

Chapter 4

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.

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 these course notes, we introduce the main aspects of CTL (Section 1.2). Moreover, we discuss the proof theoretical perspective on form-meaning assembly of linguistic expressions (Section 1.3).

4.1 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 this 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.

4.1.1 Classical Categorical Grammar

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

DEFINITION 4.1. [Type Language and Rules of CG] The language of CG is recursively built over atomic categories by means of the category forming operators \backslash 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 $A \in \text{ATOM}$, then $A \in \text{CAT}$;
- ii.* if A and $B \in \text{CAT}$, then A/B and $B \backslash A \in \text{CAT}$.

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

$$\begin{array}{l} A/B, B \Rightarrow A \quad [\text{FA}] \\ B, B \backslash A \Rightarrow A \quad [\text{BA}]. \end{array}$$

[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 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 4.2. [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:

$$\begin{array}{l} A \Rightarrow A \quad [\text{id}] \\ \Gamma, A, \Gamma' \Rightarrow B \text{ and } \Delta \Rightarrow A, \text{ then } \Gamma, \Delta, \Gamma' \Rightarrow B. \quad [\text{cut}] \end{array}$$

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.* $\text{LEX} \subseteq \Sigma \times \text{CAT}$. We say that G generates a string $w_1 \dots w_n \in \Sigma^+$ as an expression of category B if and only if there are categories A_1, \dots, A_n such that $(w_i, A_i) \in \text{LEX}$ and $A_1, \dots, A_n \Rightarrow B$. $L(G)$, 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 combinatorial 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 4.3. [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 4.4. [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	$np\s$
knows, wrote, wears	$(np\s)/np$	some, every, one, any, no	$(s/(np\s))/n$
which, who	$(n\n)/ (np\s)$	there, yet, already	$(np\s)\(np\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 $n\n$. 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.

$$\text{Sara wears the new dress} \in s? \rightsquigarrow np, (np\s)/np, np/n, n/n, n \Rightarrow s?$$

$$\frac{\frac{\text{Sara}}{np} \quad \frac{\frac{\text{wears}}{(np \setminus s)/np} \quad \frac{\frac{\text{the}}{np/n} \quad \frac{\frac{\text{new}}{n/n} \quad \frac{\text{dress}}{n}}{n}}{n}}{np}}{np \setminus s} \quad [\text{FA}]}{s} \quad [\text{BA}]$$

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

$$\frac{\frac{\text{who}}{(n \setminus n)/(np \setminus s)} \quad \frac{\frac{\text{knows}}{(np \setminus s)/np} \quad \frac{\text{Lori}}{np}}{np \setminus s} \quad [\text{FA}]}{n \setminus n} \quad [\text{FA}]$$

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 $(n \setminus n)/(np \setminus s)$ 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 $(np \setminus s)/np$, the assignment $((n \setminus n)/tv)/np$ 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 $np \setminus s$, the determiners *some, every, one, any, no* could be assigned a second type $(tv \setminus iv)/n$ 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 Categorical Grammar, developed by Mark Steedman, offers the most elaborate proposal for this strategy.

4.1.2 Combinatory Categorical Grammar

For an detailed exposition of Combinatory Categorical Grammar (CCG), we refer the reader to [Ste00, Bal02]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].

¹In 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 been 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	$A \Rightarrow B/(A \setminus B)$	[T]
Forward Composition	$A/B, B/C \Rightarrow A/C$	[B]
Backward Crossed Composition	$A/B, A \setminus C \Rightarrow C/B$	[B _×]

EXAMPLE 4.5. [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*: $(n \setminus n)/(s/np)$. 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 $s/(np \setminus s)$ which can then combine with the transitive verb by means of Forward Composition.

$$\frac{\frac{\text{which}}{(n \setminus n)/(s/np)} \quad \frac{\frac{\text{Sara}}{np} \quad \frac{\text{wrote}}{(np \setminus s)/np}}{s/(np \setminus s)} \quad [T]}{s/np} \quad [B]}{n \setminus n} \quad [FA]$$

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 [B_×] as illustrated below.

$$\frac{\frac{\text{which}}{(n \setminus n)/(s/np)} \quad \frac{\frac{\text{Sara}}{s/(np \setminus s)} \quad \frac{\frac{\text{wrote}}{(np \setminus s)/np} \quad \frac{\text{there}}{(np \setminus s) \setminus (np \setminus s)}}{(np \setminus s)/np}}{s/np}}{n \setminus n} \quad [FA]}{n \setminus n} \quad [B_{\times}]$$

EXAMPLE 4.6. [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 $np \setminus s$ as a result. A derivation is given in (i) below. In CCG, there is a second option for typing the direct object: $(s/np) \setminus s$. 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]. By way of illustration, we present the derivations of (5-b) in (i) and (ii), the reader is referred to [Ste00] for a proper analysis of quantifiers within CCG. In the discussion of meaning assembly in Section 1.3, we will come back to these two options for object generalized quantifiers.

