Chapter 1

The Logical Approach in Linguistics

The framework of categorial type logic (CTL) [Moo97] developed out of earlier work in the tradition of categorial grammar. In this chapter, we briefly present these ancestral lines of research, and we give the reader an idea of the kind of problems that have led to the introduction of CTL. Readers familiar with the categorial approach to natural language syntax and semantics can skip this chapter and go directly to Chapter 2.

The present chapter is organized as follows. We start by introducing classical and combinatory categorial grammars, two formalisms closely related to CTL (Section 1.1). Then, by highlighting the differences between these frameworks and the logical approach assumed in this thesis, we introduce the main aspects of CTL (Section 1.2). Moreover, we discuss the proof theoretical perspective on form-meaning assembly of linguistic expressions.

1.1 Rule-Based Categorial Grammars

The categorial tradition of natural language analysis goes back to the pioneering works of Lesniewski [Les29] and Ajdukiewicz [Ajd35]. The ingredients of a categorial grammar are extremely simple: a system of syntactic categories (or types), and a set of rules to compute with these types. The categories are either atomic, or they are structured as 'fractions' $\frac{a}{b}$. Atomic types categorize expressions that in some intuitive sense are 'complete'; incomplete expressions are assigned a fractional category. The basic combinatory rule schema takes the form of a kind of 'multiplication': from $\frac{a}{b} \times b$ one obtains the category a. The algebraic nature of the schemata for category combination was emphasized by Bar-Hillel in [BH53].

In the section, we discuss two categorial frameworks: the classical categorial grammars of Ajdukiewicz and Bar-Hillel (CG, also known as AB grammars), and the combinatory categorial grammars of Steedman (CCG, [Ste00]). These frameworks have the same category concept, but they have different sets of rule schemata for category combination: the CCG rule set extends the schemata of CG in order to overcome certain expressive limitations of the classical categorial approach.

1.1.1 Classical Categorial Grammar

The type language and the rules of classical Categorial Grammar (CG) are defined as below.

Definition 1.1. [Type Language and Rules of CG] The language of CG is recursively built over atomic categories by means of the category forming operators \ and /. The combinatorial behavior of categories is captured by the left/right application rules.

CG language. Given a set of basic categories ATOM, the set of categories CAT is the smallest set such that:

i. if $A \in \mathsf{ATOM}$, then $A \in \mathsf{CAT}$;

ii. if A and $B \in \mathsf{CAT}$, then A/B and $B\backslash A \in \mathsf{CAT}$.

There are two schemata for category combination, *backward application* (BA) and *forward application* (FA) *CG rules.*

$$A/B, B \Rightarrow A \quad [\mathtt{FA}]$$
$$B, B\backslash A \Rightarrow A \quad [\mathtt{BA}].$$

[FA] (resp. [BA]) says that when an expression of category A/B (resp. $B\backslash A$) is concatenated with an expression of category B on its right (resp. on its left), it yields a structure of category A.

To facilitate the comparison between CG and the categorial systems developed by Jim Lambek (Section 1.2), we present CG as a deductive system (cf. Buszkowski [Bus97]). Below we define the *derives* relation, holding between a finite sequence of categories Γ and a category A.

Definition 1.2. [Derivability Relation] Let $\Rightarrow$ be the *derivability* relation between a finite non-empty sequence of categories Γ and a category B ($\Gamma \Rightarrow B$), fulfilling the following conditions:

$$A \Rightarrow A \qquad [\text{id}]$$
$$\Gamma, A, \Gamma' \Rightarrow B \text{ and } \Delta \Rightarrow A, \text{ then } \Gamma, \Delta, \Gamma' \Rightarrow B. \qquad [\text{cut}]$$

In CG $\Rightarrow$ is the smallest relation containing the logical axioms [id], the application rules [BA] and [FA] as non-logical axioms, and it is closed under [cut].

To obtain a grammar G, we add a lexicon to the deductive part. Let Σ be the terminal alphabet, *i.e.* the set of basic natural language expressions. The lexicon LEX assigns a finite number of types to the elements of Σ, *i.e.* $\mathsf{LEX} \subseteq \Sigma \times \mathsf{CAT}$. We say that G generates a string $w_1 \ldots w_n \in \Sigma^+$ as an expression of category B if and only if there are categories $A_1, \ldots, A_n$ such that $(w_i, A_i) \in \mathsf{LEX}$ and $A_1, \ldots, A_n \Rightarrow B$. $L(G)$, the language of G, is the set of strings generated by G for some designated category, the start symbol of G.

It was shown in [BGS60] that CG has the weak generative capacity of Context Free Grammar (CFG). But conceptually, CG already improves on CFG. The structured category format allows one to replace a stipulated set of rewrite rules by two simple combinatory schemata. In phrase structure grammar, this categorial idea later resurfaced in the form of the X-Bar Theory [Jac77].

In order to get a feeling for the kind of phenomena that can be handled by CG, and for the limitations of this framework, we introduce an extremely elementary fragment of English in Example 1.3. We will use the phrases given there as a checklist throughout this chapter, and come back to them later to see how the descendants of CG improve on the original framework.

Example 1.3. [English Toy Fragment] The fragment contains simple declarative sentences, with intransitive or transitive verbs; proper names and full noun phrases introduced by determiners; nominal and adverbial modifiers; relative clauses with subject and object relativization.

(1) a. Lori left.
b. Lori knows Sara.
c. Sara wears the new dress.

(2) a. The student left.
b. Some student left.

(3) a. No student left yet.
b. Some student left already.

(4) a. who knows Lori.
b. which Sara wrote.
c. which Sara wrote there.

(5) a. Every student knows one book.
b. Every student knows some book.
c. No student knows any book.

Let us see whether we can come up with a CG that generates the phrases of our toy fragment.

Example 1.4. [CG Grammar for the Toy Fragment] Let ATOM be $\{n, s, np\}$ (for common nouns, sentences and names, respectively) and LEX as given below:

Lori, Sara	np	the	np/n
student, book, dress	n	left	$np\backslash s$
knows, wrote, wears	$(np\backslash s)/np$	some, every, one, any, no	$(s/(np\backslash s))/n$
which, who	$(n\backslash n)/(np\backslash s)$	there, yet, already	$(np\backslash s)\backslash(np\backslash s)$
new, tall	n/n		

Given the lexicon above, our sample grammar recognizes the strings in (1), (2) and (3) as expressions of category s; the relative clause in (4-a) is recognized as an expression of type $n\backslash n$. By way of illustration, we give the derivations of (1-c) and (4-a). We use the familiar *parse tree* format, with vocabulary items as leaves and the types assigned to them in the lexicon as preterminals.

Sara wears the new dress $\in s?$ $\quad\rightsquigarrow\quad$ $np, (np\backslash s)/np, np/n, n/n, n \Rightarrow s?$

$$\dfrac{\dfrac{\text{Sara}}{np} \quad \dfrac{\dfrac{\text{wears}}{(np\backslash s)/np} \quad \dfrac{\dfrac{\text{the}}{np/n} \quad \dfrac{\dfrac{\text{new}}{n/n} \quad \dfrac{\text{dress}}{n}}{n}\,[\mathtt{FA}]}{np}\,[\mathtt{FA}]}{np\backslash s}\,[\mathtt{FA}]}{s}\,[\mathtt{BA}]$$

who knows Lori $\in n\backslash n?$ $\quad\rightsquigarrow\quad$ $(n\backslash n)/(np\backslash s), (np\backslash s)/np, np \Rightarrow n\backslash n?$

$$\dfrac{\dfrac{\text{who}}{(n\backslash n)/(np\backslash s)} \quad \dfrac{\dfrac{\text{knows}}{(np\backslash s)/np} \quad \dfrac{\text{Lori}}{np}}{np\backslash s}\,[\mathtt{FA}]}{n\backslash n}\,[\mathtt{FA}]$$

Turning to the remaining examples, our CG runs into problems. Let us look at the relative clauses first. The case of subject relativization (4-a) is derivable from the assignment $(n\backslash n)/(np\backslash s)$ to the relative pronoun, but this type will not do for object relativization (4-b), or for (4-c) where the relativized position is a non-peripheral constituent of the relative clause body. To generate these structures, our CG would have to multiply lexical assignments in an *ad hoc* way for each case. Writing tv as an abbreviation for $(np\backslash s)/np$, the assignment $((n\backslash n)/tv)/np$ to the relative pronoun would produce (4-b); for the non-peripheral case of relativization, yet another type would be needed —obviously, not a very satisfactory situation. In a similar way, multiple lexical assignments would be needed to obtain the examples in (5), with full noun phrases in direct object position: the lexicon, as it stands, only covers the subject case. Writing iv as an abbreviation for $np\backslash s$, the determiners *some, every, one, any, no* could be assigned a second type $(tv\backslash iv)/n$ for their occurrence in direct object position.

