Chapter 15

Syntax-Semantics Transparency

The preceding chapter illuminated the fundamental challenge posed by the finitude of human knowledge, thereby stipulating three axiomatic desiderata for any comprehensive theory of natural language meaning. This chapter undertakes the task of demonstrating how Dependent Type Semantics (DTS), as our chosen framework for meaning, rigorously satisfies these prerequisites.

The first such desideratum mandates a syntactic theory for natural languages comprising a finite set of generative rules. In this treatise, we adopt Combinatory Categorial Grammar (CCG) (Ades and Steedman, 1982; Steedman, 1988, 1997, 2000) as our foundational syntactic framework[*1], the intricacies of which will be elucidated in the subsequent section. The elegance and computational tractability of CCG render it eminently suitable for this purpose.

The second requirement posits a foundational hypothesis concerning the nature of sentential meaning, coupled with a representational theory congruent with this hypothesis. As adumbrated at the close of the previous chapter, this work operates under the hypothesis that the meaning of a sentence inheres in its verification condition. Consequently, the meaning of a sentence is formally represented as a type within the framework of Dependent Type Theory (DTT). This choice is not arbitrary; it grounds meaning in the constructive and inferential power inherent in types.

The third and final desideratum demands a systematic method for computing the meaning of a complex whole from the meanings of its constituent parts, in correspondence with the structural definitions provided by the chosen syntactic theory. Given our commitment to CCG as the syntactic engine, this translates into demonstrating, for each combinatory rule of CCG, the precise mechanism by which the meaning of a mother node is derived from the meanings of its daughter nodes.

Crucially, a supplementary mechanism is then necessitated to ensure that the resultant semantic representation of any given sentence is, in fact, a well-formed type within DTT. This vital safeguard is introduced in the present chapter as the con-

[*1] CCG stands as a notable exemplar among syntactic theories amenable to semantic interpretation. Yet, its adoptability does not presuppose its indispensability for the construction of a theory of meaning. Indeed, the landscape of categorial grammars is rich and varied. Should a reader find themselves committed to an alternative categorial framework, they are at liberty, following the methodology herein presented, to graft a suitable method of semantic representation, via DTS, onto the syntactic structures posited by their chosen theory.

cept of a *semantic felicity condition*. It serves as a formal filter, guaranteeing the well-typedness and coherence of the semantic output.

By meticulously addressing these three requirements, this chapter aims to establish the robust explanatory power and formal precision of DTS as a theory of natural language meaning. This exploration will illuminate not only the mechanics of meaning composition but also the profound link between syntax and semantics forged through the type-theoretic lens.

15.1 Combinatory Categorial Grammar

CCG is one of the syntactic theories called *categorial grammars*. Categorial 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 322. $S, NP, N, PP, CONJ, CP, RN$ 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 323. 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

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

Seen as logical rules, X, Y are syntactic types, and these rules correspond to 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).

elimination rules for right and left implication in substructural logic.[*3]Regarding the 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)

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

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.

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

[*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.

[*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.

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

(67) Syntactic structure of (66)

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

The second CCG combination rule is the following coordination rule.

Coordination Rule

$$\dfrac{X \quad CONJ_X \quad X}{X}\,\langle \Phi \rangle_X$$

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 324 (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 325. 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 type $CONJ_X$ to the lexicon.

Lexical Items

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

where X is S-reducible.

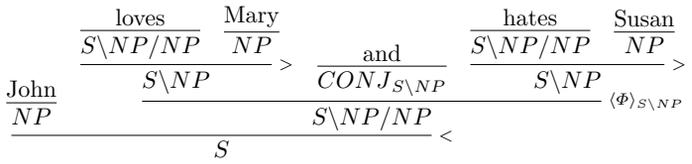
In these lexical items X can be instantiated by any S-reducible syntactic type. For example, since $S\backslash NP$ is S-reducible, the following is a lexical entry.

(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)



The third CCG rule is the function composition rule.