$$\begin{array}{c}
 \text{(i)} \\
 \frac{\frac{\text{every_student}}{s/(np \setminus s)} \quad \frac{\frac{\text{knows}}{(np \setminus s)/np} \quad \frac{\text{some_book}}{((np \setminus s)/np) \setminus (np \setminus s)}}{np \setminus s}}{s} \quad [\text{FA}] \quad [\text{BA}]
 \end{array}$$

$$\begin{array}{c}
 \text{(ii)} \\
 \frac{\frac{\text{every_student}}{s/(np \setminus s)} \quad \frac{\text{knows}}{(np \setminus s)/np}}{s/np} \quad [\text{B}] \quad \frac{\text{some_book}}{(s/np) \setminus s}}{s} \quad [\text{BA}]
 \end{array}$$

Let us evaluate the CCG strategy. We notice first of all that a combinator like $[B_{\times}]$, 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? (See [Bal02] for an answer to the last question within the CCG framework.) These questions will be addressed in the next two sections.

4.2 A Logic of Types

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. (See Chapter 3 for the formal details.)

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

Table 4.1: Rules-based approach vs. logical approach.

4.2.1 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) presentation², which is closer to the format we have used in the previous sections, is the Natural Deduction (N.D.) format.

DEFINITION 4.7. [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:

$$\begin{array}{c}
 \overline{A \vdash A} \text{ [axiom]} \\
 \\
 \frac{\Delta \vdash B/A \quad \Gamma \vdash A}{\Delta, \Gamma \vdash B} \text{ [/E]} \quad \frac{\Gamma \vdash A \quad \Delta \vdash A \setminus B}{\Gamma, \Delta \vdash B} \text{ [\E]} \\
 \\
 \frac{\Delta, B \vdash C}{\Delta \vdash C/B} \text{ [/I]} \quad \frac{B, \Delta \vdash C}{\Delta \vdash B \setminus C} \text{ [\I]}
 \end{array}$$

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 ($B \setminus C$), 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 4.8. [Semigroup Interpretation]

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

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

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 4.9. [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 one³, is a typical application of hypothetical reasoning. Its derivation is illustrated below.

$$\frac{\Delta \vdash A \quad [(A \setminus B) \vdash (A \setminus B)]^1}{\frac{\Delta, (A \setminus B) \vdash B}{\Delta \vdash B / (A \setminus B)} [I]^1} [\setminus E]$$

The derivation proves that if a structure Δ is of type A , then it is of type $B / (A \setminus B)$ as well. The proof is given by hypothetical reasoning: Assume a structure of type $A \setminus B$, given $\Delta \vdash A$, then Δ composed with $A \setminus B$ 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:

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

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.

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

EXAMPLE 4.10. [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 $n \setminus n$ as follows.

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

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

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

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 [B_x], 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 Pentus [Pen93] that L grammars are context free provides the formal underpinnings for this claim.

4.2.2 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 $A \in \mathbf{FORM}$, then $A \in \mathbf{STRUCT}$;
- ii.* If Γ and $\Delta \in \mathbf{STRUCT}$, then $(\Gamma \circ \Delta) \in \mathbf{STRUCT}$.

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(\diamond)**, because it is obtained from **L** by dropping associativity.

DEFINITION 4.12. [The Lambek Family]. **Logical rules** for the base logic **NL(\diamond)**:

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

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

$$\frac{(\Delta \circ B) \vdash C}{\Delta \vdash C/B} \text{ [/I]} \quad \frac{(B \circ \Delta) \vdash C}{\Delta \vdash B \setminus C} \text{ [\setminus I]}$$

As we have shown in Chapter 3, the algebraic structure interpreting these operators has space for n-ary operators, and in particular for the unary ones studied in [Moo96b, Moo97]. The non associative Lambek calculus extended with these operators is usually referred to as **NL(\diamond)** and is obtained by adding to **NL** the logical rules below.

$$\frac{\Gamma \vdash t \square \downarrow A}{\langle \Gamma \rangle \vdash A} \text{ [\square \downarrow E]} \quad \frac{\langle \Gamma \rangle \vdash A}{\Gamma \vdash \square \downarrow A} \text{ [\square \downarrow I]}$$

$$\frac{\Delta \vdash \diamond A \quad \Gamma[\langle A \rangle] \vdash B}{\Gamma[\Delta] \vdash B} \text{ [\diamond E]} \quad \frac{\Gamma \vdash A}{\langle \Gamma \rangle \vdash \diamond A} \text{ [\diamond I]}$$

The relation between the unary structural operator, $\langle \cdot \rangle$, and the unary logical operator, \diamond , is the same one holding between the binary structural and logical operator (\circ and \bullet , respectively). This can be clearly seen by observing rule $[\diamond I]$ where the introduction of the \diamond in the right side (i.e. the logical side) of the sequent corresponds to the introduction of $\langle \cdot \rangle$ in the left side (i.e. the structural side). This correspondence explains also the $[\diamond E]$ rule, which performs a cut (within a structure Γ of the formula A headed by the diamond operator (where the latter is a logical when occurs in the right side of the sequent and a structural diamond when it's on its left side). In the remaining part of this chapter we will try to give more linguistic intuition on the role of the unary structural operator and of its interplay with its residual (the $\square \downarrow$). As for now, notice that relation between them is established by means of the $[\square \downarrow E]$ and $[\square \downarrow I]$.