One way of dealing with this failure to express structural generalizations in lexical type assignments is to extend the inventory of combinatory rules of CG. The framework of Combinatory Categorial Grammar, developed by Mark Steedman, offers the most elaborate proposal for this strategy.

1.1.2 Combinatory Categorial Grammar

For an excellent exposition of Combinatory Categorial Grammar (CCG), we refer the reader to [Ste00][1]. The architecture of CCG is the same as that of CG: we can take over the definitions of the category language, the derives relation, lexicon, grammar G and the language generated by $L(G)$ from the previous section, with one important change: instead of having just the forward/backward application rules as non-logical axioms, CCG introduces a larger set of rule schemata. The name CCG derives from the fact that these extra schemata are inspired by the combinators of Curry's Combinatory Logic [CF68].

[1] In order to avoid confusion with the notation and facilitate the comparison between CCG and CTL we replace the "left-result" notation used in CCG, with the "result on top" one we have being using so far.

Below we present some of the rule schemata that have been proposed in the CCG framework, and we return to our toy fragment, to see how they can help in the cases where CG failed.

Lifting	$A \Rightarrow B/(A\backslash B)$	[T]
Forward Composition	$A/B, B/C \Rightarrow A/C$	[B]
Backward Crossed Composition	$A/B, A\backslash C \Rightarrow C/B$	[$\mathtt{B}_\times$]

Example 1.5. [Wh-Dependencies] Let us look first at the cases of direct object relativization in (4-b) and (4-c). Suppose we extend the lexicon given in Example 1.4 with a second type for *which* and *who*: $(n\backslash n)/(s/np)$. Intuitively, this type says that the relative pronoun looks for a clause with an *np* missing at the right edge. With the combinators [T] and [B], we can compose subject and transitive verb in (4-b), and produce the required type s/np for combination with the relative pronoun as shown in the derivation below. The [T] combinator lifts the subject *np* type into a fractional type $s/(np\backslash s)$ which can then combine with the transitive verb by means of Forward Composition.

$$\dfrac{\dfrac{\text{which}}{(n\backslash n)/(s/np)} \qquad \dfrac{\dfrac{\dfrac{\text{Sara}}{np}}{s/(np\backslash s)}[\mathtt{T}] \qquad \dfrac{\text{wrote}}{(np\backslash s)/np}}{s/np}[\mathtt{B}]}{n\backslash n}[\mathtt{FA}]$$

The combinators [B] and [T] are not enough to parse the phrase in (4-c): *which Sara wrote there*. Here, the missing *np* in the relative clause body comes from a non-peripheral position, whereas our lexical entry for non-subject relativization insists on a peripheral missing *np*, as indicated by the argument subtype s/np for the relative pronoun. To derive the non-peripheral case of relativization, our CCG grammar has to rely on the combinator [$\mathtt{B}_\times$] as illustrated below.

$$\dfrac{\dfrac{\text{which}}{(n\backslash n)/(s/np)} \qquad \dfrac{\dfrac{\text{Sara}}{s/(np\backslash s)}[\mathtt{T}] \qquad \dfrac{\dfrac{\text{wrote}}{(np\backslash s)/np} \qquad \dfrac{\text{there}}{(np\backslash s)\backslash(np\backslash s)}}{(np\backslash s)/np}[\mathtt{B}_\times]}{s/np}[\mathtt{B}]}{n\backslash n}[\mathtt{FA}]$$

Example 1.6. [Object generalized quantifiers] The next set of examples are the sentences with full noun phrases in direct object position. In our discussion of CG, we already noticed that the noun phrase *some book* can be assigned a type which allows it to combine with a transitive verb by means of Backward Application producing $np\backslash s$ as a result. A derivation is given in (i) below. In CCG , there is a second option for typing the direct object: $(s/np)\backslash s$. This type requires the combination of the subject and the transitive verb into a constituent of type s/np. This combination, as we have already seen in the derivation of relative clauses, can be obtained by means of the Composition combinator [B]. We present the derivations of (5-b) in (i) and (ii). In the discussion of meaning assembly in Section 1.3, we will come back to these two options for object generalized quantifiers.

(i)

$$\dfrac{\dfrac{\text{every\_student}}{s/(np\backslash s)} \qquad \dfrac{\dfrac{\text{knows}}{(np\backslash s)/np} \quad \dfrac{\text{some\_book}}{((np\backslash s)/np)\backslash(np\backslash s)}}{np\backslash s}\,[\mathtt{BA}]}{s}\,[\mathtt{FA}]$$

(ii)

$$\dfrac{\dfrac{\dfrac{\text{every\_student}}{s/(np\backslash s)} \quad \dfrac{\text{knows}}{(np\backslash s)/np}}{s/np}\,[\mathtt{B}] \qquad \dfrac{\text{some\_book}}{(s/np)\backslash s}}{s}\,[\mathtt{BA}]$$

Let us evaluate the CCG strategy. We notice first of all that a combinator like [$\mathtt{B}_\times$], which was used in the derivation of non-peripheral cases of extraction, implicitly involves a form of commutativity. It is obvious that such a combinator, if it would be available in its full generality, would lead to problems of overgeneration. CCG avoids such problems by restricting the application of combinatory rules to certain categories. Different languages could impose their individual restrictions on the rules; also, they can make their individual choices as to which combinators they allow. As for generative capacity, it is shown in [VW90] that an appropriately restricted version of CCG is weakly equivalent to linear indexed grammars, which means CCG belongs to the class of mildly context-sensitive formalisms. Important questions that remain are: What is the set of combinatory schemata allowed by Universal Grammar? and: Could we refine schemata in such a way that side conditions on their applicability can be avoided? These questions will be addressed in the next two sections.

1.2 A Logic of Types: Lambek 1958

At the beginning of this chapter, we commented on the resemblance between complex categories and fractions in arithmetic, and between the Application schemata and multiplication. The crucial insight of Lambek [Lam58] was that one can also see the categories as *logical formulas*. The changes introduced by this logical perspective with respect to the rule-based approach are summarized in Table 1.1. To start with, categories are seen as *formulas* and their type forming operators as connectives, i.e. *logical constants.* As a result, the rules for category combination can now be formulated as rules of inference for these connectives, rather than as the non-logical axiom schemata we had in CG and CCG. Parsing literally becomes a process of deduction in the logic of the categorial type formulas.

The logical perspective introduces another important theme: the distinction between proof theory and model theory. In the logical setup, formulas will be assigned a modeltheoretic interpretation. The syntactic side of derivations (the prooftheoretic machinery) can then be judged in terms of its soundness and completeness with respect to the proposed interpretation.

CG & CCG	L
Categories	Formulas
Type forming operators	Logical constants
Rule schemata	Inference Rules
Parsing	Deduction

Table 1.1: Rules-based approach vs. logical approach.

1.2.1 Parsing as Deduction

Let us look at the syntax of the Lambek calculus (L) first. Lambek himself presented his type logic in the format of a Gentzen-style Sequent Calculus [Gen38]. An alternative (equivalent) presentation[2], which is closer to the format we have used in the previous sections, is the Natural Deduction (N.D.) format.

Definition 1.7. [Natural Deduction Rules for L] Let Γ, Δ stand for finite non-empty sequences of formulas and A, B, C for logical formulas. The logical rules of L are:

$$\frac{}{A \vdash A}\;[\text{axiom}]$$

$$\frac{\Delta \vdash B/A \quad \Gamma \vdash A}{\Delta, \Gamma \vdash B}\;[/\text{E}] \qquad \frac{\Gamma \vdash A \quad \Delta \vdash A\backslash B}{\Gamma, \Delta \vdash B}\;[\backslash\text{E}]$$

$$\frac{\Delta, B \vdash C}{\Delta \vdash C/B}\;[/\text{I}] \qquad\qquad \frac{B, \Delta \vdash C}{\Delta \vdash B\backslash C}\;[\backslash\text{I}]$$

The rules of Forward and Backward Application in this format take the form of the familiar inference patterns of Modus Ponens, where we see the ‘fractional’ categories now as ‘implicational’ formulas. Compiling in the Cut rule of our definition of the ‘derives’ relation, we obtain the Elimination rules for ’/’ and ’\’. But the elimination rules capture only one half of the inferential possibilities of these connectives: they tell us how we can *use* an implicational formula in a derivation. To obtain the other half, we need inference rules to *derive* an implicational formula. These are the Introduction rules for the ’/’ and ’\’ connectives. As rules of inference, they give our grammar logic access to *hypothetical reasoning*: to obtain a formula C/B ($B\backslash C$), we withdraw a hypothesis B as the rightmost (leftmost) assumption of the antecedent sequence of formulas.

