3

Lambek Calculus and Montague Grammar

Summary. This chapter discusses one of the important advantages of using (Lambek) categorial grammars: the straightforward correspondence between Lambek calculus proofs and derivations in Montague-style semantics, which extends straightforwardly to modern theories like DRT. In order to keep the exposition simple, we will only briefly discuss the intensional operators of Montague.

3.1 Introduction

The Lambek calculus is a lexicalized formalism: that is, the Lambek calculus consists of a universal set of deduction rules — the logical properties of which we have studied in detail in the previous chapter — and to obtain a Lambek *grammar* we only add a lexicon, a function Lex which assigns a finite set of types to each word.

Now we will turn our attention to its relation to Montague semantics, introduced in (Montague, 1970a,b, 1973) which is a very important feature of categorial grammars.

We do not intend to give a lecture on Montague semantics, which is a large research topic in itself — the reader interested in this topic is referred to (Dowty et al, 1981; Gamut, 1991; Partee and Hendriks, 2011) for general introductions, and to (Carpenter, 1996) for many more applications of the Lambek calculus and Montague grammar to different semantic phenomena — but only to illustrate Montague semantics viewed from the perspective of the Lambek calculus. Montague semantics is also a controversial view of semantics. Indeed it has nothing fancy to say about mental representation or the organization of concepts as for instance in (Jackendoff, 1995) or (Pustejovsky, 1995) (though we believe that Montague grammar is at least compatible with a more comprehensive cognitive theory of meaning): the semantics of a sentence is given by formulae of (higher-order) predicate calculus, possibly of intensional logic, and the elementary expression are interpreted by logical constants: the word “Paul” is interpreted by the logical constant *Paul* and the word “car” by the logical constant *car* (or equivalently, as is more usual in Montague semantics, $\lambda x^e.car(x)$). Nevertheless it enables a neat and

R. Moot and C. Retoré: The Logic of Categorial Grammars 2012, LNCS 6850, pp. 65–99, 2012.

computational treatment of (co)reference and of quantifiers and this is an important step towards modeling meaning computationally.[1] Interpreting sentences in a logical syntax also allows us to model entailment: a set of sentences $s_1,\ldots,s_n$ *entails* another sentence s if and only is the truth of all of the sentences $s_1,\ldots,s_n$ implies the truth of s. If all of the sentences are interpreted as logical formulae, then this semantic notion of entailment corresponds to the logical notion of entailment, that is to $s_1,\ldots,s_n \vdash s$ where '$\vdash$' is the logical entailment relation of the logic we use for our interpretation.[2]

Although Montague himself thought that one should forget about whatever lies between the sentence and its interpretation in possible world semantics, we shall choose a more intermediate level, which could be called the "syntax of semantics", as the endpoint of our semantic interpretation: the logical formula itself. We agree with Montague and others that the intermediate steps, being unobservable, are difficult to study (or even to substantiate). The logical form is already less observable than the sentence, though tools like entailment allow us to reason about properties these logical forms must have.

3.2 Logic and Lambda Calculus

We will first give an extremely brief introduction to the typed lambda calculus, then see how to model logical formulae in typed lambda calculus.

3.2.1 Typed Lambda Calculus and Intuitionistic Propositional Calculus

We do not pretend to include here a presentation of the typed lambda calculus, many of them exist (Krivine, 1990; Seldin and Hindley, 1980; Girard et al, 1988) but we provide a brief reminder of the minimal background necessary to follow a presentation of Montague semantics. There are at least two unrelated ways to consider the relation between the typed lambda calculus and logic:

[Church]. Typed lambda terms as logical formulae disregarding their truth and provability. Provided the type systems includes a type **e** for entities (also called individuals) and one, **t**, for truth values (or propositions), constants for logical connectives and constants for the predicates, functions and constants, every closed formula correspond to a normal lambda term of type **t**, and conversely, every closed normal lambda term (with constants as indicated) of type **t** corresponds to a formula (this will be made precise in next section).