In **CTL**, unary operators have been used mainly in two ways: to control structural reasoning and to encode partial orders among elements of the same domain of interpretation. In this Chapter we will focus attention on the first use by briefly reviewing the literature, whereas in Chapter 5 we will describe the partial ordering strategy.

Structural rules. Let us write $\Gamma[\Delta]$ for a structure Γ containing 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**.

$$\frac{\frac{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash C}{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash C} [\text{ass}] \quad \frac{\Gamma[(\Delta_2 \circ \Delta_1)] \vdash C}{\Gamma[(\Delta_1 \circ \Delta_2)] \vdash C} [\text{per}]}{\frac{\Gamma[\langle \Delta_1 \circ \Delta_2 \rangle] \vdash C}{\Gamma[\langle \Delta_1 \rangle \circ \langle \Delta_2 \rangle] \vdash C} \text{dis}_\diamond}$$

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** (introduced in Chapter 3), 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 $(\backslash_i, \bullet_i/i)$ and $(\diamond_i, \square_i^\downarrow)$ and \circ_i and \rangle_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 $\Gamma[\Delta \circ_2 \Delta'] \vdash A$; if $\Gamma[\langle \Delta \rangle_1] \vdash A$ then $\Gamma[\langle \Delta \rangle_2] \vdash A$
- ii. Interaction* structural rules which mix distinct modes and operators.

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 .

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

EXAMPLE 4.13. [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.

$$\begin{array}{c}
\frac{\text{wrote} \vdash (np \setminus s) / {}_a np \quad [np \vdash np]^1}{\text{wrote} \circ_a np \vdash np \setminus s} \quad [/_a E] \quad \text{there} \vdash (np \setminus s) \setminus (np \setminus s) \quad [\setminus E] \\
\frac{\text{Sara} \vdash np \quad ((\text{wrote} \circ_a np) \circ \text{there}) \vdash np \setminus s}{\text{Sara} \circ ((\text{wrote} \circ_a np) \circ \text{there}) \vdash s} \quad [\setminus E] \\
\frac{\text{Sara} \circ ((\text{wrote} \circ_a np) \circ \text{there}) \vdash s}{\text{Sara} \circ ((\text{wrote} \circ \text{there}) \circ_a np) \vdash s} \quad [\text{diss}] \\
\frac{\text{Sara} \circ ((\text{wrote} \circ \text{there}) \circ_a np) \vdash s}{(\text{Sara} \circ (\text{wrote} \circ \text{there})) \circ_a np \vdash s} \quad [\text{mixass}] \\
\frac{\text{which} \vdash (n \setminus n) / (s / {}_a np) \quad (\text{Sara} \circ (\text{wrote} \circ \text{there})) \circ_a np \vdash s}{\text{which} \circ (\text{Sara} \circ (\text{wrote} \circ \text{there})) \vdash n \setminus n} \quad [/_a I]^1 \quad [/ E]
\end{array}$$

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 $[B_{\times}]$. 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: $(np \setminus s) / {}_a np$ in contrast with the linguistic facts.

Suppose now, we have the interaction structural rule below, in [Moo99], it is shown that they perform the required task, together with a lexical type assignment $(n \setminus n) / (s / \diamond \square \downarrow np)$.

$$\frac{\Gamma[\Delta_1 \circ (\Delta_2 \circ \langle \Delta_3 \rangle)] \vdash C}{\Gamma[(\Delta_1 \circ \Delta_2) \circ \langle \Delta_3 \rangle] \vdash C} \quad [\text{ass}_{\diamond}] \quad \frac{\Gamma[(\Delta_1 \circ \langle \Delta_3 \rangle) \circ \Delta_2] \vdash C}{\Gamma[(\Delta_1 \circ \Delta_2) \circ \langle \Delta_3 \rangle] \vdash C} \quad [\text{diss}_{\diamond}] \quad (4.1)$$

Note, that these rules are available only for marked formulas, where the latter are introduced only due to information stored in the lexical entries. Let us first look at the derivation of (4-b) where the extraction is performed from a peripheral position.

EXAMPLE 4.14. [Right-Peripheral Extraction in $NL(\diamond)$ plus $[\text{ass}_{\diamond}]$]