On the modeltheoretic side, we want to interpret formulas (i.e. syntactic categories) as sets of expressions, and the ‘derives’ relation as settheoretic inclusion at the interpretive level. In the systems considered so far, categorial combination was intuitively interpreted as concatenation. We can make this interpretation precise by considering semigroup models. It was shown by Pentus in [Pen95] that the calculus of [Lam88] is indeed sound and complete with respect to this interpretation.

Definition 1.8. [Semigroup Interpretation]

[2] See [Res00] for a detailed comparison of the two presentations.

$$\begin{array}{rcl} A\ B & = & \{xy \in M \mid x \in A \wedge y \in B\} \\ C/B & = & \{x \in M \mid \forall y(y \in B \rightarrow xy \in C)\} \\ B\backslash C & = & \{y \in M \mid \forall x(x \in B \rightarrow xy \in C)\}. \end{array}$$

A pleasant consequence of the shift to the logical perspective is that a number of combinators that have the status of non-logical axioms in CCG now turn out to be theorems of our type logic.

Example 1.9. [Hypothetical Reasoning] We show that the combinatory rules [T] and [B] of CCG considered above are theorems of L.

The combinator T of CCG. The lifting theorem, which raises a type to a higher order one[3], is a typical application of hypothetical reasoning. Its derivation is illustrated below.

$$\dfrac{\dfrac{\Delta \vdash A \quad [(A\backslash B) \vdash (A\backslash B)]^1}{\Delta, (A\backslash B) \vdash B}\,[\backslash \mathrm{E}]}{\Delta \vdash B/(A\backslash B)}\,[/\mathrm{I}]^1$$

The derivation proves that if a structure Δ is of type A, then it is of type $B/(A\backslash B)$ as well. The proof is given by hypothetical reasoning: Assume a structure of type $A\backslash B$, given $\Delta \vdash A$, then Δ composed with $A\backslash B$ is of type B. Then by withdrawing the hypothesis by means of the coindexed rule, Δ is proved to be of the higher order type. Note that the introduction rule can discharge one hypothesis at a time since we are in a resource sensitive system..

The combinator B of CCG. The forward composition added in CCG to the function application of CG is derivable in L as shown below:

$$\dfrac{\dfrac{\Delta \vdash A/B \quad \dfrac{\Gamma \vdash B/C \quad [C \vdash C]^1}{\Gamma, C \vdash B}\,[/\mathrm{E}]}{\Delta, \Gamma, C \vdash A}\,[/\mathrm{E}]}{\Delta, \Gamma \vdash A/C}\,[/\mathrm{I}]^1$$

Similarly to the previous derivation, the combinator is inferred by means of the logical rules of L. In particular, the derivation is based on the hypothetical reasoning: it starts by assuming a hypothesis C and it withdraws it once the functions are composed.

Let us turn to the examples of our toy fragment, and present some Lambek derivations in the sequent-style Natural Deduction format introduced above. The leaves of the N.D. derivations are axioms $A \vdash A$. Some of these leaves correspond to lexical assumptions, others to hypothetical assumptions that will have to be withdrawn in the course of the derivation. To make the derivations more readable, we replace the formula on the left of $\vdash$ by the lexical item in the case of lexical assumptions.

[3] The order of the categories is defined as following: $\texttt{order}(A) = 0$, if $A \in \mathsf{ATOM}, \texttt{order}(A/B) = \texttt{max}(\texttt{order}(A), \texttt{order}(B) + 1)$ and the same holds for $(B\backslash A)$.

Example 1.10. [Function Application in L] Given the lexicon of our toy grammar, the expression in (4-a), *who knows Lori*, is shown to be an expression of type $n\backslash n$ as follows.

$$\dfrac{\text{who} \vdash (n\backslash n)/(np\backslash s) \qquad \dfrac{\text{knows} \vdash (np\backslash s)/np \quad \text{Lori} \vdash np}{\text{knows, Lori} \vdash np\backslash s}\,[/\text{E}]}{\text{who, knows, Lori} \vdash n\backslash n}\,[/\text{E}]$$

As we discussed above, hypothetical reasoning is applied in the derivation of the combinator [B] which is required to account for right-peripheral extraction. We show how the structure *which Sara wrote* is proved to be grammatical in L.

Example 1.11. [Right-Peripheral Extraction in L] The string *which Sara wrote* is derived as an expression of type $n\backslash n$, by starting from the lexical entries it consists of and by assuming a hypothetical np taken as object by the transitive verb.

$$\dfrac{\text{which} \vdash (n\backslash n)/(s/np) \qquad \dfrac{\dfrac{\text{Sara} \vdash np \qquad \dfrac{\text{wrote} \vdash (np\backslash s)/np \quad [np \vdash np]^1}{\text{wrote}, np \vdash np\backslash s}\,[/\text{E}]}{\text{Sara, wrote}, np \vdash s}\,[\backslash\text{E}]}{\text{Sara, wrote} \vdash s/np}\,[/\text{I}]^1}{\text{which, Sara, wrote} \vdash n\backslash n}\,[/\text{E}]$$

First, the string 'Sara, wrote, np' is proved to be of category s. Then, the hypothesis np is withdrawn. This is done by means of [/I] which produces the formula s/np required by the type assigned to the relative pronoun.

The type logic L does not succeed in producing a derivation for the case of non-peripheral extraction *which Sara wrote there*. As we saw in our discussion of Backward Crossed Composition [$\mathsf{B}_\times$], this combinator involves a form of commutativity. This combinator, in other words, is not a valid theorem of L —it would violate the concatenation interpretation. Summing up, by making the shift to a type *logic*, we have gained a better understanding of the CCG combinators, seeing which ones are indeed valid given the interpretation of the type-forming connectives and which ones are not. But as a linguistic framework, L is not expressive enough to deal with the phenomena illustrated by our toy fragment. The proof by Mati Pentus [Pen93] that L grammars are context free provides the formal underpinnings for this claim.

1.2.2 Logical Rules and Structural Rules

The presentation of the antecedent part Γ in a sequent $\Gamma \vdash A$ as a sequence of formulas hides an implicit structural assumption about grammatical composition, *viz.* that it is an associative operation, which ignores the hierarchical constituent structure of type formulas. Lambek in his [Lam61] paper was the first to notice that this assumption is too strong, and that it leads to overgeneration. The formulation of his [Lam61] system removes the implicit structural assumption, which means that structural rules have to be introduced in a fully explicit fashion. The type logics so obtained have a combination of logical rules for the connectives (Introduction and Elimination rules), plus structural rules of inference for the manipulation of antecedent configurations. Structures are built from the set of formulas FORM by means of the *binary* structural operator $\circ$ as follows.

i. If $A \in \mathsf{FORM}$, then $A \in \mathsf{STRUCT}$;
ii. If Γ and $\Delta \in \mathsf{STRUCT}$, then $(\Gamma \circ \Delta) \in \mathsf{STRUCT}$.

The separation of logical and structural rules makes it possible to generate a family of logics with the same logical rules, but different structural rules. We refer to this family as Categorial Type Logics (CTLs). The base logic for this family is the system presented in [Lam61]: the type logic with absolutely no structural rules. It is usually abbreviated as NL, because it is obtained from L by dropping associativity.

Definition 1.12. [The Lambek Family]. **Logical rules** for the base logic NL:

$$\frac{}{A \vdash A}\,[\text{axiom}]$$

$$\frac{\Delta \vdash B/A \quad \Gamma \vdash A}{(\Delta \circ \Gamma) \vdash B}\,[/\text{E}] \qquad \frac{\Gamma \vdash A \quad \Delta \vdash A\backslash B}{(\Gamma \circ \Delta) \vdash B}\,[\backslash\text{E}]$$

$$\frac{(\Delta \circ B) \vdash C}{\Delta \vdash C/B}\,[/\text{I}] \qquad \frac{(B \circ \Delta) \vdash C}{\Delta \vdash B\backslash C}\,[\backslash\text{I}]$$

Structural rules. Let us write $\Gamma[\Delta]$ for a structure Γ contaning a distinguished occurrence of the substructure Δ. Adding a structural rule of Associativity [ass] to NL , one obtains L . By adding commutativity [per] to L one obtains LP [Ben88]. The picture is completed with the non associative and commutative Lambek calculus NLP.

$$\frac{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash C}{\overline{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash C}}\,[\text{ass}] \qquad \frac{\Gamma[(\Delta_2 \circ \Delta_1)] \vdash C}{\Gamma[(\Delta_1 \circ \Delta_2)] \vdash C}\,[\text{per}]$$

Multimodal systems. The structural rules above apply in a *global* fashion. While discussing the linguistic application of L and of CCG, we have noted that we need *control* over structural options. In the so-called multimodal version of CTL, the required control is achieved by distinguishing different *modes* of composition, which can then live together and interact within one grammatical logic. In the notation, we keep the different modes apart by indexing the logical and the structural connectives, *i.e.* we now write $(\backslash_i, /_i)$ and $\circ_i$, where $i \in I$ and I is a set of mode indices. The different modes have the same logical rules, but they can differ in their structural properties. Thus, one can introduce structural rules *locally* by restricting them to a certain family of logical constants. Finally, the addition of modes increases the number of logics which can be obtained from the base logic. Besides associativity and/or commutativity options for individual composition modes, one can formulate inclusion and interaction rules for configurations involving multiple modes.

i. *Inclusion* structural rules (also known as entropy principles), e.g. if $\Gamma[\Delta \circ_1 \Delta'] \vdash A$ then $\Gamma[\Delta \circ_2 \Delta'] \vdash A$;

ii. Interaction structural rules which mix distinct modes.

