

Chapter 15

Syntax-Semantics

Transparency

The previous chapter discussed the problem of finite knowledge and the three requirements it makes on theories of natural language meaning.

The first requirement is a syntactic theory of natural languages consisting of a finite number of rules. In this book, I adopt combinatory categorial grammar (CCG) (Ades and Steedman, 1982; Steedman, 1988, 1997, 2000) as a syntactic theory^{*1}, which I will introduce in the next section.

The second requirement is a hypothesis about what the meaning of a sentence is, and a theory of meaning representations based on that hypothesis. It was mentioned at the end of the previous chapter that, based on the hypothesis that the meaning of a sentence is its verification condition, this book represents the meaning of a sentence with a *type* of dependent type theory.

The third requirement is to present a method for calculating the meaning of the whole from the meaning of the parts for each constituent that the syntactic theory defines. If CCG is adopted as a syntactic theory, it is sufficient to show how to calculate the meaning of the mother node from the meaning of the daughter nodes for each combinatory rules of CCG. A mechanism is then required to ensure that the semantic representation of a sentence is a type in DTT, which is given in this chapter as the notion of *semantic felicity condition*.

This chapter describes how DTS, as the theory of meaning, satisfies the above three requirements.

^{*1} CCG is an example of a syntactic theory that a theory of meaning can adopt, but it does not mean that a theory of meaning cannot be constructed without adopting CCG. Various other categorial grammars have been proposed, and the readers can adopt a categorial grammar to which they are committed, if any, while adopting it, in a similar way to this book, to the syntactic structure of that syntactic theory, and consider a method of assigning semantic indications by DTS.

However, it is worth noting that CCG has several advantages over other syntactic theories. First, the syntax of natural languages is claimed to be a mildly-context sensitive grammar (or type 1.5 grammar), and CCG is the weakest generative grammar among the family of type 1.5 grammars, i.e. the most modest. Second, CCG has the CYK algorithm available in parsing, with a computational complexity of $O(n^3)$, while most of the other categorial grammars have complexities of polynomial, exponential, or even NP-complete. This suggests that CCG, while being a theory of linguistic competence, serve also as a theory of language performance as it is.

15.1 Combinatory Categorial Grammar

CCG is one of the syntactic theories called *categorial grammars*. Categorical grammars are theories of grammar that regard the syntactic structure of a sentence as a proof diagram in substructural logic^{*2}, namely, a proof diagram showing that a sequence of words is a sentence.

As a logical system, the formal definition of CCG consists of the definitions of propositions, inference rules, and axioms. In CCG, what play the role of propositions are *syntactic types*, which represent parts of speech in syntactic theory. The definition of the syntactic types are given as follows.

Definition 320. $S, NP, N, PP, CONJ, CP, RN$ are *base syntactic types* of CCG.

S is specified as a syntactic type for a sentence. $NP, N, PP, CONJ, CP, RN$ are syntactic types for noun phrases, common nouns, prepositional phrases, conjunctions, complementizer phrases, and relative nouns, respectively. From this set of base syntactic types, a collection of syntactic types is recursively defined as follows.

Definition 321. A collection of *syntactic types* of CCG is recursively defined as follows:

1. Base syntactic types are syntactic types.
2. If X and Y are syntactic types, then X/Y and $X \backslash Y$ are syntactic types.

The inference rules in CCG are called *combinatory rules*. For example, CCG adopts the following two combinatory rules (forward function application rule and inverse function application rule) as inference rules.

Function Application Rules

$$\frac{X/Y \quad Y}{X} > \quad \frac{Y \quad X \backslash Y}{X} <$$

Seen as logical rules, X, Y are syntactic types, and these rules correspond to the elimination rules for right and left implication in substructural logic.^{*3} Regarding the

^{*2} For more information on substructural logics, refer to textbooks such as Schroeder-Heister and Dosen (1994), Restall (2000), and Galatos et al. (2007), among others. For the relationship between categorial grammars and substructural logics, see Buszkowski (2010), and for the relation between CCG and substructural logics, see Bekki (2011).