$$\begin{array}{c}
\frac{\text{wrote} \vdash (np \setminus s) / np \quad \frac{[\square \downarrow np \vdash \square \downarrow np]^1}{\langle \square \downarrow np \rangle \vdash np} \quad [\square \downarrow E]}{\text{wrote} \circ \langle \square \downarrow np \rangle \vdash np \setminus s} \quad [/ E] \\
\frac{\text{Sara} \vdash np \quad (\text{wrote} \circ \langle \square \downarrow np \rangle) \vdash np \setminus s}{\text{Sara} \circ (\text{wrote} \circ \langle \square \downarrow np \rangle) \vdash s} \quad [\setminus E] \\
\frac{[\diamond \square \downarrow np \vdash \diamond \square \downarrow np]^2 \quad (\text{Sara} \circ \text{wrote}) \circ \langle \square \downarrow np \rangle \vdash s}{(\text{Sara} \circ \text{wrote}) \circ \diamond \square \downarrow np \vdash s} \quad [\text{ass}_{\diamond}] \quad [\diamond E]^1 \\
\frac{\text{which} \vdash (n \setminus n) / (s / \diamond \square \downarrow np) \quad (\text{Sara} \circ \text{wrote}) \circ \diamond \square \downarrow np \vdash s}{\text{which} \circ (\text{Sara} \circ \text{wrote}) \vdash n \setminus n} \quad [/ I]^2 \quad [/ E]
\end{array}$$

Note how the re-bracketing is controlled by the pronoun type assignment which requires a sentence missing a noun phrase occurring in a special position $\diamond\Box^{\downarrow}np$. This forces the assumption of a marked noun phrase $\Box^{\downarrow}np$. This marker is then passed from the logical to the structural language by means of $[\Box^{\downarrow}E]$, to allow the required re-bracketing $[ass_{\diamond}]$. However, abstraction can take place only from atomic structural formulas (*i.e.* logical formulas). Therefore, once $\langle\Box^{\downarrow}np\rangle$ has fulfilled its task on the structural level, it is replaced by the corresponding logical formula by means of $[\diamond E]^1$. This rule substitutes the co-indexed hypothesis with a second one, which is finally discharged building the type suitable for the pronoun.

The interaction structural rule $[ass_{\diamond}]$ required by the derivation of right-branch extraction in peripheral position in itself is not enough to express the proper structural generalization of wh-dependencies in English. In particular, this structural rule does not help deriving (4-c). To account for right-branch extraction from a non-peripheral position one needs the interaction structural rule $[diss_{\diamond}]$ as well.

EXAMPLE 4.15. [Non-Peripheral Extraction in $NL(\diamond)$ plus $[ass_{\diamond}]$ and $[diss_{\diamond}]$]

$$\begin{array}{c}
\frac{[\Box^{\downarrow}np \vdash \Box^{\downarrow}np]^1}{\vdots} \\
\frac{\frac{\frac{Sara \circ ((wrote \circ \langle\Box^{\downarrow}np\rangle) \circ there) \vdash s}{Sara \circ ((wrote \circ there) \circ \langle\Box^{\downarrow}np\rangle) \vdash s} [diss_{\diamond}]}{[\diamond\Box^{\downarrow}np \vdash \diamond\Box^{\downarrow}np]^2} (Sara \circ (wrote \circ there)) \circ \langle\Box^{\downarrow}np\rangle \vdash s}{(Sara \circ (wrote \circ there)) \circ \diamond\Box^{\downarrow}np \vdash s} [ass_{\diamond}] \\
\frac{\frac{which \vdash wh/(s/\diamond\Box^{\downarrow}np)}{Sara \circ (wrote \circ there) \vdash s/\diamond\Box^{\downarrow}np} [I]^2}{which \circ (Sara \circ (wrote \circ there)) \vdash n \setminus n} [E]
\end{array}$$

This derivation can be read in a similar way than the previous one. The only difference is the application of $[diss_{\diamond}]$ which brings the hypothesis in a peripheral position.

These examples are meant only as an illustration of the method used in CTL to account for long distance phenomena. A detailed discussion can be found in [Moo99], where unary operators and structural reasoning are also exploited to deal with crosslinguistic variations. In particular, the lexical type assignments and structural packages required to model English relative clauses are compared with the ones required by subject-object-verb language like Dutch. In a few words, the structural variation between the two languages with respect to relative clauses is captured by combining a universal base logic with different structural packages.⁴

The categorial account of structural control is surprisingly close to ‘feature-driven’ structural reasoning in generative grammar, especially within the minimalism program

⁴See [Bal02] for a detailed analysis of long distance dependencies in CCG extended with modes. The reader is invited to compare the two approaches for an understanding of their differences and similarities. The multi-modal CCG is the topic of the course given at ESSLLI’04 by Geert-Jan Kruijff, further references to this work will be provided there.

[Cho95] formalized in [Sta97]. For some discussion of this correspondence and a study of the connection between Stabler’s Minimalist Grammar and CTL see [Ber02] and [Ver99], respectively.

The example above illustrates 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 4.16. [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.

$$\begin{array}{c}
 \text{(i)} \\
 \frac{a \vdash qp/n \quad \frac{\text{new} \vdash n/n \quad \text{dress} \vdash n}{\text{new} \circ \text{dress} \vdash n} [/\text{E}]}{a \circ (\text{new} \circ \text{dress}) \vdash qp} [/\text{E}]}{a \circ (\text{dress} \circ \text{new}) \vdash qp} [\text{per}\bullet]^*
 \end{array}
 \qquad
 \begin{array}{c}
 \text{(ii)} \\
 \frac{\text{un} \vdash qp/n \quad \frac{\text{nuovo} \vdash n/cn \quad \text{vestito} \vdash n}{\text{nuovo} \circ_c \text{vestito} \vdash n} [/_c\text{E}]}{\text{un} \circ (\text{nuovo} \circ_c \text{vestito}) \vdash qp} [/\text{E}]}{\text{un} \circ (\text{vestito} \circ_c \text{nuovo}) \vdash qp} [\text{per}\bullet]
 \end{array}$$

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).