For an illustration of interaction principles, we can return to the non-peripheral extraction example in our toy fragment. Suppose we have the structural rules below for the interaction between two modes, $\circ$ and $\circ_a$.

$$\frac{\Gamma[\Delta_1 \circ (\Delta_2 \circ_a \Delta_3)] \vdash C}{\Gamma[(\Delta_1 \circ \Delta_2) \circ_a \Delta_3] \vdash C}\,[\text{mixass}] \qquad \frac{\Gamma[(\Delta_1 \circ_a \Delta_2) \circ \Delta_3] \vdash C}{\Gamma[(\Delta_1 \circ \Delta_3) \circ_a \Delta_2] \vdash C}\,[\text{diss}]$$

Example 1.13. [Non-Peripheral Extraction] We modify the lexicon in such a way that $\circ$ is used for regular phrasal composition, and $\circ_a$ for extraction. We need a type assignment to introduce a *wh* dependency, and a type assignment to eliminate it. In this example, these are $(n\backslash n)/(s/_a np)$ for the relative pronoun, and $(np\backslash s)/_a np$ for the transitive verb, The derivation of *which Sara wrote there* is then as follows.

$$\dfrac{\text{which} \vdash (n\backslash n)/(s/_a np) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{Sara} \vdash np \qquad \dfrac{\dfrac{\text{wrote} \vdash (np\backslash s)/_a np \quad [np \vdash np]^1}{\text{wrote} \circ_a np \vdash np\backslash s}\,[/_a\text{E}] \qquad \text{there} \vdash (np\backslash s)\backslash(np\backslash s)}{((\text{wrote} \circ_a np) \circ \text{there}) \vdash np\backslash s}\,[\backslash\text{E}]}{\text{Sara} \circ ((\text{wrote} \circ_a np) \circ \text{there}) \vdash s}\,[\backslash\text{E}]}{\text{Sara} \circ ((\text{wrote} \circ \text{there}) \circ_a np) \vdash s}\,[\text{diss}]}{(\text{Sara} \circ (\text{wrote} \circ \text{there})) \circ_a np \vdash s}\,[\text{mixass}]}{(\text{Sara} \circ (\text{wrote} \circ \text{there})) \vdash s/_a np}\,[/_a\text{I}]^1}{\text{which} \circ (\text{Sara} \circ (\text{wrote} \circ \text{there})) \vdash n\backslash n}\,[/\text{E}]$$

Note that the application of the structural rules is *lexically anchored.* The modes labelling the connectives of the types assigned to the transitive verb *wrote* and the relative pronoun *which* drive the structural reasoning in the derivation. The structural rule [diss] brings the *np* in the peripheral position and [mixass] makes it available to the abstraction. The application of these rules is restricted to the environments requiring them.

We have seen that in CCG the above expression is parsed by applying the combinator $[\mathsf{B}_\times]$. The latter is derivable in NL extended with the structural rules above. However, the use of modes to account for long distance phenomena is still not completely satisfactory since the application of the structural rules is tied to the lexical entries both of the relative pronoun and the transitive verb, which now gets a special lexical entry that allows its direct object to be extracted: $(np\backslash s)/_a np$ in contrast with the linguistic facts. We will come back to this point in Chapter 3 after we have explored the algebraic structure of NL.

The example above illustrate how modes and structural rules can be used to account for differences among contexts *within* the same languages. Similarly, these logical tools are used to account for differences holding *across* languages. By way of illustration, we look at Italian and English adjectives.

Example 1.14. [Italian vs. English Adjectives] English and Italian adjectives may differ in their ordering possibilities with respect to a noun.

(6) a. Sara wears a new dress.
b. *Sara wears a dress new.

(7) a. Sara indossa un nuovo vestito.
Sara wears a new dress
tr. Sara wears a new dress.
b. Sara indossa un vestito nuovo.
Sara wears a dress new
tr. Sara wears a new dress.

As the examples show, some adjectives in Italian require more freedom with respect to word order than their English counterparts. This crosslinguistic difference can be expressed by assigning different logical types to Italian and English adjectives. Since the exhibited structural property is not shared by all Italian phrases, the structural freedom of the adjectives must have been lexically anchored. This restriction can be expressed by means of modes. Let us try to make things more concrete by looking at the derivation of the relevant structures in (6) and (7). Let *qp* abbreviate the type of quantifier phrases.

(i)

$$\dfrac{\dfrac{\mathrm{a} \vdash qp/n \qquad \dfrac{\mathrm{new} \vdash n/n \quad \mathrm{dress} \vdash n}{\mathrm{new} \circ \mathrm{dress} \vdash n}[/\mathrm{E}]}{\mathrm{a} \circ (\mathrm{new} \circ \mathrm{dress}) \vdash qp}[/\mathrm{E}]}{\mathrm{a} \circ (\mathrm{dress} \circ \mathrm{new}) \vdash qp}[\mathrm{per}_\bullet]^*$$

(ii)

$$\dfrac{\dfrac{\mathrm{un} \vdash qp/n \qquad \dfrac{\mathrm{nuovo} \vdash n/_c n \quad \mathrm{vestito} \vdash n}{\mathrm{nuovo} \circ_c \mathrm{vestito} \vdash n}[/_c\mathrm{E}]}{\mathrm{un} \circ (\mathrm{nuovo} \circ_c \mathrm{vestito}) \vdash qp}[/\mathrm{E}]}{\mathrm{un} \circ (\mathrm{vestito} \circ_c \mathrm{nuovo}) \vdash qp}[\mathrm{per}_\bullet]$$

The $*$ on the last step of the derivation in (i) marks where the derivation fails in accounting for (6). On the other hand, the use of a commutative composition operator, introduced by the lexical assignment of *nuovo*, allows the permutation required to build the structures in (7).

The logical and structural modules of CTL have been used to account for the constants of grammatical reasoning and the structural variations, respectively. In Chapter 2, we show how NL is interpreted by a *universal* algebraic structure which can be restricted so to capture the variations expressed by the other CTLs obtained from NL by adding structural rules.

In this thesis, attention is focused on the logical module. To this end, in Chapter 2 we investigate the algebraic structure of NL and highlight other logical properties which have not been investigated so far. When we make use of structural rules (Chapter 4), we apply them to carry semantic information which are universally shared. On the other hand, when we assume a crosslinguistic perspective, (Chapter 7), the differences across languages are reduced to different lexical type assignments exploiting the expressivity of the logical module.

1.2.3 Structural Constraints

It will be clear from the above that structural rules have an effect on the generative capacity of CTL systems. The base logic NL is strictly context free. By allowing structural rules to copy or delete type formulas, the systems become Turing-complete [Car99]. But it is shown in [Moo02] that with a linearity restriction on structural rules, one stays within PSPACE, the complexity class of context-sensitive grammars. The linearity constraint requires structural rules to be non-expanding in the sense defined below.

Definition 1.15. [Non-Expanding Structural Rules] Given an antecedent configuration Σ, the length of Σ is defined as follows:

$$\begin{array}{lcl} \mathit{length}(\Delta_1 \circ \Delta_2) & = & \mathit{length}(\Delta_1) + \mathit{length}(\Delta_2) + 2 \\ \mathit{length}(\Delta) & = & 0. \end{array}$$

A structural rule

$$\frac{\Gamma[\Sigma'[\Delta_1, \ldots, \Delta_n]] \vdash C}{\Gamma[\Sigma[\Delta_{\pi_1}, \ldots, \Delta_{\pi_n}]] \vdash C}$$

where Σ' is non empty, is *non-expanding* if

$$\mathit{length}(\Sigma[\Delta_{\pi_1}, \ldots, \Delta_{\pi_n}]) \leq \mathit{length}(\Sigma'[\Delta_1, \ldots, \Delta_n]).$$

1.3 The Composition of Meaning

Linguistic signs have a form and a meaning component. The discussion so far has concentrated on the form aspect of grammatical composition. Let us turn now to meaning assembly and the relation between natural language form and meaning. See [Gam91] for an introduction to the field of formal semantics. Montague's Universal Grammar program [Tho74] provides a general framework to study these issues. The core of this program is an algebraic formulation of Frege's principle of compositionality [Fre84]. Intuitively, the principle says that the meaning of a complex syntactic expression is a function of the meaning of its constituent parts and of the derivational steps that have put them together. Montague formalizes the principle as a mapping between a syntactic and a semantic algebra. The mapping is a *homomorphism*, *i.e.* it preserves structure in the following sense [Jan97].