[1] We consider these phenomena part of semantics. However, in generative grammar, they are considered part of syntax. For a good introduction to semantics from the point of view of generative grammar, see (Heim and Kratzer, 1997).

[2] There are, of course, many caveats to using logical entailment to model semantic entailment: semantic entailment depends, in many cases, on complex world knowledge and there has been much debate about the difference between the logical implication "$A \Rightarrow B$" and the construction "if A then B" in natural language.

[Curry-Howard]. Typed lambda terms as proofs in intuitionistic logic. Any proof in the pure implicative propositional logic of C under the assumptions $H_1, \dots, H_n$ with propositional variables in P can be identified with a lambda term (not necessarily normal) of type C with free variables of type $H_1, \dots, H_n$.

When studied from the context of a non proof-theoretic syntax, Montague semantics is only concerned with the first point of view, since it computes logical formulae. Here we shall also consider the second viewpoint, because the starting point for our calculations are the syntactic analyses in type logical grammars (Lambek grammars, multimodal categorial grammars, etc.), which are proofs, and which we convert into lambda terms that are proofs as well, though they are proofs which represent logical formulae.

Historically, the first viewpoint was introduced by Church in order to give a proper account of substitution in Hilbert-style deductive systems. The second one, which appears later, is the viewpoint responsible for typed functional programming languages such as ML, CaML, Haskell, etc. One may wonder about the connection between the two viewpoints, if any. It is actually a tiny one: what is a formula a proof of? A formula is a proof of its own correctness, and there is no relation with a proof of the formula itself! We are to see this tiny connection at work.

Definition 3.1 (Types). *From a (finite) set of basic types P, also called atomic types, we define the set of types as follows.*

$$\mathscr{T} \quad ::= \quad P \quad | \quad (\mathscr{T} \to \mathscr{T})$$

Types which are not basic types are called compound *types.*

For instance, if $a, b \in P$ then $a \to ((a \to b) \to b)$ is a type. Note that some authors write $\langle a, \langle \langle a, b \rangle, b \rangle \rangle$; using $\langle X, Y \rangle$ when we write $X \to Y$.

To represent formulae we require that P contains the type **t** for truth values and the type **e** for individuals (in case one wishes to represent a multisorted logic with sorts $e_1, \dots, e_n$ these sorts should be in P).

Any type $T \in \mathscr{T}$, be it atomic or compound, is given a countable set of variables of this type. Each of these variables, written as $x_i : T$ or x_i^T, is a term of type T with free variable x_i. Types may be provided with a finite or countable set of *constants* of this type. They behave much like free variables, with the exception that they cannot be bound. A term t of type T is written $t : T$ or t^T. We will switch between these two notations for terms and variables freely throughout this chapter.

- Variables: if x is a variable of type T written $x : T$, then x is a term of type T with one free occurrence of the variable x. The only subterm of x is x.
- Constant: if k is a constant of type T written $k : T$, then k is a term of type T without free variables. The only subterm of k is k itself.
- Abstraction: if t is term of type T with the free variables V, and if x is a variable of type U then $w = \lambda x^U.t$ is a term of type $U \to T$ with free variables $F \setminus \{x\}$ the free occurrences of $x : U$ in t are bound in w by the initial λx^U. Occurrences of other free variables remain free. The subterms of w are w and the subterms of t.

- Application: if t is a term of type $U \to V$ with free variables F and if u is a term of type U with free variables G then $w = (t(u))$ it a term of type V with free variables $F \cup G$. Any free occurrence of a free variable in t or in u is free in $(t(u))$, but observe there might be free variables common to t and u. The subterms of w are w itself, the subterms of t and the subterms of u.

An occurrence of a variable x which is not a free occurrence of this variable x is a *bound* variable and therefore it is associated with its λx binder.

We use implicit right bracketing for types: $a{\to}b{\to}c = a{\to}(b{\to}c)$, which goes with implicit left bracketing for λ-terms: $w\ v\ u = (w\ v)\ u$.