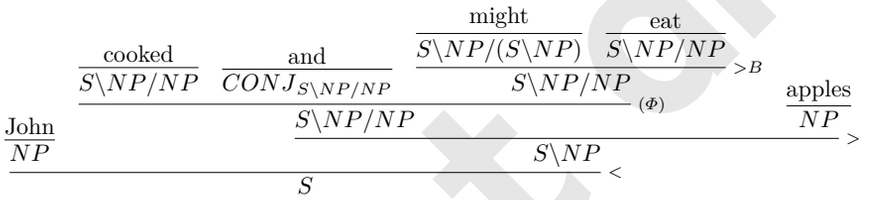
Functional Composition Rules

$$\frac{X/Y \quad Y/Z}{X/Z}\ {>}\mathbf{B} \qquad \frac{Y\backslash Z \quad X\backslash Y}{X\backslash Z}\ {<}\mathbf{B}$$

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)



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 ⋆-CCG straightforwardly.

Definition 326 (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

$$\frac{X/Y \quad Y/Z\,|\,W}{X/Z\,|\,W} >B^2$$

Backward level-2 Composition

$$\frac{Y\backslash Z\,|\,W \quad X\backslash Y}{X\backslash Z\,|\,W} <B^2$$

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 $[\![-]\!]$ 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 $[\![-]\!]$ for a CCG syntactic structure whose lowest level is S is a type in DTT.

The map $[\![-]\!]$ 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 $[\![-]\!]$, 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.

$$[\![\text{John} \vdash NP]\!] \mapsto john$$
$$[\![\text{loves} \vdash S\backslash NP/NP]\!] \mapsto \lambda y.\lambda x.\mathbf{love}(x, y)$$
$$[\![\text{Mary} \vdash NP]\!] \mapsto mary$$

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

$$\frac{X_1 : M_1 \quad \cdots \quad X_n : M_n}{Y : (\!|R|\!)\, M_1 \cdots M_n}\ R$$

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

Definition 328. Semantic interpretation of CCG combinatory rule symbols are defined as follows.

$$\begin{array}{rcl@{\qquad}rcl}
(\!|{>}|\!) & \stackrel{def}{\equiv} & \lambda f.\lambda x.fx & (\!|{<}|\!) & \stackrel{def}{\equiv} & \lambda x.\lambda f.fx \\
(\!|{>}B|\!) & \stackrel{def}{\equiv} & \lambda f.\lambda g.\lambda x.f(gx) & (\!|{<}B|\!) & \stackrel{def}{\equiv} & \lambda g.\lambda f.\lambda x.f(gx) \\
(\!|{>}B_\times|\!) & \stackrel{def}{\equiv} & \lambda f.\lambda g.\lambda x.f(gx) & (\!|{<}B_\times|\!) & \stackrel{def}{\equiv} & \lambda g.\lambda f.\lambda x.f(gx) \\
(\!|{>}B^2|\!) & \stackrel{def}{\equiv} & \lambda f.\lambda g.\lambda x.f(gx) & (\!|{<}B^2|\!) & \stackrel{def}{\equiv} & \lambda g.\lambda f.\lambda x.f(gx) \\
(\!|{>}T|\!) & \stackrel{def}{\equiv} & \lambda x.\lambda p.px & (\!|{<}T|\!) & \stackrel{def}{\equiv} & \lambda x.\lambda p.px \\
(\!|{>}S_\times|\!) & \stackrel{def}{\equiv} & \lambda f.\lambda g.\lambda x.(fx)(gx) & (\!|{<}S_\times|\!) & \stackrel{def}{\equiv} & \lambda g.\lambda f.\lambda x.(fx)(gx) \\
(\!|\langle\Phi_S\rangle|\!) & \stackrel{def}{\equiv} & \lambda f.\lambda c.\lambda g.((cf)g) & & & \\
(\!|\langle\Phi_{Y|X}\rangle|\!) & \stackrel{def}{\equiv} & \lambda f.\lambda c.\lambda g.\lambda x.((\langle\Phi_Y\rangle(fx))c)(gx) & & & \\
 & & \text{where } Y \text{ is an } S\text{-reducible syntactic type.} & & &
\end{array}$$

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

(73) $[\![\texttt{John} \vdash NP]\!] = john$

(74) $[\![\texttt{runs} \vdash S\backslash NP]\!] = \lambda x.\mathbf{run}(x)$

The syntax-semantics map $[\![-]\!]$ 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.