Finally, unary operators have also been employed to encode morphological information. The type logical analysis of morphological agreement worked out by Heylen [Hey99] provides a categorial alternative for the unification and subsumption based approach of framework like Head Driven Phrase Structure Grammar, and Lexicalized Functional Grammar. In order to recast their mechanisms in logical terms, underspecification and

overspecification are expressed via inclusion postulates and the law of residuation is exploited to account for the subsumption relation among expressions of the same syntactic category. Moreover, interaction postulates involving unary and binary operators govern the way information is distributed through phrase structure. In our exposition of the linguistic applications of the unary operators, Heylen's proposal is of particular interest for the use of the inclusion postulates and more generally for the application of inclusion relations among types.

The following postulates exemplify the encoding of underspecification. We indicate the modes as indexes, where 'pl' and 'sg' stand for 'plural' and 'singular', respectively and 'num' identifies underspecification.

Inclusion Postulates

$$[\text{PL}] \diamond_{\text{num}}A \longrightarrow \diamond_{\text{pl}}A \quad \text{and} \quad [\text{SG}] \diamond_{\text{num}}A \longrightarrow \diamond_{\text{sg}}A$$

These postulates can be read as saying that a phrase of syntactic category A underspecified for its number, $\diamond_{\text{num}}A$, could be either plural $\diamond_{\text{pl}}A$, or singular, $\diamond_{\text{sg}}A$. The alternative presentation with structural rules is given below.

$$\frac{\Gamma[\langle \Delta \rangle^{\text{pl}}] \vdash C}{\Gamma[\langle \Delta \rangle^{\text{num}}] \vdash C} [\text{pl}] \quad \frac{\Gamma[\langle \Delta \rangle^{\text{sg}}] \vdash C}{\Gamma[\langle \Delta \rangle^{\text{num}}] \vdash C} [\text{sg}]$$

Morphological agreement is required, for instance, for the combination of the definite article with its noun in Italian. Differently from English, Italian uses definitive articles sensitive to the number of the noun they combine with, e.g. *i pomodori* (tr. the tomatoes) is correct, whereas *i pomodoro* (tr. the tomato) is not. On the other hand, transitive verbs are underspecified regarding the number of their object, e.g. both *il gatto mangia i pomodori* (tr. the cat eats the tomatoes) and *il gatto mangia il pomodoro* (tr. the cat eats the tomato) are correct Italian sentences. When encoding morphological information into lexical assignments, the relation holding among expressions of the same syntactic category but with different morphological properties, must be taken into account. For this specific case, it must be stated that the expression taken as argument by the transitive verb can be either plural or singular. In [Hey99], this information would be expressed by the unary operators labelling the lexical type assignments as shown by the example below.

$$\begin{array}{ll} il & \in \square_{\text{sg}}^{\downarrow} np / \square_{\text{sg}}^{\downarrow} n & pomodori & \in \square_{\text{pl}}^{\downarrow} n \\ i & \in \square_{\text{pl}}^{\downarrow} np / \square_{\text{pl}}^{\downarrow} n & mangia & \in (\square_{\text{sg}}^{\downarrow} np \setminus s) / \square_{\text{num}}^{\downarrow} np \end{array}$$

The type assignment of the article i (resp. il) specifies that it combines with a plural (resp. singular) noun, to give a plural (resp. singular) noun phrase, whereas the verb $mangia$ is sensitive to the number of its subject, but is underspecified for the number of the noun phrase taken as object.

The assembly of the plural article i with the singular noun $pomodoro$ is blocked simply by the mismatch of their types. On the other hand, the possibility of the verb $mangia$ to combine both with the plural and singular noun phrases $i pomodori$ and il

pomodoro, is carried out by means of the inclusion relations [pl] and [sg]. In a top-down reading the derivation below can be read as saying that a structure which is specified for its number can also be underspecified if required by the constituent it composes with.

$$\frac{\frac{\frac{i \circ pomodori \vdash \square_{pl}^\downarrow np}{\langle i \circ pomodori \rangle^{pl} \vdash np} [\square_{pl}^\downarrow E]}{\langle i \circ pomodori \rangle^{num} \vdash np} [pl]}{\frac{mangia \vdash (\square_{sg}^\downarrow np \setminus s) / \square_{num}^\downarrow np \quad i \circ pomodori \vdash \square_{num}^\downarrow np}{mangia \circ (i \circ pomodori) \vdash \square_{sg}^\downarrow np \setminus s} [\square_{num}^\downarrow I]} [/E]}$$

Note how once again the residuation law makes possible a division of labor between the logical and structural languages; the former takes care of feature checking, whereas the subsumption relation is checked by the latter. Moreover, notice how in the enforcing of the agreement relation a crucial role is played by the monotonicity of the binary operators. This can be better understood by abstracting away from the details of the derivation and looking at the general schema below. Let $C \longrightarrow B$, then in natural deduction there is a derivation from $\Gamma \vdash C$ to $\Gamma \vdash B$, therefore

$$\frac{\Delta \vdash A/B \quad \begin{array}{c} \Gamma \vdash C \\ \vdots \\ \Gamma \vdash B \end{array}}{\Delta \circ \Gamma \vdash A} [/E]$$

Put differently, one could say that since $/$ is downward monotone in its second argument position, a structure of type A/B will combine with any structure of a type C smaller than or equal to B .