^{*3} The introduction rules of implication are explicitly adopted in categorial grammars other than CCG, but since CCG is a Hilbert-style substructural logic, the introduction rules of implication are obtained via the (weak) deduction theorem: When CCG adopt all of function application rules, function composition rules and type raising rules, a weak deduction theorem holds, which provides a mechanism for deriving long-distance dependency (or wh-movement) in theoretical syntax.

notation, categorial grammars including CCG adopt the vertical proof diagram commonly used in natural deduction.

While combinatory rules play the role of inference rules, *lexical items*, which describes the behaviours of words, play the role of axioms. A *lexicon* is a collection of lexical items, and the list below is an example of it. Each lexical item represents the syntactic type of the corresponding word.

Lexical Items

John $\vdash NP$
 Mary $\vdash NP$
 runs $\vdash S \backslash NP$
 laughs $\vdash S \backslash NP$
 loves $\vdash S \backslash NP / NP$
 hates $\vdash S \backslash NP / NP$

or equivalently, using the vertical notation,

Lexical Items (in vertical notation)

$\frac{\text{John}}{NP}$ $\frac{\text{Mary}}{NP}$ $\frac{\text{runs}}{S \backslash NP}$...

The combination of combinatory rules and the lexicon is, in logic, a combination of inference rules and axioms, and the proof diagram leading to the syntactic type S is regarded as a proof that a given sequence of words is a sentence. (66) is an example of a transitive sentence.

(66) John loves Mary.

(67) is the syntactic structure of (66), represented by a tree structure according to the notation of natural deduction.*⁴

(67) Syntactic Structure of (66)

$$\frac{\frac{\frac{\text{John}}{NP} \quad \frac{\frac{\text{loves}}{S \backslash NP / NP} \quad \frac{\text{Mary}}{NP}}{S \backslash NP} >}{S} <$$

The second CCG combination rule is the following coordination rule.

*⁴ The sequent calculi notation can also be used as follows, but these are only notational differences that is not essential which one to use.

$$\frac{\frac{\text{John} \vdash NP \quad \frac{\frac{\text{loves} \vdash S \backslash NP / NP \quad \text{Mary} \vdash NP}{\text{loves, Mary} \vdash S \backslash NP} >}{\text{John, loves, Mary} \vdash S} <$$

Coordination Rule

$$\frac{X \quad CONJ_X \quad X}{X} \langle \Phi \rangle_X$$

where X is S -reducible.

The coordination rule connects two constituents with the same syntactic type by a conjunction such as *and*, *or*, *but*, and so forth. However, not every syntactic type instantiates the syntactic type X : the syntactic type X to be coordinated by the coordination rule must be an S -reducible syntactic type. Intuitively, a syntactic type is S -reducible if it takes zero or more types to finally become a syntactic type S . Precise definition for a general syntactic type X is given as follows.

Definition 322 (T -reducible syntactic type). For any syntactic type T , a collection of T -reducible syntactic types is recursively defined as follows:

1. T is a T -reducible syntactic type.
2. If τ is a T -reducible syntactic type, then τ/σ and $\tau \backslash \sigma$ are T -reducible syntactic types for any syntactic type σ .

Example 323. S , $S \backslash NP$, $S \backslash NP / NP$, and $(S \backslash NP) / (S \backslash NP)$ are examples of S -reducible syntactic types.

Let us add conjunctions with the syntactic category $CONJ_X$ to the lexicon.

Lexical Items

and $\vdash CONJ_X$
 or $\vdash CONJ_X$
 but $\vdash CONJ_X$

where X is S -reducible.

In these lexical items X can be instantiated by any S -reducible syntactic type. For example, since $S \backslash NP$ is S -reducible, the following is a lexical entry.

(68) and $\vdash CONJ_{S \backslash NP}$

This allows us to analyze coordinated structures between constituents with variety of syntactic types. For example, (69) is an example of VP's coordinated structures.

(69) John loves Mary and hates Susan.