Definition 1.16. [Homomorphism] Let $\mathcal{A} = (A, F)$ and $\mathcal{B} = (B, G)$ be algebras. A mapping $m : \mathcal{A} \rightarrow \mathcal{B}$ is called a *homomorphism* if there is a mapping $m' : F \rightarrow G$ s.t. for all $f \in F$ and all $a_1, \ldots, a_m \in A$ holds $m(f(a_1, \ldots, a_n)) = m'(f)(m(a_1), \ldots, m(a_n))$.

1.3.1 Semantic Types and Typed Lambda Terms

The definition above requires the syntactic algebra and the semantic algebra of a grammar to work in tandem. Syntactic combinatorics is determined by the syntactic categories, similarly the semantic laws of composition are governed by semantic types. To set up the form-meaning correspondence, it is useful to build a language of semantic types in parallel to the syntactic type language.

Definition 1.17. [Types] Given a non-empty set of *basic types* Base, the set of types TYPE is the smallest set such that

i. $\mathsf{Base} \subseteq \mathsf{TYPE}$;

ii. $(a, b) \in \mathsf{TYPE}$, if a and $b \in \mathsf{TYPE}$.

Note that this definition closely resembles the one of the syntactic categories of CG. The only difference is the lack of directionality of the functional type (a, b). A function mapping the syntactic categories into TYPE can be given as follows.

Definition 1.18. [Categories and Types] Let us define a function $\mathtt{type} : \mathsf{CAT} \rightarrow \mathsf{TYPE}$ which maps syntactic categories to semantic types.

$$\begin{array}{ll}
\mathtt{type}(np) = e; & \mathtt{type}(A/B) = (\mathtt{type}(B), \mathtt{type}(A)); \\
\mathtt{type}(s) = t; & \mathtt{type}(B\backslash A) = (\mathtt{type}(B), \mathtt{type}(A)); \\
\mathtt{type}(n) = (e, t). &
\end{array}$$

To represent meaning assembly, we use the tools of the typed λ-calculus. Terms are built out of variables and constants of the various types.

Definition 1.19. [Typed λ-terms] Let VAR_a be a countably infinite set of *variables* of type a and CON_a a collection of *constants* of type a. The set TERM_a of λ-terms of type a is defined by mutual recursion as the smallest set such that the following holds:

i. $\mathsf{VAR}_a \subseteq \mathsf{TERM}_a$,

ii. $\mathsf{CON}_a \subseteq \mathsf{TERM}_a$,

iii. $(\alpha(\beta)) \in \mathsf{TERM}_a$ if $\alpha \in \mathsf{TERM}_{(a,b)}$ and $\beta \in \mathsf{TERM}_b$,

iv. $\lambda x.\alpha \in \mathsf{TERM}_{(a,b)}$, if $x \in \mathsf{VAR}_a$ and $\alpha \in \mathsf{TERM}_b$.

We represent with α_a a term α of type a.

The relevant items are *iii.* and *iv.* The former defines function application, the latter abstraction over variables. The λ is an operator which binds variables following specific constraints for which it is important to distinguish free and bound variables.

Definition 1.20. [Free and Bound Variables] The set $\mathrm{Free}(\alpha)$ of *free variables* of the λ-term α is defined by

i. $\mathrm{Free}(x_b) = \{x_b\}$ if $x_b \in \mathsf{VAR}_b$,

ii. $\mathrm{Free}(c_b) = \{\}$ if $c_b \in \mathsf{CON}_b$,

iii. $\mathrm{Free}(\alpha_{(a,b)}(\beta_a)) = \mathrm{Free}(\alpha_{(a,b)}) \cup \mathrm{Free}(\beta_a)$,

iv. $\mathrm{Free}(\lambda x_a.\alpha_b) = \mathrm{Free}(\alpha_b) - \{x_a\}$.

A variable v' is *free for v in* the expression β iff no free occurrence of v in β is within the scope of $\lambda v'$.

Reduction rules determine the equivalence among λ-terms.

Definition 1.21. [Reduction Rules] The λ-calculus is characterized by the following *reduction rules*, where $\alpha_b([\beta_a/x_a]))$ stands for the result of substituting a term β_a for x_a in α_b.

$$\begin{array}{lll} (\lambda x_a.\alpha_b)(\beta_a) \Rightarrow \alpha_b[\beta_a/x_a] & x_a \text{ is free for } \beta_a \text{ in } \alpha_b & \beta\text{-reduction} \\ \lambda x_a.\alpha_{(a,b)}(x_a) \Rightarrow \alpha_{(a,b)} & x_a \text{ is not free in } \alpha_{(a,b)} & \eta\text{-reduction} \end{array}$$

These rules reduce a term into a simpler one. Applying this re-writing system we can determine whether two terms are logically equivalent, *viz.* whether they reduce to a common result. An important theorem concerning λ-calculus is that reduction eventually terminates with a term that can no longer be reduced using the above reduction rules. Such a term is said to be in *β,η normal form.*

The main novelty introduced by Montague is that the interpretation of the type-theoretical logical system may also serve as the interpretation of natural language expressions. To this end, he adopted a model theoretic semantics. When applied to natural language, model theory can be thought of as a theory designed to explain entailment relations among sentences and consequently to account for truth conditions of meanings. In order to capture these relations, meanings are seen as objects in an abstract model. A bit more formally, this is expressed by saying that natural language sentences refer to or *denote* objects in the model. In other words, the denotation assigned to typed lambda terms serve as a bridge to interpret linguistic expressions. Models are pairs consisting of a frame and a valuation. They are defined below.

Definition 1.22. [Frame] A *frame* D consists of the collection of basic domains, *i.e.* $\cup_{\alpha\in\mathsf{Base}} Dom_\alpha$ and the domains for functional types. The latter are as follows

$$Dom_{(a,b)} = Dom_b^{Dom_a} = \{f \mid f : Dom_a \to Dom_b\}.$$

In words, expressions corresponding to functional types, like verb phrases, denote in the set of functions from the domain of their argument to the domain of their value. In our case, given the set of individuals E, the domains of functions are built up from the primitive ones below:

$$Dom_e = E \quad \text{and} \quad Dom_t = \{1, 0\}.$$

Besides the set of typed domains, a model must include an interpretation function I mapping the items of the lexicon to elements of the domains.

Definition 1.23. [Model] A *model* is a pair $\mathcal{M} = \langle D, I\rangle$ in which the interpretation of the constant terms `lex` in the lexicon `Lexicon` of a given language are obtained as follow

i. D is a frame;

ii. The interpretation function is $I : \mathtt{Lexicon} \to D$, s.t. if α is of type a, $I(\alpha) \in Dom_a$.

The interpretation function over lexical expressions is extended by the denotation function which recursively assigns an interpretation to all expressions.

Definition 1.24. [Denotation] The *denotation* $[\![\alpha_a]\!]^f_{\mathcal{M}}$ of a λ-term α_a with respect to the model $\mathcal{M} = \langle D, I\rangle$ and assignment f, where $f : \mathsf{VAR}_a \to Dom_a$, is given by

i. $[\![x_a]\!]^f_{\mathcal{M}} = f(x_a)$ if $x_a \in \mathsf{VAR}_a$.
ii. $[\![\alpha_a]\!]^f_{\mathcal{M}} = I(\alpha_a)$ if $\alpha_a \in \mathsf{CON}_a$.
iii. $[\![\alpha_{(a,b)}(\beta_a)]\!]^f_{\mathcal{M}} = [\![\alpha_{(a,b)}]\!]^f_{\mathcal{M}}([\![\beta_a]\!]^f_{\mathcal{M}})$.
iv. $[\![\lambda x_a.\alpha_b]\!]^f_{\mathcal{M}} = g$ such that $g(d) = [\![\alpha_b]\!]^{f[x_a:=d]}_{\mathcal{M}}$.

where $f[x_a := d]$ stands for the assignment that maps x_a to $d \in Dom_a$ and maps $y_a \neq x_a$ to $f(y_a)$.

Intuitively, the denotation of a term formed by the λ-operator says that applying the denotation of a functional term $\lambda x.\alpha$ to an object d is the result of evaluating α in an assignment where x takes the value d.