Unary rules	Binary rules	Trinary rules
$\dfrac{X : M}{Y : (\!|R|\!)\, M}\ R$	$\dfrac{X : M \quad Y : N}{Z : ((\!|R|\!)\, M)\, N}\ R$	$\dfrac{X : M \quad Y : N \quad Z : L}{W : (((\!|R|\!)\, M)\, N)\, L}\ R$

[*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.

$$\left[\!\!\left[\dfrac{\dfrac{\text{John}}{NP} \quad \dfrac{\text{runs}}{S\backslash NP}}{S}< \right]\!\!\right] \equiv \left(\Vdash\!\!< \right) \left[\!\!\left[\dfrac{\text{John}}{NP} \right]\!\!\right] \left[\!\!\left[\dfrac{\text{runs}}{S\backslash NP} \right]\!\!\right]$$
$$\equiv ((\lambda x.\lambda f.fx)\ john)(\mathbf{run})$$
$$\rightarrow_\beta (\lambda f.f(john))(\mathbf{run})$$
$$\rightarrow_\beta \mathbf{run}(john)$$

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.

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

15.2 Semantic Felicity Condition

Now, the result of the calculations so far is that $\mathbf{run}(john)$ 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 $[\![-]\!]$ 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, $\lceil - \rceil$. This is a map from the syntactic type 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 329 (Type correspondence)**.** Type correspondence $\lceil - \rceil$ from syntactic types of CCG to types of DTT is recursively defined as follows.

$$\lceil S \rceil \stackrel{def}{\equiv} \mathsf{type} \qquad \lceil X/Y \rceil \stackrel{def}{\equiv} \lceil Y \rceil \rightarrow \lceil X \rceil$$
$$\lceil NP \rceil \stackrel{def}{\equiv} \mathsf{entity} \qquad \lceil X\backslash Y \rceil \stackrel{def}{\equiv} \lceil Y \rceil \rightarrow \lceil X \rceil$$
$$\lceil N \rceil \stackrel{def}{\equiv} \mathsf{entity} \rightarrow \mathsf{type}$$
$$\lceil PP \rceil \stackrel{def}{\equiv} \mathsf{entity}$$
$$\lceil CONJ \rceil \stackrel{def}{\equiv} \mathsf{type} \rightarrow \mathsf{type} \rightarrow \mathsf{type}$$
$$\lceil CP \rceil \stackrel{def}{\equiv} \mathsf{type}$$
$$\lceil RN \rceil \stackrel{def}{\equiv} \mathsf{entity} \rightarrow \mathsf{entity}$$

[*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 330 (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 $\Gamma \not\vdash \bot$) that satisfies the following DTT judgement.

$$\Gamma \vdash \left[\!\!\left[\overset{\mathcal{D}}{X} \right]\!\!\right] : \lceil X \rceil$$

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

Theorem 331. 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.

$$\begin{aligned}\left[\!\!\left[\dfrac{\begin{matrix}\mathcal{D}_1 & \mathcal{D}_2 \\ X/Y & Y\end{matrix}}{X}\, > \right]\!\!\right] &\equiv (\!|>|\!)\, [\![\mathcal{D}_1]\!]\, [\![\mathcal{D}_2]\!] \\ &\equiv ((\lambda f.\lambda x.fx)\, [\![\mathcal{D}_1]\!])\, [\![\mathcal{D}_2]\!]\end{aligned}$$

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

$$\begin{aligned}\Gamma &\vdash [\![\mathcal{D}_1]\!] : \lceil Y \rceil \to \lceil X \rceil \\ \Gamma' &\vdash [\![\mathcal{D}_2]\!] : \lceil Y \rceil\end{aligned}$$

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

$$\begin{aligned}\Gamma'' &\vdash [\![\mathcal{D}_1]\!] : \lceil Y \rceil \to \lceil X \rceil \\ \Gamma'' &\vdash [\![\mathcal{D}_2]\!] : \lceil Y \rceil\end{aligned}$$