(70) Syntactic Structure of (69)

$$\frac{\frac{\frac{\text{loves}}{S \backslash NP / NP} \quad \frac{\text{Mary}}{NP}}{S \backslash NP} > \quad \frac{\text{and}}{CONJ_{S \backslash NP}} \quad \frac{\frac{\text{hates}}{S \backslash NP / NP} \quad \frac{\text{Susan}}{NP}}{S \backslash NP} >}{\frac{\frac{\text{John}}{NP} \quad \frac{\text{hates}}{S \backslash NP / NP} \quad \frac{\text{Susan}}{NP}}{S \backslash NP / NP} <_{(\Phi)_{S \backslash NP}}} S$$

The third CCG rule is the function composition rule.

Functional Composition Rules

$$\frac{X/Y \quad Y/Z}{X/Z} >_B \quad \frac{Y \backslash Z \quad X \backslash Y}{X \backslash Z} <_B$$

This allows syntactic structures such as (71) (known as the *right-node raising* construction) to be analysed as (72).

(71) John cooked and might eat apples.

(72) Syntactic Structure of (71)

$$\frac{\frac{\frac{\text{cooked}}{S \backslash NP / NP} \quad \frac{\text{and}}{CONJ_{S \backslash NP / NP}} \quad \frac{\frac{\text{might}}{S \backslash NP / (S \backslash NP)} \quad \frac{\text{eat}}{S \backslash NP / NP}}{S \backslash NP / NP} >_B \quad \frac{\text{apples}}{NP}}{\frac{\frac{\text{John}}{NP} \quad \frac{\text{might}}{S \backslash NP / (S \backslash NP)} \quad \frac{\text{eat}}{S \backslash NP / NP} \quad \frac{\text{apples}}{NP}}{S \backslash NP} <_{(\Phi)}} S$$

Here is the list of all combinatory rules in CCG. The use of the rules other than functional application rules, coordination rule, and composition rules will be introduced later as necessary.*5

*5 We omit the presentation of modalities for syntactic types (cf. Steedman (2024)) for the sake of space, which could be integrated with \star -CCG straightforwardly.

Definition 324 (Combinatory Rules of CCG). Let X, Y, Z, W be syntactic types.

Forward Functional Application

$$\frac{X/Y \quad Y}{X} >$$

Backward Functional Application

$$\frac{Y \quad X \backslash Y}{X} <$$

Forward Composition

$$\frac{X/Y \quad Y/Z}{X/Z} >_B$$

Backward Composition

$$\frac{Y \backslash Z \quad X \backslash Y}{X \backslash Z} <_B$$

Forward Crossing Composition

$$\frac{X/Y \quad Y \backslash Z}{X \backslash Z} >_{B \times}$$

Backward Crossing Composition

$$\frac{Y/Z \quad X \backslash Y}{X/Z} <_{B \times}$$

Forward level-2 Composition

$$\frac{X/Y \quad Y/Z \mid W}{X/Z \mid W} >_{B^2}$$

Backward level-2 Composition

$$\frac{Y \backslash Z \mid W \quad X \backslash Y}{X \backslash Z \mid W} <_{B^2}$$

Forward Type Raising

$$\frac{X}{Y/(Y \backslash X)} >^T$$

Backward Type Raising

$$\frac{X}{Y \backslash (Y/X)} <^T$$

Forward Crossing Substitution

$$\frac{X/Y \backslash Z \quad Y \backslash Z}{X \backslash Z} >_{S \times}$$

Backward Crossing Substitution

$$\frac{Y/Z \quad X \backslash Y/Z}{X/Z} <_{S \times}$$

Coordination

$$\frac{X \quad CONJ \quad X}{X} \langle \Phi \rangle_X \quad \text{where } X \text{ is } S\text{-reducible.}$$

The derivation of syntactic structures presented in this subsection is rudimentary and covers only a small fraction of the countably infinite number of acceptable sentences. Still, in the discussion that follows, we assume that the evolution of CCG as a syntactic theory will give a CCG syntactic structure for every sentence.*⁶

*⁶ This is of course a non-trivial matter, but for example, in Hockenmaier and Steedman (2005), CCG syntactic structures were given to a huge number of real texts taken from the Wall Street Journal, and in Abzianidze et al. (2017), the CCG syntactic structures are given for real texts including German, Dutch and Italian, not only English. CCG parsers based on them have also been developed, achieving high parsing accuracy for real texts. These facts do not guarantee that CCGs can give appropriate syntactic structures to all sentences in natural languages, but at least they are developing as empirical rather than purely theoretical research.