Remark 1.25. The form and meaning components of linguistic signs are inhabitants of their corresponding syntactic and semantic types, respectively. The definitions above say that two signs may differ in their form (belong to different syntactic types) despite being similar in their meaning (belonging to the same semantic type). Consequently, the two signs receive the same interpretation denoting the same object in the domain. For instance, this is the case of signs whose forms are in the syntactic type A/B and $B\backslash A$ and, therefore, their meanings are in the semantic type $(\mathtt{type}(B), \mathtt{type}(A))$ and are interpreted in the domain $Dom_{(b,a)}$.

1.3.2 Interpretations for the Sample Grammar

Natural language expressions can be interpreted by assuming either a relational or a functional perspective. We briefly illustrate the two approaches and their connection by discussing some examples. As a notational convention, we represent the constants in TERM with special fonts. For the ease of presentation, we do not indicate the semantic types unless necessary. For instance, the individual *Lori* is assigned a denotation in the domain of entities, and is represented by the term `lori`. The meaning of complex phrases is built out of the meaning of the lexical items. Thus we must start by adding the semantic information in the lexicon.

Definition 1.26. [Term Labelled Lexicon] Given a set of basic expressions of a natural language Σ, a term labeled categorial lexicon is a relation,

$$\mathsf{LEX} \subseteq \Sigma \times (\mathsf{CAT} \times \mathsf{TERM}) \text{ such that if } (w, (A, \alpha)) \in \mathsf{LEX}, \text{ then } \alpha \in \mathsf{TERM}_{\mathtt{type}(A)}$$

This constraint on lexical entries enforces the requirement that if the expression w is assigned a syntactic category A and term α, then the term α is of the appropriate type for the category A.

Example 1.27. [Extended Lexical Entries] Labelled lexical entries are for instance the ones below,

Sara	$np:$	`sara`	which	$(n\backslash n)/(np\backslash s):$	$\lambda xyz.x(z)\wedge y(z)$
Pim	$np:$	`pim`	which	$(n\backslash n)/(np\backslash s):$	$\lambda xyz.x(z)\wedge y(z)$
Lori	$np:$	`lori`	some	$(s/(np\backslash s))/n:$	$\lambda xy.\exists z(x(z)\wedge y(z))$
knows	$(np\backslash s)/np:$	`know`	some	$((s/np)\backslash s)/n:$	$\lambda xy.\exists z(x(z)\wedge y(z))$
student	$n:$	`student`	some	$(tv\backslash(np\backslash s))/n:$	$\lambda xyu.\exists z(x(z)\wedge y(z)(u))$
professor	$n:$	`professor`	every	$(s/(np\backslash s))/n:$	$\lambda xy.\forall z(x(z)\rightarrow y(z))$
tall	$n/n:$	`tall`	every	$((s/np)\backslash s)/n:$	$\lambda xy.\forall z(x(z)\rightarrow y(z))$

Notice the different term assignment for the logical (the determiners and the relative pronoun) and the non-logical constants.

The denotations of the linguistic expressions are illustrated by the examples below.

Example 1.28. [Relational Interpretation of Non-Logical Constants] Let our model be based on the set of entities $E=\{\texttt{lori},\texttt{ale},\texttt{sara},\texttt{pim}\}$ which represent *Lori, Ale, Sara* and *Pim*, respectively. Assume that they all know themselves, plus *Ale* and *Lori* know each other, but they do not know *Sara* or *Pim*; *Sara* does know *Lori* but not *Ale* or *Pim*. The first three are students whereas *Pim* is a professor, and both *Lori* and *Pim* are tall. This is easily expressed set theoretically. Let $[\![\texttt{w}]\!]$ indicate the interpretation of w:

$$\begin{array}{lcl}
[\![\texttt{sara}]\!] & = & \text{sara;} \\
[\![\texttt{pim}]\!] & = & \text{pim;} \\
[\![\texttt{lori}]\!] & = & \text{lori;} \\
[\![\texttt{know}]\!] & = & \{\langle\text{lori, ale}\rangle, \langle\text{ale,lori}\rangle, \langle\text{sara, lori}\rangle, \\
 & & \langle\text{lori, lori}\rangle, \langle\text{ale, ale}\rangle, \langle\text{sara, sara}\rangle, \langle\text{pim, pim}\rangle\}; \\
[\![\texttt{student}]\!] & = & \{\text{lori, ale, sara}\}; \\
[\![\texttt{professor}]\!] & = & \{\text{pim}\}; \\
[\![\texttt{tall}]\!] & = & \{\text{lori, pim}\}.
\end{array}$$

which is nothing else to say that, for example, the relation *know* is the set of pairs $\langle\alpha,\beta\rangle$ where α knows β; or that 'student' is the set of all those elements which are a student.

Alternatively, one can assume a functional perspective and interpret, for example, *know* as a function $f: Dom_e\rightarrow(Dom_e\rightarrow Dom_t)$. The shift from the relational to the functional perspective is made possible by the fact that the sets and their characteristic functions amount to the same thing: if f_X is a function from Y to $\{0,1\}$, then $X=\{y\mid f_X(y)=1\}$. In other words, the assertion '$y\in X$' and '$f_X(y)=1$' are equivalent.[4]

The interpretation of complex phrases is obtained by interpreting the corresponding lambda terms. For example, if $\texttt{walk}\in\mathsf{CON}_{(e,t)}$ and $x\in\mathsf{VAR}_e$, then $\texttt{walk}(x)$ expresses the fact that x has the property of walking, whereas $\lambda x.\texttt{walk}(x)$ is an abstraction over x and it represents the property itself. Moreover, due to the reduction rules of the lambda calculus the constant $\texttt{walk}_{(e,t)}$ is equivalent to the term $\lambda x_e.(\texttt{walk}(x))_t$. Applying Definition 1.24 this term is denoted by a function g such that for each entity $d\in Dom_e$, gives $g(d)=1$ iff $[\![\texttt{walk}(x)]\!]^{f[x:=d]}_{\mathcal{M}}=1$, or in other words iff x has the property expressed by `walk`.

The logical constants are interpreted by using set theoretical operations as illustrated below.

[4] Consquently, the two notations $y(z)(u)$ and $y(u,z)$ are equivalent.

Example 1.29. [Logical Constants] By evaluating the lambda expressions in Example 1.27 in a model, one obtains the interpretations below:

$$\begin{array}{lcl}
[\![\texttt{no N}]\!] & = & \{X \subseteq E \mid [\![\texttt{N}]\!] \cap X = \emptyset\}. \\
[\![\texttt{some N}]\!] & = & \{X \subseteq E \mid [\![\texttt{N}]\!] \cap X \neq \emptyset\}. \\
[\![\texttt{every N}]\!] & = & \{X \subseteq E \mid [\![\texttt{N}]\!] \subseteq X\}. \\
[\![\texttt{N which VP}]\!] & = & [\![\texttt{N}]\!] \cap [\![\texttt{VP}]\!].
\end{array}$$

Generalized quantifiers have attracted the attention of many researchers working on the interaction between logic and linguistics [KF85, Eij85]. We will come back to them in Chapter 6.

Taking advantage of the fact that the denotation of all natural language expressions can be reduced to sets, we can extend our model with a partial order recursively defined again by means of types [GS84, Ben86].

Definition 1.30. [Partially Ordered Domains] Let $\mathcal{M} = \langle D, \leq, I\rangle$ be our model, where $\leq$ is defined recursively as follows

$$\begin{array}{lll}
\text{If } \beta, \gamma \in Dom_e, \text{ then} & [\![\beta]\!] \leq_e [\![\gamma]\!] & \text{iff } [\![\beta]\!] = [\![\gamma]\!] \\
\text{If } \beta, \gamma \in Dom_t, \text{ then} & [\![\beta]\!] \leq_t [\![\gamma]\!] & \text{iff } [\![\beta]\!] = 0 \text{ or } [\![\gamma]\!] = 1 \\
\text{If } \beta, \gamma \in Dom_{(a,b)}, \text{ then} & [\![\beta]\!] \leq_{(a,b)} [\![\gamma]\!] & \text{iff } \forall \alpha \in Dom_a, [\![\beta(\alpha)]\!] \leq_b [\![\gamma(\alpha)]\!].
\end{array}$$

Let us look at our toy-model again and check the order relations holding among its expressions. Establishing such an order is quite immediate when working with sets, and only appears more complex when using the recursive Definition 1.30.