Then, we obtain

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{f : \lceil Y \rceil \to \lceil X \rceil}^{1} \quad \overline{x : \lceil Y \rceil}^{2}}{fx : \lceil X \rceil}(\Pi E)}{\lambda x.fx : \lceil Y \rceil \to \lceil X \rceil}(\Pi I),_2}{\lambda f.\lambda x.fx : (\lceil Y \rceil \to \lceil X \rceil) \to \lceil Y \rceil \to \lceil X \rceil}(\Pi I),_1 \quad \dfrac{}{\left[\!\left[\begin{smallmatrix} \mathcal{D}_1 \\ X/Y \end{smallmatrix} \right]\!\right] : \lceil Y \rceil \to \lceil X \rceil} IH}{(\lambda f.\lambda x.fx)\left[\!\left[\begin{smallmatrix} \mathcal{D}_1 \\ X/Y \end{smallmatrix} \right]\!\right] : \lceil Y \rceil \to \lceil X \rceil}(\Pi E) \quad \dfrac{}{\left[\!\left[\begin{smallmatrix} \mathcal{D}_2 \\ Y \end{smallmatrix} \right]\!\right] : \lceil Y \rceil} IH}{(\lambda f.\lambda x.fx)\left[\!\left[\begin{smallmatrix} \mathcal{D}_1 \\ X/Y \end{smallmatrix} \right]\!\right]\left[\!\left[\begin{smallmatrix} \mathcal{D}_2 \\ Y \end{smallmatrix} \right]\!\right] : \lceil X \rceil}(\Pi E)$$

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

Exercise 332. Complete the proof for the case of other combinatory rules.

Remark 333. 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.

$$\Gamma \vdash \mathit{john} : \mathbf{entity}$$
$$\Gamma \vdash \lambda y.\lambda x.\mathbf{love}(x, y) : \mathbf{entity} \to \mathbf{entity} \to \mathbf{type}$$
$$\Gamma \vdash \mathit{mary} : \mathbf{entity}$$

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

Remark 334. There is room to change the definition of Definition **??** for each theory of meaning the readers want to adopt. Importantly, the proof of Lemma 331 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 331. Therefore, the following holds.

Corollary 335. In a theory of meaning in which all lexical items are semantically felicitous, syntactic structures whose syntactic type is S are of type **type** by the syntax-semantics map $[\![-]\!]$ 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 336. 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 composition every time as follows.

$$\left[\!\!\left[\dfrac{\dfrac{\text{John}}{NP} \quad \dfrac{\dfrac{\text{loves}}{S\backslash NP/NP} \quad \dfrac{\text{Mary}}{NP}}{S\backslash NP}>}{S}< \right]\!\!\right] = \left[\!\!\left[\dfrac{\dfrac{\text{loves}}{S\backslash NP/NP} \quad \dfrac{\text{Mary}}{NP}}{S\backslash NP}> \right]\!\!\right] \left(\left[\!\!\left[\dfrac{\text{John}}{NP} \right]\!\!\right] \right)$$

$$= \left(\left[\!\!\left[\dfrac{\text{loves}}{S\backslash NP/NP} \right]\!\!\right] \left(\left[\!\!\left[\dfrac{\text{Mary}}{NP} \right]\!\!\right] \right) \right) \left(\left[\!\!\left[\dfrac{\text{John}}{NP} \right]\!\!\right] \right)$$

$$= (\lambda y.\lambda x.\mathbf{love}(x,y))(mary)(john)$$

$$\twoheadrightarrow_\beta \mathbf{love}(john, mary)$$

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

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

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.

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

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 reduction 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

The genesis of Categorial Grammar (CG) lies in the pioneering work of Ajdukiewicz (1935), who conceived the syntax of mathematical formulas as a formal system. This foundational insight laid the groundwork for subsequent investigations into its computational properties.

In the mid-twentieth century, a period marked by intense scrutiny of formal language theory and the emergence of the Chomsky hierarchy, the intrinsic power of classical categorial grammar was rigorously established. Bar-Hillel (1953), Lambek (1958); **?** demonstrated its equivalence to Context-Free Grammar, thereby situating CG within the prevailing landscape of formal language models.