4.3 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 4.17. [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 $a_1, \dots, a_m \in A$ holds $m(f(a_1, \dots, a_m)) = m'(f)(m(a_1), \dots, m(a_m))$.

4.3.1 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 4.18. [Types] Given a non-empty set of *basic types* \mathbf{Base} , the set of types \mathbf{TYPE} is the smallest set such that

- i. $\mathbf{Base} \subseteq \mathbf{TYPE}$;
- ii. $(a, b) \in \mathbf{TYPE}$, if a and $b \in \mathbf{TYPE}$.

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

DEFINITION 4.19. [Categories and Types] Let us define a function $\mathbf{type} : \mathbf{CAT} \rightarrow \mathbf{TYPE}$ which maps syntactic categories to semantic types.

$$\begin{aligned} \mathbf{type}(np) &= e; & \mathbf{type}(A/B) &= (\mathbf{type}(B), \mathbf{type}(A)); \\ \mathbf{type}(s) &= t; & \mathbf{type}(B \setminus A) &= (\mathbf{type}(B), \mathbf{type}(A)); \\ \mathbf{type}(n) &= (e, t). \end{aligned}$$

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 4.20. [Typed λ -terms] Let \mathbf{VAR}_a be a countably infinite set of *variables* of type a and \mathbf{CON}_a a collection of *constants* of type a . The set \mathbf{TERM}_a of λ -terms of type a is defined by mutual recursion as the smallest set such that the following holds:

- i. $\mathbf{VAR}_a \subseteq \mathbf{TERM}_a$,
- ii. $\mathbf{CON}_a \subseteq \mathbf{TERM}_a$,
- iii. $(\alpha(\beta)) \in \mathbf{TERM}_a$ if $\alpha \in \mathbf{TERM}_{(a,b)}$ and $\beta \in \mathbf{TERM}_b$,
- iv. $\lambda x. \alpha \in \mathbf{TERM}_{(a,b)}$, if $x \in \mathbf{VAR}_a$ and $\alpha \in \mathbf{TERM}_b$.

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

- i. $\mathbf{Free}(x_b) = \{x_b\}$ if $x_b \in \mathbf{VAR}_b$,
- ii. $\mathbf{Free}(c_b) = \{\}$ if $c_b \in \mathbf{CON}_b$,
- iii. $\mathbf{Free}(\alpha_{(a,b)}(\beta_a)) = \mathbf{Free}(\alpha_{(a,b)}) \cup \mathbf{Free}(\beta_a)$,
- iv. $\mathbf{Free}(\lambda x_a. \alpha_b) = \mathbf{Free}(\alpha_b) - \{x_a\}$.

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 4.22. [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 4.23. [Frame] A *frame* D consists of the collection of basic domains, *i.e.* $\cup_{\alpha \in \text{Base}} \text{Dom}_\alpha$ and the domains for functional types. The latter are as follows

$$\text{Dom}_{(a,b)} = \text{Dom}_b^{\text{Dom}_a} = \{f \mid f : \text{Dom}_a \rightarrow \text{Dom}_b\}.$$

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:

$$\text{Dom}_e = E \quad \text{and} \quad \text{Dom}_t = \{1, 0\}.$$

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 4.24. [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 $I : \text{Lexicon} \rightarrow D$, s.t. if α is of type a , $I(\alpha) \in \text{Dom}_a$.

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

DEFINITION 4.25. [Denotation] The *denotation* $\llbracket \alpha_a \rrbracket_{\mathcal{M}}^f$ of a λ -term α_a with respect to the model $\mathcal{M} = \langle D, I \rangle$ and assignment f , where $f : \text{VAR}_a \rightarrow \text{Dom}_a$, is given by

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

where $f[x_a := d]$ stands for the assignment that maps x_a to $d \in \text{Dom}_a$ 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 4.26. 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 $B \setminus A$ and, therefore, their meanings are in the semantic type $(\text{type}(B), \text{type}(A))$ and are interpreted in the domain $\text{Dom}_{(b,a)}$.