Let $w = t[u]$ be a term with the subterm $u = \lambda x\, t'$, and let z be a variable not occurring in w. Then $w' = t[\lambda z\, t'[x := z]]$ is a term which is said to be *alpha equivalent* to w. All operations defined on lambda terms are defined up to alpha equivalence, that is up to the renaming of free variables.[3]

The notation $t[x := u]$, where t is a term, x a variable and u a lambda term of the same type as x, represents the term obtained from t by replacing all free occurrences of x by the lambda term u. Note that we may need to substitute u into a term t' alpha equivalent to t to make sure all free variables of u are free variables of $t[x := u]$. For example, $(\lambda y.f(x,y))[x := y]$ is *not* equal to $\lambda y.f(y,y)$, where the free occurrence y is accidentally bound, but — after alpha conversion to $(\lambda z.f(x,z))[x := y]$ — to $\lambda z.f(y,z)$.

A one-step beta reduction is defined by: $(\lambda x\ t)\ u \overset{\beta}{\rightsquigarrow}_1 t[x := u]$ and it preserves the typing. Such a step of beta reduction may also take place in a subterm of a term. Finally beta reduction can be iterated: beta reduction, written $\overset{\beta}{\rightsquigarrow}$ is the reflexive, transitive closure of the one-step beta reduction.

One should know the following results:

- Beta reduction is confluent, one also says Church-Rosser, that is to say if a term t beta reduces to two terms t' and t'' then there exists a term w such that t' and t'' both beta reduce to w.
- There is no infinite path of beta reduction in the typed lambda calculus (strong normalization).
- As a consequence of the first two points: every typed lambda term has a unique normal form, which is reached no matter how the beta reductions are performed.

One may also consider eta expansion, which is defined as follows, for a term t without free occurrence of x:

$$t^{A\to B} \overset{\eta}{\rightsquigarrow} \lambda x^{A}\ (t(x))$$

It will be useful to consider β-normal η-long forms Huet (1976) which are defined using the auxiliary notion of atomic forms:

- Variables are atomic forms.
- $(M\ N)$ is an atomic form when $M : A \to B$ is an atomic form and $N : A$ is in β-normal η-long form.

[3] Basically, alpha equivalence says that $x \mapsto 2x$ is the same function as $z \mapsto 2z$.

- Atomic terms whose type is a base type are β-normal η-long forms.
- $\lambda x^A\ t$ is a β-normal η-long form whenever t is.

Every term as a unique β-normal η-long form, which is obtained by beta reduction and then some eta expansion steps. Roughly speaking every term of type $A \to B$ without arguments is of the form $\lambda x^A\ t$ with $t : B$ (the beta-normal eta-long form requires replacing an $f : A \to B$ without argument with $(\lambda x^A\ (f\ x)) : A \to B$.

Observe that a λ-term consists of λ-abstractions, $\lambda x_1 \cdots \lambda x_n$, $1 \leq i \leq n$ (possibly none when $i = 0$) and then one subterm f which is successively applied to several arguments, λ-terms $t_1, \ldots t_p$ (possibly none when $p = 0$). If the lambda term is normal, f cannot be a λ-abstraction. If it were, $f = \lambda x_0\ u$, because the term is normal, then there would be no application of f (p=0) but in this case the λy should be considered as λx_{n+1}. Hence f is either a constant or a variable.

3.2.2 First Order Logic, Mono and Multisorted

First order logic has a single sort of individuals denoted here, as in Montague's work by $\mathbf{e}$ (entities) — Church call this type ι. Here we shall consider several sorts $\mathbf{e}_i$ with $1 \leq i \leq N$ the standard first-order case being $N = 1$.