A pivotal moment in the intellectual history of categorial grammar arrived with Montague (1973), who, in his seminal contributions to formal semantics, adopted a variant of CG as the syntactic component of Montague Grammar. This adoption lent considerable prestige to the framework, yet its trajectory diverged from the mainstream. As generative grammar ascended, swiftly claiming the empirical domain of syntactic inquiry, categorial grammar largely receded into the province of logicians, often perceived more as a theoretical amusement than a practical tool for linguistic analysis.

Nonetheless, the 1980s witnessed a subterranean resurgence of interest in categorial grammar. Researchers immersed in alternative syntactic frameworks, such as GPSG and HPSG, recognised the expressive potential of Lambek's calculus, sparking a renewed period of inquiry. This quiet efflorescence, occurring amidst the dominance of generative grammar, culminated in significant advancements. The fruits of this era are testament to the accumulation of numerous critical insights, comprehensively documented in works such as "Categorial Grammars and Natural Language Structures" in 1988.

From this rich lineage, CCG emerged as a direct descendant of classical categorial grammar. Initiated by Ades and Steedman (1982), and subsequently refined and expanded through the influential contributions of Steedman (1988, 1990, 1996, 2000), CCG underwent a profound period of development during the 1980s and 1990s, solidifying its position as a robust and empirically rich theory of natural language syntax.

CCG Treebanks and Parsers

The automated generation of parsers from treebanks, a methodology firmly established in the 1990s with the advent of Probabilistic Context-Free Grammar (PCFG) parsers, found a compelling parallel in the realm of CCG. The seminal release of CCGbank (Hockenmaier and Steedman, 2005), swiftly followed by the development of sophisticated parsers such as the C&C parser (Clark and Curran, 2007), EasyCCG (Lewis and Steedman, 2014), and depccg (Yoshikawa et al., 2017), unequivocally demonstrated that parsers predicated on CCG, a formal syntactic theory of considerable elegance, could indeed be cultivated through an analogous computational pipeline.

This influential trajectory was not confined to English; its resonance extended to the intricate domain of Japanese syntactic parsing. Concurrently, the robust and formal exposition by Bekki (2010) evinced the remarkable capacity of CCG to describe the syntactic architecture of Japanese. This pivotal work catalyzed the creation of the Japanese CCGbank by Uematsu et al. (2013), which subsequently paved the way for further advancements, including the Jigg by Noji and Miyao (2016) and additional Japanese CCG parsers.

Notwithstanding these advancements, the construction of a CCGbank inherently entails a significantly greater cost compared to, for instance, a dependency treebank. This elevated expenditure stems from two primary factors: the inherently richer informational content encoded within individual CCG trees relative to dependency trees, and the scarcity of human expertise capable of reliably validating the intricate judgments demanded by CCG syntactic structures. Consequently, the prevailing approach has eschewed de novo corpus creation in favour of automatic conversion from existing treebanks.

Consider, for illustration, the Japanese CCG treebank (Uematsu et al., 2013). Its derivation is a testament to the integrated utilization of diverse linguistic resources: the Kyoto University Text Corpus[*10], meticulously annotated for dependency structures and collocational relations; the NAIST corpus[*11], replete with predicate-argument structures; and the "and" corpus by Hanaoka et al. (2012), which meticulously delineates term-predicate relationships in sentences featuring the particle "and." Yet, the transition from such resources to CCG trees is far from trivial. CCG trees encapsulate a wealth of information—including argument structure and fine-grained syntactic features—that often lies beyond the scope of mere dependency annotations. Thus, automatic conversion necessitates the judicious incorporation of substantial linguistic knowledge. Regrettably, within the domain of natural language processing, even specialists in syntactic parsing may lack a deep grounding in formal syntactic theory. This deficit often leads to automatic conversion being executed via a collection of largely ad-hoc rules, as pointed out and discussed in Bekki and Yanaka (2023) and Tomita et al. (2024).

[*10] https://github.com/ku-nlp/KyotoCorpus
[*11] https://sites.google.com/site/naisttextcorpus/