4.3.2 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 4.27. [Term Labelled Lexicon] Given a set of basic expressions of a natural language Σ , a term labeled categorical lexicon is a relation,

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

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

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

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 4.29. [Relational Interpretation of Non-Logical Constants] Let our model be based on the set of entities $E = \{\text{lori, ale, sara, pim}\}$ 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 $\llbracket w \rrbracket$ indicate the interpretation of w :

$\llbracket \text{sara} \rrbracket$	=	sara;
$\llbracket \text{pim} \rrbracket$	=	pim;
$\llbracket \text{lori} \rrbracket$	=	lori;
$\llbracket \text{know} \rrbracket$	=	$\{\langle \text{lori, ale} \rangle, \langle \text{ale, lori} \rangle, \langle \text{sara, lori} \rangle,$ $\langle \text{lori, lori} \rangle, \langle \text{ale, ale} \rangle, \langle \text{sara, sara} \rangle, \langle \text{pim, pim} \rangle\};$
$\llbracket \text{student} \rrbracket$	=	$\{\text{lori, ale, sara}\};$
$\llbracket \text{professor} \rrbracket$	=	$\{\text{pim}\};$
$\llbracket \text{tall} \rrbracket$	=	$\{\text{lori, pim}\}.$

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.⁵

The interpretation of complex phrases is obtained by interpreting the corresponding lambda terms. For example, if $\text{walk} \in \text{CON}_{(e,t)}$ and $x \in \text{VAR}_e$, then $\text{walk}(x)$ expresses the fact that x has the property of walking, whereas $\lambda x.\text{walk}(x)$ is an abstraction over

⁵Consequently, the two notations $y(z)(u)$ and $y(u, z)$ are equivalent.

x and it represents the property itself. Moreover, due to the reduction rules of the lambda calculus the constant $\mathbf{walk}_{(e,t)}$ is equivalent to the term $\lambda x_e.(\mathbf{walk}(x))_t$. Applying Definition 1.25 this term is denoted by a function g such that for each entity $d \in Dom_e$, gives $g(d) = 1$ iff $\llbracket \mathbf{walk}(x) \rrbracket_{\mathcal{M}}^{f[x:=d]} = 1$, or in other words iff x has the property expressed by \mathbf{walk} .

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

EXAMPLE 4.30. [Logical Constants] By evaluating the lambda expressions in Example 1.28 in a model, one obtains the interpretations below:

$$\begin{aligned} \llbracket \mathbf{no\ N} \rrbracket &= \{X \subseteq E \mid \llbracket \mathbf{N} \rrbracket \cap X = \emptyset\}. \\ \llbracket \mathbf{some\ N} \rrbracket &= \{X \subseteq E \mid \llbracket \mathbf{N} \rrbracket \cap X \neq \emptyset\}. \\ \llbracket \mathbf{every\ N} \rrbracket &= \{X \subseteq E \mid \llbracket \mathbf{N} \rrbracket \subseteq X\}. \\ \llbracket \mathbf{N\ which\ VP} \rrbracket &= \llbracket \mathbf{N} \rrbracket \cap \llbracket \mathbf{VP} \rrbracket. \end{aligned}$$

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 5.

4.4 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.

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

EXAMPLE 4.31. [Meaning Assembly in CCG] Given the lexical assignments of the labelled lexicon above, CCG builds the meaning of *which Sara wrote* as follows.

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

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

EXAMPLE 4.32. [Ambiguous Sentences] Let **some**, **some'** and **every** abbreviate the lambda terms from our labelled lexicon $\lambda x.\exists z\mathbf{student}(z) \wedge x(z)$, $\lambda x u.\exists z\mathbf{student}(z) \wedge x(u, z)$, and $\lambda x.\forall z\mathbf{student}(z) \rightarrow x(z)$, respectively. Considered the toy-grammar of CCG we have given above by way of illustration, the meaning of *every student knows some book* is built as following.

$$\begin{array}{c}
 \text{(i)} \\
 \frac{\frac{\text{every_student}}{s/(np \setminus s) : \mathbf{every}} \quad \frac{\frac{\text{knows}}{(np \setminus s)/np : \mathbf{know}} \quad \frac{\text{some_book}}{((np \setminus s)/np) \setminus (np \setminus s) : \mathbf{some}'}}{np \setminus s : \mathbf{some}'(\mathbf{know})} \text{ [FA]}}{s : \mathbf{every}(\mathbf{some}'(\mathbf{know}))} \text{ [BA]} \\
 \\
 \text{(ii)} \\
 \frac{\frac{\frac{\text{every_student}}{s/(np \setminus s) : \mathbf{every}} \quad \frac{\text{knows}}{(np \setminus s)/np : \mathbf{know}}}{s/np : \lambda x.\mathbf{every}(\mathbf{know} x)} \text{ [B]} \quad \frac{\text{some_book}}{(s/np) \setminus s : \mathbf{some}} \text{ [BA]}}{s : \mathbf{some}(\lambda x.\mathbf{every}(\mathbf{know} x))} \text{ [BA]}
 \end{array}$$

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

4.4.2 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 \setminus and $/$ produces functional application terms, whereas the abstraction over variables corresponds to the introduction of the functional operators.