Let us define functional types as $\mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (.... \to \mathbf{e}_{q_{s_q}} \to \mathbf{e}_{q_0}))$ (in case $s_q = 0$ this type is $\mathbf{e}_{q_0}$)) and relational types as $\mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (.... \to \mathbf{e}_{q_{s_q}} \to \mathbf{t}))$ (in case $s_q = 0$, this type is $\mathbf{t}$) .

To define a multisorted language one needs several sets of symbols which are assumed to be at most countable and possibly empty:

- Constants: for each sort $\alpha = \mathbf{e}_{q_0}$ we have a set of constants a_n^α of sort α.
- Variables: for each sort $\alpha = \mathbf{e}_{q_0}$ we have a set of variables x_n^α of sort α.
- Function symbols: for every functional type $\phi = \mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (.... \to \mathbf{e}_{q_{s_q}} \to \mathbf{e}_{q_0}))$ we have a countable set of function symbols f_q^ϕ of this type ϕ.[4]
- Relational symbols: for every relational type $\psi = \mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (.... \to \mathbf{e}_{q_{s_q}} \to \mathbf{t}))$ we have a countable set of relational symbols R_k^ψ of this type ψ. Propositional constants, if any, are viewed as relational symbols with $s_q = 0$ i.e. with $\psi = \mathbf{t}$.

We first need to define *terms*:

- A constant a_n^α is a term of sort α.
- A variable x_n^α is a term of sort α.
- If f_q is function symbol of type $\phi = \mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (.... \to \mathbf{e}_{q_{s_q}} \to \mathbf{e}_{q_0}))$ and if t_i for $i = 1, \ldots, s_q$ are terms of respective sorts $\mathbf{e}_{q_i}$ then $f_q(t_1, \ldots, t_{s_q})$ is a term of sort $\mathbf{e}_{q_0}$.
- Nothing else is a term. Terms without variables are called closed terms; every occurrence of a variable in a term is a free occurrence.

[4] Constants can be viewed as function symbols with $s_q = 0$ i.e. with $\phi = \mathbf{e}_{q_0}$ for some q_0, but as we shall often use languages with constants and without proper symbol we do not use this generalization.

Then atomic formulae are defined as follows: if R is a relational symbol of type $\psi = \mathbf{e}_{q_1} \rightarrow (\mathbf{e}_{q_2} \rightarrow (.... \rightarrow \mathbf{e}_{q_{s_q}} \rightarrow \mathbf{t}))$ and if t_i for $1 \leq i \leq s_q$ terms of respective sorts $\mathbf{e}_{q_i}$, then the following is an *atomic formula*:

$$R(t_1, \ldots, t_{s_q})$$

The free variables in an atomic formula are the free variables in the terms.

Next we can define *formulae*

- An atomic formula is a formula.
- If F is a formula, so is $\neg F$.
- If F and G are two formulae so are $F \wedge G$, $F \vee G$, $F \Rightarrow G$.
- If F is a formula, and if x is a variable of some sort $\mathbf{e}_i$ then $\forall x : \mathbf{e}_i.F$ and $\exists x : \mathbf{e}_i.F$ are formulae.
- Nothing else is a formula.

Models for multisorted logic

The intended models for many-sorted logic with N sorts $(\mathbf{e}_i)_{1 \leq i \leq N}$ are defined by a partition of a set $\bar{\mathbf{e}}$ into N classes $\bar{\mathbf{e}}_i$. In order to define the model, we shall need an interpretation I as well as an assignment A which maps every free variable into an object of $\bar{\mathbf{e}}_i$.