15.1.1 Semantic Composition

If for every sentence, the syntactic structure can be given by a finite set of combination rules and lexical items of the CCG, then the goal of semantic theory, to provide a verification condition for every sentence, can be realised if the following two mechanisms are supplied.

1. A map $\llbracket - \rrbracket$ from a CCG syntactic structure of any sentence to its semantic representation (= a preterm of DTT)
2. A guarantee that the output of the mapping $\llbracket - \rrbracket$ for a CCG syntactic structure whose lowest level is S is a type in DTT.

The map $\llbracket - \rrbracket$ is called the *syntax-semantic mapping*. The second guarantee is called the *semantic felicity condition*, the exact definition of which will be given later.

First, for the syntax-semantic mapping $\llbracket - \rrbracket$, since it is a mapping that takes the syntactic structure of the CCG as an argument, it is possible to divide the cases according to whether the last rule used is a lexical item or a combinatorial rule. It can therefore be defined by the following two maps (this map satisfies the principle of compositionality).

- A map from lexical items in CCG to DTS preterms
- For each combinatory rule in CCG, a map from DTS preterms obtained from the premises of the rule to a DTS preterm corresponding to the conclusion.

The first map is defined by specifying the following assignments to all lexical items.

$$\begin{aligned}\llbracket \text{John} \vdash NP \rrbracket &\mapsto \textit{john} \\ \llbracket \text{loves} \vdash S \backslash NP / NP \rrbracket &\mapsto \lambda y. \lambda x. \textit{love}(x, y) \\ \llbracket \text{Mary} \vdash NP \rrbracket &\mapsto \textit{mary}\end{aligned}$$

The second map is defined as follows.*7

*7 In the standard definition of CCG, each combinatory rule is accompanied by a corresponding semantic composition rule. For example, a forward functional application rule is accompanied with the following semantic composition rule.

$$\frac{X/Y : M \quad Y : N}{X : M(N)} >$$

In words, suppose that a constituent of the syntactic type X/Y has the logical form M , and a constituent of the syntactic type Y that appears to its right has the logical form A , then the whole constituent has the syntactic type X and its logical form is $M(N)$.

Instead, we define CCG in the *functorial style*, where each combinatory rule name has its own semantic interpretation as a function, and the semantic interpretation of a constituent is uniformly defined as a result of function application between the interpretations of the rule name and the daughter constituents. The original and the functorial definition of CCG give an extensionally equivalent grammar, but the latter enables us to extend CCG to a monadic version of CCG, which we will see in Chapter 17.

Definition 325 (Semantic Composition Rule). For any combinatory rule symbol R , syntactic types X_1, \dots, X_n, Y and logical form M_1, \dots, M_n , semantic composition for combinatory rules are defined as follows.*⁸

$$\frac{X_1 : M_1 \quad \dots \quad X_n : M_n}{Y : \langle R \rangle M_1 \dots M_n} R$$

Each rule name has an interpretation as a function, and the LFs of the daughter constituents are passed to it as its arguments.*⁹

Definition 326. Semantic interpretation of CCG combinatory rule symbols are defined as follows.