Example 1.31. [Order Relations] The set denoting the expression *tall student*, obtained by taking all elements which are in both the sets $[\![\text{tall}]\!]$ and $[\![\text{student}]\!]$, *viz.* $[\![\text{tall student}]\!] = \{\texttt{lori}\}$, is clearly a subset of the set denoting *student*, *viz.* $[\![\text{student}]\!] = \{\texttt{lori}, \texttt{ale}, \texttt{sara}\}$. Using functional denotation the proof of $[\![\text{tall student}]\!] \leq_{(e,t)} [\![\text{student}]\!]$ is as follows.

$$\begin{array}{ll}
[\![\text{tall student}]\!] \leq_{(e,t)} [\![\text{student}]\!] & \text{iff } \forall \alpha \in D_e \\
[\![\text{tall student}(\alpha)]\!] \leq_t [\![\text{student}(\alpha)]\!] & \text{iff} \\
[\![\text{tall student}]\!]([\![\alpha]\!]) \leq_t [\![\text{student}]\!]([\![\alpha]\!]) & \text{iff} \\
[\![\text{tall student}]\!]([\![\alpha]\!]) = 0 \text{ or } [\![\text{student}]\!]([\![\alpha]\!]) = 1. &
\end{array}$$

Assume this is not true, *viz.* $[\![\text{tall student}]\!]([\![\alpha]\!]) = 1$ and $[\![\text{student}]\!]([\![\alpha]\!]) = 0$. Then $\exists d \in Dom_e$ s.t. $d \in [\![\text{tall student}]\!]$, but $d \not\in [\![\text{student}]\!]$, which is obviously impossible. Note, that the inclusion relation is due to the presence of the intersective predicate *tall*.

Finally, having a formal definition of the domains of interpretation and a partial order over them, one can distinguish expressions interpreted in the same domain but which differ with respect to the partial order. For instance, we can distinguish upward (↑Mon) and downward (↓Mon) monotone functions, where the former preserve and the latter reverse the partial order. We illustrate this concept by means of the example below.

Example 1.32. [Monotonicity in Natural Language] Let W be our domain of interpretation. Consider the generalized quantifier *every N*, which has a syntactic category $s/(np\backslash s)$. It is interpreted as a function in $D_{(e,t)} \to D_t$, defined by

$$[\![\mathtt{every\_N}(X)]\!] = 1 \text{ iff } \mathrm{card}([\![N]\!] - [\![X]\!]) = 0.$$

To prove that *every_N* is an $\uparrow$Mon function, we have to show that whenever $[\![X]\!] \leq [\![Y]\!]$ then $[\![\mathtt{every\_N}(X)]\!] \leq [\![\mathtt{every\_N}(Y)]\!]$. Assume $[\![\mathtt{every\_N}(X)]\!] = 1$, then it holds that $\mathrm{card}([\![N]\!] - [\![X]\!]) = 0$ by definition. This implies that for every superset Y of X, $\mathrm{card}([\![N]\!] - [\![Y]\!]) = 0$. Hence $[\![X]\!] \leq [\![Y]\!]$ and $[\![\mathtt{every\_N}(X)]\!] = 1$ implies $[\![\mathtt{every\_N}(Y)]\!] = 1$ and we are done.

In a similar way, one can prove that, for instance, *nobody* is a downward monotone function.

The connection between syntactic categories and semantic types seems to be lost when looking at the current research in CTL and in the Montagovian school. The categorial grammarians are mostly interested in the grammaticality of linguistic structures, whereas the Montagovians are focused on their interpretation and entailment relations. The system presented in Chapter 4 combines again the two traditions by exploiting their logical connection. In particular, we encode the different monotonicity properties of GQs in the logical types of a CTL and make use of Definition 1.30 to achieve a proof theoretical account of natural reasoning.

1.4 Putting Things Together

In this section we explain how the syntactic derivations of the formal grammars discussed in Sections 1.1 and 1.2 are associated with instructions for meaning assembly.

1.4.1 Rule-Based Approach vs. Deductive Approach

In CG and CCG , the syntactic rules for category combination have the status of non-logical axioms. To obtain a Montague-style compositional interpretation, we have to associate them with instructions for meaning assembly in a rule-by-rule fashion. Below are the combination schemata we have been using paired rule-by-rule with their semantic interpretation.

Forward Application	$A/B : f \quad B : x \Rightarrow A : f(x)$	[FA]
Backward Application	$B : x \quad B\backslash A : f \Rightarrow A : f(x)$	[BA]
Lifting	$A : x \Rightarrow B/(A\backslash B) : \lambda y.yx$	[T]
Forward Composition	$A/B : f \quad B/C : g \Rightarrow A/C : \lambda x.f(gx)$	[B]
Backward Crossed Composition	$A/B : g \quad A\backslash C : f \Rightarrow C/B : \lambda x.f(gx)$	[B$_\times$]

Example 1.33. [Meaning Assembly in CCG] Given the lexical assignments of the labelled lexicon above, CCG builds the meaning of *which Sara wrote* as follows.

$$\dfrac{\dfrac{\text{which}}{(n\backslash n)/(s/np) : \lambda xyu.x(u) \wedge y(u)} \quad \dfrac{\dfrac{\dfrac{\text{Sara}}{np : \mathtt{sara}}}{s/(np\backslash s) : \lambda z.z(\mathtt{sara})}[\mathtt{T}] \quad \dfrac{\text{wrote}}{(np\backslash s)/np : \lambda yx.\mathtt{wrote}(x,y)}}{s/np : \lambda y.\mathtt{wrote}(\mathtt{sara}, y)}[\mathtt{B}]}{n\backslash n : \lambda yu.\mathtt{wrote}(\mathtt{sara}, u) \wedge y(u)}[\mathtt{FA}]$$

Note that in the derivation we have hidden the β-conversion rules.

Example 1.34. [Ambiguous Sentences] Let some, some' and every abbreviate the lambda terms from our labelled lexicon $\lambda x.\exists z \mathtt{student}(z) \wedge x(z)$, $\lambda xu.\exists z \mathtt{student}(z) \wedge x(u, z)$, and $\lambda x.\forall z \mathtt{student}(z) \rightarrow x(z)$, respectively. CCG builds the meaning of *every student knows some book* as following.

(i)

$$\dfrac{\dfrac{\text{every\_student}}{s/(np\backslash s) : \mathtt{every}} \quad \dfrac{\dfrac{\text{knows}}{(np\backslash s)/np : \mathtt{know}} \quad \dfrac{\text{some\_book}}{((np\backslash s)/np)\backslash(np\backslash s) : \mathtt{some}'}}{np\backslash s : \mathtt{some}'(\mathtt{know})}\,[\mathtt{BA}]}{s : \mathtt{every}(\mathtt{some}'(\mathtt{know}))}\,[\mathtt{FA}]$$

(ii)

$$\dfrac{\dfrac{\dfrac{\text{every\_student}}{s/(np\backslash s) : \mathtt{every}} \quad \dfrac{\text{knows}}{(np\backslash s)/np : \mathtt{know}}}{s/np : \lambda x.\mathtt{every}(\mathtt{know}\ x)}\,[\mathtt{B}] \quad \dfrac{\text{some\_book}}{(s/np)\backslash s : \mathtt{some}}}{s : \mathtt{some}(\lambda x.\mathtt{every}(\mathtt{know}\ x))}\,[\mathtt{BA}]$$

The derivation in (i) (resp. (ii)) gives the subject wide (resp. narrow) scope reading.

1.4.2 Curry-Howard Correspondence

In the Lambek calculus framework, syntactic rules are replaced by logical rules of inference. Therefore, the semantic rules are obtained deductively by exploiting the correspondence between proofs and terms. The famous Curry-Howard correspondence tells us that every proof in the natural deduction calculus for intuitionistic implicational logic can be encoded by a typed λ-term and *vice versa* [How80]. The categorial interpretation of derivations can be modelled directly on the Curry-Howard result, with the proviso that in the absence of structural rules in the categorial systems, the obtainable terms will be a sublanguage of the full λ-calculus.

Let us define the correspondence between the logical rules of NL and the application and abstraction rules of the lambda calculus. In a few words, the elimination of the functional connectives $\backslash$ and $/$ produces functional application terms, whereas the abstraction over variables corresponds to the introduction of the functional operators.

Definition 1.35. [Term Assignment for Natural Deduction] Let $\Gamma \vdash t : A$ stand for a deduction of the formula A decorated with the term t from a structured configuration of undischarged term-decorated assumptions Γ.

$$x : A \vdash x : A$$

$$\dfrac{\Gamma \vdash t : A/B \quad \Delta \vdash u : B}{\Gamma \circ \Delta \vdash t(u) : A}\,[/\mathrm{E}] \qquad \dfrac{(\Gamma \circ x : B) \vdash t : A}{\Gamma \vdash \lambda x.t : A/B}\,[/\mathrm{I}]$$

$$\dfrac{\Delta \vdash u : B \quad \Gamma \vdash t : B\backslash A}{\Delta \circ \Gamma \vdash t(u) : A}\,[\backslash\mathrm{E}] \qquad \dfrac{(x : B \circ \Gamma) \vdash t : A}{\Gamma \vdash \lambda x.t : B\backslash A}\,[\backslash\mathrm{I}]$$

The Lambek calculi are fragments of intuitionistic implicational logic [Abr90]. Consequently, the lambda terms computed by it form a fragment of the full language of lambda terms. First of all, since empty antecedents are not allowed and the Lambek calculi are resource sensitive, *viz.* each assumption is used exactly once, the system reasons about lambda terms with specific properties: (i) each subterm contains a free variable; and (ii) no multiple occurrences of the same variable are present. The latter could seem to be too strong constraint when thinking of linguistic applications. However, this is not the case as we will discuss at the end of this section (Example 1.42). A formal definition of the lambda calculus fragment corresponding to LP is given below[5].