An interpretation is fully defined by

- an element of $\bar{\mathbf{e}}_i$ for every constant of sort $\mathbf{e}_i$.
- a function $\bar{f}_q$ from $\bar{\mathbf{e}}_{q_1} \times \ldots \times \bar{\mathbf{e}}_{q_{s_q}} \mapsto \bar{\mathbf{e}}_{q_0}$ for each function symbol f_q of type $\phi = \mathbf{e}_{q_1} \rightarrow (\mathbf{e}_{q_2} \rightarrow (.... \rightarrow \mathbf{e}_{q_{s_q}} \rightarrow \mathbf{e}_{q_0}))$, that is a function from s_q elements of the model (each of the appropriate sort $\bar{\mathbf{e}}_{q_i}$) to an element $\bar{\mathbf{e}}_{q_0}$.
- a subset $\bar{R}$ of $\bar{\mathbf{e}}_{q_1} \times \ldots \times \bar{\mathbf{e}}_{q_{s_q}}$ for each relational symbol R of type $\psi = \mathbf{e}_{q_1} \rightarrow (\mathbf{e}_{q_2} \rightarrow (.... \rightarrow \mathbf{e}_{q_{s_q}} \rightarrow \mathbf{t}))$, that is a set of tuples with s_q elements of the model (each of sort $\bar{\mathbf{e}}_{q_i}$) for which the relation R is true.

An assignment A is a map from variables to elements in the corresponding set: $A(x) \in \bar{\mathbf{e}}_i$ whenever x is of sort $\mathbf{e}_i$.

An interpretation together with an assignment maps terms of sort $\mathbf{e}_i$ to the set $\bar{\mathbf{e}}_i$ as follows: constants of sort $\mathbf{e}_i$ are mapped to elements in $\bar{\mathbf{e}}_i$ by the interpretation, variables of sort $\mathbf{e}_i$ are mapped to elements in $\bar{\mathbf{e}}_i$ by the assignment, and, after that, if f_q is function symbol of type $\phi = \mathbf{e}_{q_1} \rightarrow (\mathbf{e}_{q_2} \rightarrow (.... \rightarrow \mathbf{e}_{q_{s_q}} \rightarrow \mathbf{e}_{q_0}))$ and if t_i for $i = 1, \ldots, s_q$ are terms of respective sorts $\mathbf{e}_{q_i}$ which the interpretation and the assignment map to $\bar{\mathbf{e}}_{q_i}$ then the interpretation of $f_q(t_1, \ldots, t_{s_q})$ which is in $\bar{\mathbf{e}}_{q_0}$ is simply $\bar{f}_q(\bar{\mathbf{e}}_1, \ldots, \bar{\mathbf{e}}_{s_q})$.

Given an interpretation I and an assignment A, formulae are interpreted as usual taking sorts into account:

- An atomic formula $R(t_1, \ldots, t_{s_q})$ with R a relational symbol of type $\psi = \mathbf{e}_{q_1} \rightarrow (\mathbf{e}_{q_2} \rightarrow (.... \rightarrow \mathbf{e}_{q_{s_q}} \rightarrow \mathbf{t}))$ and t_i for $1 \leq i \leq s_q$ terms of respective sorts $\mathbf{e}_{q_i}$, is true whenever $(\bar{\mathbf{e}}_{q_1} \times \bar{\mathbf{e}}_{q_2} \times \ldots \times \bar{\mathbf{e}}_{q_{s_q}})$ is in $\bar{R}$.

- If F_1 and F_2 are two formulae and $\diamond$ is a connective, the truth of the formula $F_1 \diamond F_2$ and $\neg F_1$ is given by the usual truth tables.
- If F is a formula, and if F is true for the interpretation I and for some assignment A' which coincides with A except, possibly, for the variable x of sort $\mathbf{e}_i$ (i.e. $A'(x)$ may differ from $A(x)$) then $\exists x F$ is true with interpretation I and assignment A.
- If F is a formula, and if F is true for the interpretation I and for all assignments A' which coincides with A except, possibly, for the variable x of sort $\mathbf{e}_i$ (i.e. $A'(x) \neq A(x)$) then $\forall x.F$ is true with interpretation I and assignment A.

We will not have a lot more to say about these models in this book, but we hope that it helps to give a clearer picture about multisorted logic.