$$\begin{array}{ll} \langle \rangle & \stackrel{def}{=} \lambda f. \lambda x. f x \\ \langle \rangle B & \stackrel{def}{=} \lambda f. \lambda g. \lambda x. f(gx) \\ \langle \rangle B_{\times} & \stackrel{def}{=} \lambda f. \lambda g. \lambda x. f(gx) \\ \langle \rangle B^2 & \stackrel{def}{=} \lambda f. \lambda g. \lambda x. f(gx) \\ \langle \rangle T & \stackrel{def}{=} \lambda x. \lambda p. p x \\ \langle \rangle S_{\times} & \stackrel{def}{=} \lambda f. \lambda g. \lambda x. (f x)(g x) \\ \langle \rangle \Phi_S & \stackrel{def}{=} \lambda f. \lambda c. \lambda g. ((c f) g) \\ \langle \rangle \Phi_{Y|X} & \stackrel{def}{=} \lambda f. \lambda c. \lambda g. \lambda x. (((\Phi_Y)(f x)) c)(g x) \end{array} \quad \begin{array}{ll} \langle \rangle & \stackrel{def}{=} \lambda x. \lambda f. f x \\ \langle \rangle B & \stackrel{def}{=} \lambda g. \lambda f. \lambda x. f(gx) \\ \langle \rangle B_{\times} & \stackrel{def}{=} \lambda g. \lambda f. \lambda x. f(gx) \\ \langle \rangle B^2 & \stackrel{def}{=} \lambda g. \lambda f. \lambda x. f(gx) \\ \langle \rangle T & \stackrel{def}{=} \lambda x. \lambda p. p x \\ \langle \rangle S_{\times} & \stackrel{def}{=} \lambda g. \lambda f. \lambda x. (f x)(g x) \end{array}$$

where Y is an S -reducible syntactic type.

Suppose that, in the lexicon, the semantic interpretations for *John* and *runs* are defined (in the lexicon) as follows.

- (73) $\llbracket \text{John} \vdash NP \rrbracket = \text{john}$
 (74) $\llbracket \text{runs} \vdash S \backslash NP \rrbracket = \lambda x. \text{run}(x)$

The syntax-semantics map $\llbracket - \rrbracket$ maps the syntactic structure of *John runs* to a

*⁸ This definition is instantiated to the following three rules, according to the arity of the combinatory rules.

$$\begin{array}{lll} \text{Unary rules} & \text{Binary rules} & \text{Trinary rules} \\ \frac{X : M}{Y : \langle R \rangle M} R & \frac{X : M \quad Y : N}{Z : \langle \langle R \rangle M \rangle N} R & \frac{X : M \quad Y : N \quad Z : L}{W : \langle \langle \langle R \rangle M \rangle N \rangle L} R \end{array}$$

*⁹ The functorial definition is considered as a direct manifestation of *the adjacency assumption* in Steedman (2024), stating that rules are pure functional operations.

preterm by the following process.

$$\begin{aligned}
 \left[\frac{\frac{\text{John}}{NP} \quad \frac{\text{runs}}{S \setminus NP}}{S} < \right] &\equiv \left(\langle \langle \rangle \rangle \left[\frac{\text{John}}{NP} \right] \right) \left[\frac{\text{runs}}{S \setminus NP} \right] \\
 &\equiv ((\lambda x. \lambda f. f x) \text{john})(\mathbf{run}) \\
 &\rightarrow_{\beta} (\lambda f. f(\text{john}))(\mathbf{run}) \\
 &\rightarrow_{\beta} \mathbf{run}(\text{john})
 \end{aligned}$$

This is the same calculation as the original CCG and is defined to have the same result. However, since such formula expansion is complicated, it is abbreviated by the following tree structure.

$$\frac{\frac{\text{John}}{\text{john}} \quad \frac{\text{runs}}{\mathbf{run}}}{\mathbf{run}(\text{john})} <$$

15.2 Semantic Felicity Condition

Now, the result of the calculations so far is that $\mathbf{run}(\text{john})$ is the semantic representation of *John runs* under DTS. Now, if it is ensured that this is a type, then we have a verification condition, and we have assigned a meaning to the sentence *John runs*.

Therefore, the DTS requires that syntax-semantics map $\llbracket - \rrbracket$ satisfies the requirement of the *semantic felicity condition* (SFC). In preparation for defining the notion of SFC, we first define a map called type correspondence, $[-]$. This is a map from the syntactic category of CCG to the type of DTT.

Each syntactic type corresponds to a semantic type via the following map. Here, we assume that DTS is a semantic component of the grammar, so semantic types are preterms of DTT (recall that the logical form must obey SFC, which requires that the LF of a sentence must be of the sort **type**).

Definition 327 (Type correspondence). Type correspondence $[-]$ from syntactic types of CCG to types of DTT is recursively defined as follows.

