resulting in several different categories for sentences. A new basic

type is introduced, representing time periods, and all of the sentence categories are assigned the same type: functions from time periods to truth values. The temporal argument always corresponds to the time of the event being reported. (This is essentially similar to the intensional approach of the previous chapter, but here we distinguish time periods from possible worlds.) From this beginning, the grammar is developed to encompass many of the English constructs involving tense and aspect. Many of these constructs are very complex in their usage and generally there do not seem to be simple and complete solutions to incorporating them into the grammar.

8 Opinion

There are some typos (potentially confusing, as they sometimes occur in the types for functions), as well as some glossing over central topics (such as the discussion of groups in Chapter 8). Carpenter doesn't generally delve into syntactic explanations, that is, explaining why the theory of syntax he develops does or does not permit certain sentences. Moreover, for linguists, it may be important to note that Carpenter does not develop a theory of morphology (the structure of words at the level of morphemes).

This book fills a sorely void niche in the field of semantics of natural languages via type-logical approaches. There are some books on the subject, but the most accessible are severely limited in their development [13], while the others are typically highly mathematical and focus on the metatheory of the type-logical approach [10].

Carpenter's book is a reasonable blend of mathematical theory and linguistic applications. Its great strength is an excellent survey of type-logical approaches applied to a great variety of linguistic phenomena. On the other hand, the preliminary chapters presenting the underlying mathematical theory are slightly confusing—not necessarily surprising considering the amount of formalism needed to account for all the linguistic phenomena studied. A background or at least exposure to ideas from both logic and programming language semantics is extremely helpful. In this sense, this book seems slightly more suited, at least as an introductory book, to mathematicians and computer scientists interested in linguistic applications, than to linguists interested in learning about applicability of type-logical approaches. (Although this book could nicely follow a book such as [13], or any other introductory text on type-logical grammars that focuses more on the "big picture" than on the underlying mathematical formalisms.) People that are not linguists will most likely find chapters 9 and on hard to follow, as they assume more and more knowledge of linguistic phenomena.

This book points to interesting areas of ongoing research. In particular, the later sections of the book on aspects of intensionality highlight areas where the semantics of natural languages are not clear. (This is hardly a surprise, as intensional concepts have always been problematic, leading philosophers to develop many flavors of modal logics to attempt to explain such concepts.) Another avenue of research that is worth pointing out, although not discussed in this book, is the current attempt to base semantics of natural languages not on higher-order logic as presented in this book, but rather on Martin-Löf constructive type theory, via categorial techniques [11].

References

[1] K. Ajdukiewicz. Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27, 1935.

[2] J. Allwood, L.-G. Andersson, and O. Dahl. *Logic in Linguistics*. Cambridge Textbooks in Linguistics. Cambridge University Press, 1977.

[3] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof*. Academic Press, 1986.

[4] Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.

[5] H. P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. Studies in Logic. North-Holland, Amsterdam, 1981.

[6] N. Chomsky. *Syntactic Structures*. Mouton and Co., 1957.

[7] H. B. Curry. Some logical aspects of grammatical structure. In *American Mathematical Society Proceedings of the Symposia on Applied Mathematics 12*, pages 56–68, 1961.

[8] J. Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65:154–170, 1958.

[9] R. Montague. *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, 1974.

[10] G. V. Morrill. *Type Logical Grammar: Categorial Logic of Signs*. Kluwer Academic Publishers, 1994.

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

[12] J. van Benthem. The semantics of variety in categorial grammar. In W. Buszkowski, J. van Benthem, and W. Marciszewski, editors, *Categorial Grammar*, number 25 in Linguistics and Literary Studies in Eastern Europe, pages 37–55. John Benjamins, 1986. Previously appeared as Report 83-29, Department of Mathematics, Simon Fraser University (1983).

[13] M. M. Wood. *Categorial Grammars*. Routledge, 1993.

Review of
The π-calculus: A Theory of Mobile Processes
Author of Book: D. Sangiorgi and D. Walker
Author of Review: Riccardo Pucella, Dept of CS, Cornell
Cambridge University Press, 585 Pages

9 Introduction

With the rise of computer networks in the past decades, the spread of distributed applications with components across multiple machines, and with new notions such as mobile code, there has been a need for formal methods to model and reason about concurrency and mobility. The study of sequential computations has been based on notions such as Turing machines, recursive functions, the λ-calculus, all equivalent formalisms capturing the essence of sequential computations. Unfortunately, for concurrent programs, theories for sequential computation are not enough. Many programs are not simply programs that compute a result and return it to the user, but rather interact with other programs, and even move from machine to machine.

Process calculi are an attempt at getting a formal foundation based on such ideas. They emerged from the work of Hoare [4] and Milner [6] on models of concurrency. These calculi are meant to model systems made up of processes communicating by exchanging values across channels. They