3.2.3 Second Order and Higher Order Logic

Multisorted second order logic is a rather simple extension of the logic above: for each type $\psi = \mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (.... \to \mathbf{e}_{q_{s_q}} \to \mathbf{t}))$ in addition to the relational symbols one has relational *variables*. Terms are defined as above and are called first order terms. Atomic formulae are defined as above, except that R can also be a relational variable. Hence the main difference is that one may quantify, existentially and universally, over relational variables: given a formula F and a relational variable X of type $\psi = \mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (.... \to \mathbf{e}_{q_{s_q}} \to \mathbf{t}))$ the expressions $\forall X.F$ and $\exists X.F$ are formulae. The cleanest way to treat quantifiers over relational variables of different types is to treat them as different symbols.

Higher order logic is more complex to define in a standard logical syntax than in the lambda calculus. Usually, one skips function symbols and higher order function symbols, since they are not really necessary: they are particular cases of predicates. However it is non standard to have heterogenous higher order variables, ranging over objects with distinct status and quantification over those: e.g. it would be weird, although not impossible, to quantify over variables of relation between relations and individuals.

Intended models of such logic, which are usually called principal models, are as expected: a relational variable X of type $\psi = \mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (.... \to \mathbf{e}_{q_{s_q}} \to \mathbf{t}))$ is interpreted as any subset of $\bar{\mathbf{e}}_{q_1} \times \bar{\mathbf{e}}_{q_2} \times \ldots \times \bar{\mathbf{e}}_{q_{s_q}}$. The formula $\forall X.F$ (resp. $\exists X.F$) is true whenever it is true for any (resp. some) interpretation of the X as a subset of f $\bar{\mathbf{e}}_{q1} \times \bar{\mathbf{e}}_{q2} \times \ldots \times \bar{\mathbf{e}}_{q_{s_q}}$.

Nevertheless, we need to remember here that there is no completeness result with respect to such principal models, even for the simplest case, second order logic. Completeness can be stated as: every non-contradictory set of formulae admits a model. Compactness says that when any finite subset of a set of formulae admits a model, so does the whole set. Clearly, completeness implies compactness. Indeed, if a set of formulae does not admit a model then it proves “false”. But a proof is finite, hence if there is a proof of “false” from the set of formulae T then a finite number of the formulae of T is enough to prove “false”. Hence if no finite subset of T contains a contradiction, then neither does T, and therefore T has a model. Now consider the following closed formulae F_n $(n \geq 1)$: $\exists x_1 \cdots \exists x_n \wedge_{i=1}^{n} \wedge_{j=1}^{i} x_i \neq x_j$ and

$F_0 : \forall X(\_,\_)$ *Xis an injective application* $\Rightarrow$ *Xis surjective*[5] then clearly every finite subset S of $F_i, i \in N$ has a principal model (assuming N is the largest integer such that $F_N \in S$, take a set $\{a_i \mid i = 1, \dots, N\}$), but there cannot be a set which is finite and with at least n elements for all n.[6]

For second order logic, the reader is referred to van Dalen (1983).

3.2.4 Lambda Terms and Logical Formulae

Assume that the base types are $\mathbf{e}_i$ $1 \le i \le N$ and $\mathbf{t}$ and that the only constants are

- The logical constants:
  - $\neg$ of type $\mathbf{t} \to \mathbf{t}$
  - $\Rightarrow, \wedge, \vee$ of type $\mathbf{t} \to (\mathbf{t} \to \mathbf{t})$
  - for every with i $1 \le i \le n$, two constants $\forall_i$ and $\exists_i$ of type $(\mathbf{e}_i \to \mathbf{t}) \to \mathbf{t}$
- The language constants:
  - R_q of type $\mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (\dots. \to \mathbf{e}_{q_{s_q}} \to \mathbf{t}))$
  - f_q of type $\mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (\dots. \to \mathbf{e}_{q_{s_q}} \to \mathbf{e}_{q_0}))$
- For second order logic, we need second order variables (relational variables) and quantifiers:
  - X_q of type $(\mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (\dots. \to \mathbf{e}_{q_{s_q}} \to \mathbf{t}))) = \tau_q$
  - $\forall^2_{\tau_q} : ((\tau_q \to \mathbf{t}) \to \mathbf{t})$ and $\exists^2_{t_q} : ((\tau_q \to \mathbf{t}) \to \mathbf{t})$