Definition 1.36. [Fragment of the Lambda Terms for LP] Let $\Lambda(\mathsf{LP})$ be the largest $\mathsf{LAMBDA} \subseteq \mathsf{TERM}$ such that

i. each subterm of $\alpha \in \mathsf{LAMBDA}$ contains a free variable;

ii. no subterm of $\alpha \in \mathsf{LAMBDA}$ contains more than one free occurrence of the same variable;

iii. each occurrence of the λ abstractor in $\alpha \in \mathsf{TERM}$ binds a variable within its scope.

Derivations for the various Lambek calculi are all associated with LP term recipes. Therefore, we move from an isomorphism to a weaker correspondence. The correspondence between LP proofs and the lambda calculus was given in [Ben87a, Bus87, Wan92].

Theorem 1.37. Given an LP derivation of a sequent $A_1, \ldots, A_n \vdash B$ one can find a corresponding construction $\alpha_a \in \Lambda(\mathsf{LP})$, and conversely. A term $\alpha_a \in \Lambda(\mathsf{LP})$ is called a construction of a sequent $A_1, \ldots, A_n \vdash B$ iff α has exactly the free variable occurrences $x^1_{\mathtt{type}(A_n)}, \ldots, x^n_{\mathtt{type}(A_n)}$.

While introducing the lambda calculus we spoke of terms in normal forms. These terms are obtained proof theoretically by defining normal form derivations as following.

Definition 1.38. [Normal Form for Natural Deduction Derivations)] A derivation in natural deduction format is in *normal form* when there are no detours in it. A *detour* is formed when

i. a connective is introduced and immediately eliminated at the next step.

ii. an elimination rule is immediately followed by the introduction of the same connective.

The rules eliminating these two detours are called *reduction* rules.

Remark 1.39. The reductions of the detours in *i.* and in *ii.* correspond to β-reduction and η-reduction, respectively. Moreover, note that the above rewriting rules hold for all Lambek calculi, regardless of their structural rules.

[5] Again, for the sake of simplicity here we restrict attention to product-free Lambek calculi. See [Moo97] for the definition of the full systems.

By means of example, we give the reduction rule corresponding to η-reduction. The reader is referred to [Res00] for an extensive presentation of normalization.

$$\dfrac{\dfrac{[B \vdash x : B]^1 \quad \begin{array}{c} D_1 \\ \vdots \\ \Gamma \vdash t : B\backslash A \end{array}}{B, \Gamma \vdash t(x) : A}[\backslash \mathrm{E}]}{\Gamma \vdash \lambda x.t(x) : B\backslash A}[\backslash \mathrm{I}]^1 \qquad \text{rewrites to} \qquad \begin{array}{c} D_1 \\ \vdots \\ \Gamma \vdash t : B\backslash A \end{array}$$

in the lambda-calculus the reduction above corresponds to the rewrite rule $\lambda x.t(x) \Rightarrow_\eta t$ The correspondence between proofs and lambda terms is completed by the following theorem [Pra65, Gir87, GLT89].

Theorem 1.40. [Normalization] If $\mathcal{D}$ is a normal form derivation of $x_1 : A_1, \ldots x_n : A_n \vdash \alpha : C$, then α is in β, η normal form.

Let us now check how this framework accounts for the assembly of form-meaning pairs.

Starting from the labelled lexicon, the task for the Lambek derivational engine is to compute the lambda term representing the meaning assembly for a complex structure as a by-product of the derivation that establishes its grammaticality. The crucial distinction here is between the *derivational* meaning and the *lexical* meaning. The derivational meaning fully abstracts from lexical semantics: it is a general recipe for meaning assembly from assumptions of the given types.

Practically, one can proceed in two ways: (i) either one starts labeling the axioms of a derivation with the actual lambda terms assigned in the lexicon, or (ii) one labels the leaves of the derivation with variables, computes the proof term for the final structure and then replaces the variables by the actual lambda terms assigned in the lexicon to the basic constituents. We illustrate the two methods below in Examples 1.41 and 1.42, respectively.

Example 1.41. [Lifting] Starting from the type assignment Lori $\in np : \texttt{lori}$, one derives the higher order assignments as following:

$$\dfrac{\dfrac{\mathrm{Lori} \vdash np : \texttt{lori} \quad [np\backslash s \vdash np\backslash s : x]^1}{\mathrm{Lori} \circ np\backslash s \vdash s : x(\texttt{lori})}[\backslash \mathrm{E}]}{\mathrm{Lori} \vdash s/(np\backslash s) : \lambda x.x(\texttt{lori})}[/\mathrm{I}]^1 \qquad \dfrac{\dfrac{[s/np \vdash s/np : x]^1 \quad \mathrm{Lori} \vdash np : \texttt{lori}}{s/np \circ \mathrm{Lori} \vdash s : x(\texttt{lori})}[/\mathrm{E}]}{\mathrm{Lori} \vdash (s/np)\backslash s : \lambda x.x(\texttt{lori})}[\backslash \mathrm{I}]^1$$

First of all, note how the system assigns a variable to the hypothesis. The latter is discharged by means of [/I] (or [\I]) which corresponds to the abstraction over the variable. Moreover, note that the higher order types in the two derivations are different, but they correspond to the same lambda terms, *i.e.* the two structures are correctly assigned the same meaning.

This example shows how in the CTL framework, the assembly of meaning is a byproduct of the proof theoretical analysis. In particular, the type-lifting, stipulated in the Montagovian tradition and explicitly expressed by the [T] combinator in CCG, is obtained simply by means of logical rules. See [Oeh99] for a discussion about the advantages of having the lifting as a derivable theorem in the system.

The relative clause examples in our toy fragment offer a nice illustration of the division of labor between lexical and derivational semantics. Intuitively, a relative pronoun has to compute the intersection of two properties: the common noun property obtained from the n that is modified, and the property obtained from the body of the relative clause, a sentence with a np hypothesis missing. In the logical form, this would come down to binding two occurrences of a variable by one λ binder. On the level of *derivational* semantics, one cannot obtain this double binding: the Lambek systems are resource sensitive, which means that every assumption is used exactly once. But on the level of *lexical* semantics, we can overcome this expressive limitation (which is syntactically well-justified!) by assigning the relative pronoun a double-bind term as its lexical meaning recipe: which $\in (n\backslash n)/(s/np) : \lambda xyz.x(z) \wedge y(z)$. In this way, we obtain the proper recipe for the relative clause *which Sara wrote*, namely $\lambda yz.\texttt{wrote}(\texttt{Sara}, z) \wedge y(z)$, as shown below.

Example 1.42. [Relative Clause]

$$\dfrac{\text{which} \vdash (n\backslash n)/(s/np) : X_4 \qquad \dfrac{\dfrac{\dfrac{\text{Sara} \vdash np : X_3 \qquad \dfrac{\text{wrote} \vdash (np\backslash s)/np : X_1 \quad [x \vdash np : X_2]^1}{\text{wrote} \circ x \vdash np\backslash s : X_1X_2}\,[/E]}{\text{Sara} \circ (\text{wrote} \circ x) \vdash s : (X_1X_2)X_3}\,[\backslash E]}{(\text{Sara} \circ \text{wrote}) \circ x \vdash s : (X_1X_2)X_3}\,[\text{ass}]}{\text{Sara} \circ \text{wrote} \vdash s/np : \lambda X_3.(X_1X_2)X_3}\,[/I]^1}{\text{which} \circ (\text{Sara} \circ \text{wrote}) \vdash n\backslash n : X_4(\lambda X_3.(X_1X_2)X_3)}\,[/E]$$

Note that the structural rules do not effect the meaning assembly. By replacing the variables $X_1, \ldots, X_4$ with the corresponding lexical assignments, and applying the reduction rules, one obtains the proper meaning of the analyzed structure.

1.5 Key Concepts

The main points of this chapter to be kept in mind are the following:

1. Linguistic signs are pairs of form and meaning, and composed phrases are structures rather than strings.
2. When employing a logic to model linguistic phenomena, grammatical derivations are seen as theorems of the grammatical logic.
3. The correspondence between proofs and natural language models, via the lambda terms, properly accounts for the natural language syntax semantics interface.