$[S]$	$\stackrel{def}{\equiv}$	type	$[X/Y]$	$\stackrel{def}{\equiv}$	$[Y] \rightarrow [X]$
$[NP]$	$\stackrel{def}{\equiv}$	entity	$[X \setminus Y]$	$\stackrel{def}{\equiv}$	$[Y] \rightarrow [X]$
$[N]$	$\stackrel{def}{\equiv}$	entity \rightarrow type			
$[PP]$	$\stackrel{def}{\equiv}$	entity			
$[CONJ]$	$\stackrel{def}{\equiv}$	type \rightarrow type \rightarrow type			
$[CP]$	$\stackrel{def}{\equiv}$	type			
$[RN]$	$\stackrel{def}{\equiv}$	entity \rightarrow entity			

*9 This is replaced with UDTT in the part 3, where we discuss anaphora and presuppositions.

The semantic felicity condition for the constituent $\overset{\mathcal{D}}{X}$ (the syntactic structure \mathcal{D} whose lowest syntactic type is X) is stated as the following condition.

Definition 328 (Semantic Felicity Condition (SFC)). For any CCG syntactic structure \mathcal{D} such that the bottom syntactic type is X , the syntactic structure $\overset{\mathcal{D}}{X}$ satisfies the semantic felicity condition, or is semantically felicitous, if and only if there exists a context Γ (but $\Gamma \not\vdash \perp$) that satisfies the following DTT judgement.

$$\Gamma \vdash \left[\begin{array}{c} \mathcal{D} \\ X \end{array} \right] : [X]$$

It is ensured that a syntactic structure with syntactic type S is a type of DTT by the map $\llbracket - \rrbracket$ (namely, has meaning) via the following theorem.

Theorem 329. If all lexical items satisfy the semantic felicity condition, then all syntactic structures of the CCG are semantically felicitous.

Proof. By induction on the structure of \mathcal{D} : since the base case is a lexical item, it follows from the assumption for the base case. The cases are divided by combinatory rules as follows. First, consider the forward function application rule.

$$\begin{aligned} \left[\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ X/Y & Y \\ \hline X \end{array} > \right] &\equiv ((\triangleright) \llbracket \mathcal{D}_1 \rrbracket) \llbracket \mathcal{D}_2 \rrbracket \\ &\equiv ((\lambda f. \lambda x. f x) \llbracket \mathcal{D}_1 \rrbracket) \llbracket \mathcal{D}_2 \rrbracket \end{aligned}$$

As the induction hypothesis (IH), there exists Γ, Γ' that satisfies the following.

$$\begin{aligned} \Gamma &\vdash \llbracket \mathcal{D}_1 \rrbracket : [Y] \rightarrow [X] \\ \Gamma' &\vdash \llbracket \mathcal{D}_2 \rrbracket : [Y] \end{aligned}$$

Let it be $\Gamma'' = \Gamma \cup \Gamma'$, the following also satisfy.

$$\begin{aligned} \Gamma'' &\vdash \llbracket \mathcal{D}_1 \rrbracket : [Y] \rightarrow [X] \\ \Gamma'' &\vdash \llbracket \mathcal{D}_2 \rrbracket : [Y] \end{aligned}$$

Then, we obtain

$$\begin{array}{c}
\frac{\frac{f : [Y] \rightarrow [X]^1 \quad x : [Y]^2}{fx : [X]}_{(\Pi E)} \quad \frac{\lambda x. fx : [Y] \rightarrow [X]}{\lambda f. \lambda x. fx : ([Y] \rightarrow [X]) \rightarrow [Y] \rightarrow [X]}_{(\Pi I), 2} \quad \left[\frac{\mathcal{D}_1}{X/Y} \right] : [Y] \rightarrow [X]}{\lambda f. \lambda x. fx : ([Y] \rightarrow [X]) \rightarrow [Y] \rightarrow [X]}_{(\Pi I), 1} \quad \left[\frac{\mathcal{D}_1}{X/Y} \right] : [Y] \rightarrow [X]}_{(\Pi E)} \\
\frac{(\lambda f. \lambda x. fx) \left[\frac{\mathcal{D}_1}{X/Y} \right] : [Y] \rightarrow [X] \quad \left[\frac{\mathcal{D}_2}{Y} \right] : [Y]}{(\lambda f. \lambda x. fx) \left[\frac{\mathcal{D}_1}{X/Y} \right] \left[\frac{\mathcal{D}_2}{Y} \right] : [X]}_{(\Pi E)}
\end{array}$$

thus, the subject is valid. The same applies to the other combinatory rules. \square

Exercise 330. Complete the proof for the case of other combinatory rules.