Note that the logical connectives and quantifiers are treated simply as constants of the lambda calculus, so a first-order logic formula

$$\forall x.(f(x) \Rightarrow g(x)) \tag{3.1}$$

will be written as.

$$\forall(\lambda x((\Rightarrow \ f(x))\ g(x))) \tag{3.2}$$

Gamut (1991, Section 4.4.3) calls treating quantifiers the way we do in this chapter (exemplified by lambda-term 3.2 above) the *categorematic* way of introducing quantifiers (as opposed to the syncategorematic shown as formula 3.1 above). Introducing lambda-terms categorematically has the advantage that we do not need to modify our notions of free and bound variables, since only the λ operator binds variables. So instead of binding a variable x in a formula (as a syncategorematic quantifier does) we have a categorematic quantifier which takes a term of type $\mathbf{e}_i \to \mathbf{t}$ as an argument, which, because it is eta-expanded, is of the form $\lambda x^{\mathbf{e}_i}.y^{\mathbf{t}}$ for some y,

[5] This can be formulated as follows:

- X defines an application: $(\forall x.\exists y.X(x,y)) \wedge (\forall x.\forall y.\forall y'.\,(X(x,y) \wedge X(x,y')) \Rightarrow y = y')$
- X is injective $\forall x.\forall x'.\forall y.\,(X(x,y) \wedge X(x',y)) \Rightarrow x = x'$
- X is surjective: $(\forall y.\exists x.X(x,y))$

[6] Completeness and a compactness can be obtained but with non principal models in which n-ary relations are allowed to vary among sub-boolean algebras of powersets.

with the lambda operator taking care of binding the variable x in y. However, in the interest of readability, we will often write $\forall x.y$ instead of $\forall(\lambda x\, y)$ and $x \Rightarrow y$ instead of $((\Rightarrow x)\, y)$.

We can show by induction that every eta-expanded normal λ-term of type $\mathbf{e}_i$ with free variables x_k of type $\mathbf{e}_{i_k}$ corresponds to a term whose free variables are the x_k of sort $\mathbf{e}_{i_k}$.

Because the λ-term is normal and of type $\mathbf{e}_i$ it does not start with a λ, it is either

- a variable x of type $\mathbf{e}_i$ corresponding to the term $x : \mathbf{e}_i$ of multisorted logic (the free variable of the λ-term corresponds to the variable of the term),
- a constant f_q (in this case $s_q = 0$) of type $\mathbf{e}_i$ corresponding to the term $f : \mathbf{e}_i$ of multisorted logic (no free variable in the λ-term and none in the logical term),
- some function f_q applied to s_q arguments t_i of type $\mathbf{e}_{q_i}$ (because of the result type $\mathbf{e}_i$, it cannot be another kind of constant since they cannot yield $\mathbf{e}_i$ when applied to some arguments, and f must be provided with all its arguments to obtain this $\mathbf{e}_i$). The arguments t_i are themselves λ-terms of type $\mathbf{e}_{q_i}$,and they correspond, by induction hypothesis, to logical terms $t_i^\triangledown$ of sort $\mathbf{e}_{q_i}$, with the same variables as the free variables of the λ-term. The logical term, in this case is $f_q(t_1^\triangledown,\dots,t_{s_q}^\triangledown)$ and its free variables are the same as those of the λ-term.

Now let us prove that every λ-term of type $\mathbf{t}$ in beta normal eta long form corresponds to a formula, with the same free variables as the formula.

Because the λ-term is normal and of type $\mathbf{t}$, we note that it does not start with a λ and that it cannot be of the form f_q, because this would imply a result type $\mathbf{e}_i$. We proceed by case analysis on the form of the λ-term.