DEFINITION 4.33. [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 .

$$\begin{array}{c}
 x : A \vdash x : A \\
 \frac{\Gamma \vdash t : A/B \quad \Delta \vdash u : B}{\Gamma \circ \Delta \vdash t(u) : A} \text{ [/E]} \quad \frac{(\Gamma \circ x : B) \vdash t : A}{\Gamma \vdash \lambda x.t : A/B} \text{ [/I]} \\
 \frac{\Delta \vdash u : B \quad \Gamma \vdash t : B \setminus A}{\Delta \circ \Gamma \vdash t(u) : A} \text{ [\setminus E]} \quad \frac{(x : B \circ \Gamma) \vdash t : A}{\Gamma \vdash \lambda x.t : B \setminus A} \text{ [\setminus I]}
 \end{array}$$

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 1.40). A formal definition of the lambda calculus fragment corresponding to LP is given below⁶.

DEFINITION 4.34. [Fragment of the Lambda Terms for LP] Let $\Lambda(\text{LP})$ be the largest $\text{LAMBDA} \subseteq \text{TERM}$ such that

- i.* each subterm of $\alpha \in \text{LAMBDA}$ contains a free variable;
- ii.* no subterm of $\alpha \in \text{LAMBDA}$ contains more than one free occurrence of the same variable;
- iii.* each occurrence of the λ abstractor in $\alpha \in \text{TERM}$ 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 [Ben87, Bus87, Wan92].

THEOREM 4.35. Given an LP derivation of a sequent $A_1, \dots, A_n \vdash B$ one can find a corresponding construction $\alpha_a \in \Lambda(\text{LP})$, and conversely. A term $\alpha_a \in \Lambda(\text{LP})$ is called a construction of a sequent $A_1, \dots, A_n \vdash B$ iff α has exactly the free variable occurrences $x_{\text{type}(A_1)}^1, \dots, x_{\text{type}(A_n)}^n$.

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 4.36. [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 4.37. 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.

⁶Again, 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.

$$\frac{\frac{[B \vdash x : B]^1 \quad \Gamma \vdash t : B \setminus A}{B, \Gamma \vdash t(x) : A} [\setminus E]}{\Gamma \vdash \lambda x.t(x) : B \setminus A} [\setminus I]^1 \quad \text{rewrites to} \quad \frac{D_1}{\Gamma \vdash t : B \setminus A}$$

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 4.38. [Normalization] If \mathcal{D} is a normal form derivation of $x_1 : A_1, \dots, x_n : A_n \vdash \alpha : C$, 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 1.39 and 1.40, respectively.

EXAMPLE 4.39. [Lifting] Starting from the type assignment $\text{Lori} \in np : \text{lori}$, one derives the higher order assignments as following:

$$\frac{\frac{\text{Lori} \vdash np : \text{lori} \quad [np \setminus s \vdash np \setminus s : x]^1}{\text{Lori} \circ np \setminus s \vdash s : x(\text{lori})} [\setminus E]}{\text{Lori} \vdash s / (np \setminus s) : \lambda x.x(\text{lori})} [\setminus I]^1 \quad \frac{[s / np \vdash s / np : x]^1 \quad \text{Lori} \vdash np : \text{lori}}{s / np \circ \text{Lori} \vdash s : x(\text{lori})} [\setminus E]}{\text{Lori} \vdash (s / np) \setminus s : \lambda x.x(\text{lori})} [\setminus I]^1$$

First of all, note how the system assigns a variable to the hypothesis. The latter is discharged by means of $[\setminus I]$ (or $[\setminus I]$) 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: $\text{which} \in (n \setminus n) / (s / np) : \lambda xyz. x(z) \wedge y(z)$. In this way, we obtain the proper recipe for the relative clause *which Sara wrote*, namely $\lambda yz. \text{wrote}(\text{Sara}, z) \wedge y(z)$, as shown below.

EXAMPLE 4.40. [Relative Clause]

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

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

Generalizing over the different uses of unary operators we have reviewed in Section 1.2, one could say that unary operators allow us to express distinctions among members of the same semantic type which are relevant for the syntactic composition. In other words, the unary operators express distinctions similar to the ones expressed by the directional functional implications \setminus and $/$ at the level of meaning assembly, where the directionality information plays no role anymore.

In a similar way, the unary operators encode in the type assignments fine-grained distinctions both within and across languages which are not distinguishable in the meaning assembly. For instance, we have seen them at work to encode the different syntactic behavior of plural and singular Italian articles. They are both interpreted in the domain $\text{Dom}_{((e,t),e)}$, *viz.* the set of functions from nouns to noun phrases, however, their contributions to the linguistic composition differ: the plural article *i* is unable to compose with a singular noun, whereas the singular *il* can. The unary operators encode this difference which is not visible on the level of the domains of interpretation. Similarly, the crosslinguistic contrast between the way adjectives may combine with nouns in Italian and in English does not play any role in the assignment of the meaning to their composition, but it is relevant for their syntactic assembly in the two languages. This observation about unary operators is at the heart of the analyses presented in the next Chapter.

4.5 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. Unary operators and modes can be used to lexically anchor linguistic composition.
4. The correspondence between proofs and natural language models, via the lambda terms, properly accounts for the natural language syntax semantics interface.