Remark 331. It can be checked individually whether a lexical item satisfies the semantic felicity condition. For example, in the case of *John*, *loves*, and *Mary*, it is sufficient to show that there exists a Γ that satisfies the following.

$$\begin{aligned}
\Gamma &\vdash \textit{john} : \text{entity} \\
\Gamma &\vdash \lambda y. \lambda x. \text{love}(x, y) : \text{entity} \rightarrow \text{entity} \rightarrow \text{type} \\
\Gamma &\vdash \textit{mary} : \text{entity}
\end{aligned}$$

This is obvious if the signature contains $\text{love} : \text{entity} \times \text{entity} \rightarrow \text{type}$.

Remark 332. There is room to change the definition of Definition ?? for each theory of meaning the readers want to adopt. Importantly, the proof of Lemma 329 is not affected by this change. However, this does not apply whether or not lexical items satisfy SFC, so we have to make sure that all lexical items satisfy the semantic felicity condition when making the change.

When all lexical items satisfy SFC, then all syntactic structures satisfy SFC by Lemma 329. Therefore, the following holds.

Corollary 333. In a theory of meaning in which all lexical items are semantically felicitous, syntactic structures whose syntactic category is S are typed by the syntax-semantics map $\llbracket - \rrbracket$ under the appropriate context Γ .

This also demonstrates the following, by which the requirement made at the beginning of this chapter is also fulfilled.

Corollary 334. In the DTS, given a lexicon each lexical item of which satisfies SFC, a verification condition is given for each sentence whose syntactic type is S in CCG.

15.3 Alternative Notation

It is cumbersome to describe the process of semantic synthesis every time as follows.

$$\begin{aligned}
\left[\frac{\frac{\frac{\text{John}}{NP} \quad \frac{\frac{\text{loves}}{S \setminus NP / NP} \quad \frac{\text{Mary}}{NP}}{S \setminus NP} >}{S} < \right] &= \left[\frac{\frac{\text{loves}}{S \setminus NP / NP} \quad \frac{\text{Mary}}{NP}}{S \setminus NP} > \right] \left(\left[\frac{\text{John}}{NP} \right] \right) \\
&= \left(\left[\frac{\text{loves}}{S \setminus NP / NP} \right] \left(\left[\frac{\text{Mary}}{NP} \right] \right) \right) \left(\left[\frac{\text{John}}{NP} \right] \right) \\
&= (\lambda y. \lambda x. \mathbf{love}(x, y))(mary)(john) \\
&\rightarrow_{\beta} \mathbf{love}(john, mary)
\end{aligned}$$

An alternative notation, which is closer to the standard notation in formal semantics, describes semantic composition in line with syntactic structures.

$$\frac{\frac{\text{John}}{john} \quad \frac{\frac{\text{loves}}{\lambda y. \lambda x. \mathbf{love}(x, y)} \quad \frac{\text{Mary}}{mary}}{(\lambda y. \lambda x. \mathbf{love}(x, y))(mary)} >}{(\lambda y. \lambda x. \mathbf{love}(x, y))(mary)(john)} <$$

Furthermore, if we allow preterms to be β -reduced at each step of the syntactic structure, we obtain the following notation. Note that this operation is allowed by Subject Reduction theorem.

$$\frac{\frac{\text{John}}{john} \quad \frac{\frac{\text{loves}}{\lambda y. \lambda x. \mathbf{love}(x, y)} \quad \frac{\text{Mary}}{mary}}{\lambda x. \mathbf{love}(x, mary)} >}{\mathbf{love}(john, mary)} <$$

This notation does not affect the final semantic representation obtained in type systems with Church=Rosser property, i.e. type systems that arrive at the same normal form irrespective of the simplification path. This notation will be used in semantic composition in the subsequent chapters, bearing in mind that the principles described in the previous sections are at work in the background.

History and Further Reading

CCG as a syntactic theory

Categorial grammar の事始めは、Ajdukiewicz (1935) が数式の syntax を表す仕組みとして提案したものに始まる。その後、1950 年代に Chomsky hierarchy の研究が行われるなかで、Bar-Hillel (1953), Lambek (1958); ? によって (classical な) categorial grammar が context-free grammar と同等であることが示された。

categorial grammar にとっての一つの転機となったのは、Montague が Montague (1973) において Montague Grammar の統語部門として categorial grammar の 1 バージョンを採用したことである。しかし、その後の生成文法の急速な発展の中で、経験的な統語論の舞台は生成文法となり、categorial grammar はどちらかというと、logician の戯れのように捉えられていた時代が続いた。

1980 年代に、GPSG, HPSG といった統語理論を研究していたグループが、その頃 Lambek 計算に着目し、ふたたび categorial grammar の研究が、生成文法全盛期の水面下で発展することとなった。その当時の研究は 1988 年の *Categorial Grammars and Natural Language Structures* などの書籍において伺うことができるが、この時期に重要な知見が多数積み上げられたといえる。

CCG は、これらの categorial grammar の一つとして classical categorial grammar から派生し、Ades and Steedman (1982) を皮切りに、Steedman (1988, 1990, 1996, 2000) と、1980～90 年代に大きな発展を迎えた。

CCG parsers

ツリーバンクからパーザを生成する、という工程は 90 年代の確率的 CFG パーザの時代に確立されたものであるが、CCGbank (Hockenmaier and Steedman, 2005) の登場、それに続く C&C parser (Clark and Curran, 2007), EasyCCG (Lewis and Steedman, 2014), depccg (Yoshikawa et al., 2017) の登場は、形式統語論の理論である CCG のパーザが同様の工程において生成可能である、ということを印象付けた。

この流れは日本語の統語解析研究にも影響を与えた。おりしも、Bekki (2010) によって、CCG によって日本語の統語構造の網羅的な記述が可能であることが示されたことをうけて、Uematsu+ (2013) による日本語 CCGbank 開発が行われ、それに続いて Jigg Noji and Miyao (2016), depccg Yoshikawa et al. (2017) といった日本語 CCG パーザが開発されるに至った。

ところが、CCGbank の構築は、たとえば係り受け木によるツリーバンクと比較すると高価である。理由は、係り受け木と比較して、個々の木の持つ情報が多いこと、CCG 構文木の妥当性を判断できる人材が限られていることが挙げられる。そのため、コーパスは一から作り上げるよりは、既存のツリーバンクから自動変換によって生成する、という方法を採用ざるを得ない。

たとえば日本語 CCG ツリーバンク (Uematsu+ 2013) であれば、係り受け構造で記述された京都大学テキストコーパス^{*10}、述語項構造や照応関係の情報が付与された NAIST コーパス^{*11}、助詞「と」を含む文の項と述語の関係の情報を付与した「と」コーパス (hanaoka+ 2012) から得られた統語情報を統合的に利用して構築されている。ところが、CCG 木は係り受け木が含まない情報 (項構造や統語素性など) を多く含むため、自動変換には豊潤な言語学的知識が含まれていなければならない。しかし、自然言語処理においては統語解析の専門家といえど言語学の統語理論の知識を持つわけではない。結果として、上述の自動変換は、ad-hoc な規則の集合体によって行われている。

しかしながら、英語であれ日本語であれ、CCGbank は CCG パーザの訓練データでもあり評価データでもある。したがって、CCGbank の文法記述が CCG パーザの性能の上限を与える。言いかえれば、CCGbank の誤りは CCG パーザに引き継がれる。その意味では、CCGbank の質が重要になってくるが、ツリーバンクから ad-hoc な自動変換規則によって得られた統語構造が、形式統語論の研究者の立場からみてどの程度妥当なのか、という研究は十分になされているとはいえない。

^{*10} <https://github.com/ku-nlp/KyotoCorpus>

^{*11} <https://sites.google.com/site/naisttextcorpus/>