- the λ-term is some R_q or X_q applied to s_q arguments t_i of type $\mathbf{e}_{q_i}$. The arguments t_i are λ-terms of type $\mathbf{e}_{q_i}$,and they correspond, by induction hypothesis, to logical terms $t_i^\triangledown$ of sort $\mathbf{e}_{q_i}$, with the same variables as the free variables of the λ-term . The logical formula, in this case is $R_q(t_1^\triangledown,\dots,t_{s_q}^\triangledown)$ and its free variables are the same as those of the λ-term .
- a binary logical constant $*$ applied to two λ-terms (otherwise the result would not be of type $\mathbf{t}$) t_1 and t_2 of type $\mathbf{t}$: by induction hypothesis each of these corresponds to a formula whose free variables are those of the λ-term. The corresponding formula is $t_1^\triangledown * t_2^\triangledown$.
- a unary logical constant $\neg$ applied to a λ-term (otherwise the result would not be of type $\mathbf{t}$) t_1 of type $\mathbf{t}$: by induction hypothesis each of these corresponds to a formula whose free variables are those of the λ-term. The corresponding formula is $\neg t_1^\triangledown$.
- $\forall_i$ (resp. $\exists_i$) of type $(\mathbf{e}_i \to \mathbf{t}) \to \mathbf{t}$ applied to a term u of type $\mathbf{e}_i \to \mathbf{t}$. Because the λ-term is β-normal η-long, $u = \lambda x : \mathbf{e}_i\ v$ with $v : \mathbf{t}$ and v having an extra free variable, namely $x : \mathbf{e}_i$. The free variable of $\forall_i(\lambda x : \mathbf{e}_i\ v)$ say $y_1,\dots,y_l$ are the ones of $v : \mathbf{t}$ minus $x : \mathbf{e}_i$. By induction hypothesis, $v : \mathbf{t}$ corresponds to a formula $v^\triangledown$ with free variables $x : \mathbf{e}_i$ and $y_1,\dots,y_l$. The formula corresponding to the complete λ-term is simply $\forall x : \mathbf{e}_i v^\triangledown$ (resp. $\exists x : \mathbf{e}_i v^\triangledown$), its free variable are he ones of the λ-term , namely $y_1,\dots,y_l$.

- If there are second order variables and quantifiers, it can be one of them, applied to a term. The quantifier $\forall^2_{t_q} : ((\tau_q \to \mathbf{t}) \to \mathbf{t})$ (resp. $\exists^2_{\tau_q} : ((\tau_q \to \mathbf{t}) \to \mathbf{t})$ is applied to a term of type $(\tau_q \to \mathbf{t})$. Because term are in $\beta\eta$ normal form, the term maybe assumed to be $\lambda X^{\tau_q}.u$ with $u : \mathbf{t}$. By induction hypothesis, u corresponds to a second order formula $u^\triangledown$, as one more free predicate variable X and the corresponding formula is $\forall X : \tau_q\ u^\triangledown$ (resp. $\exists X : \tau_q\ u^\triangledown$).

3.3 From Categorial Analysis to Montague Semantic Analysis

Let us come back to the relation between Montague semantics and categorial grammars. This relation, which was emphasized in particular by van Benthem (van Benthem, 1991) (see also Hendriks, 1993, for a systematic study of this relation), is a consequence of the following fact: simply typed λ-terms, which represent formulae of predicate calculus and neatly handle substitution, are very close to proofs in the Lambek calculus, which are the syntactic analyses of Lambek grammars. Indeed, via the Curry-Howard isomorphism (see e.g. Girard et al, 1988) simply typed λ-terms are proofs in intuitionistic logic, which extends the Lambek calculus. Indeed, reading $a \backslash b$ and $b\,/\,a$ as $a \to b$ (intuitionistic implication) each rule of the Lambek calculus is a rule of intuitionistic logic.[7] Assume our Lambek grammar uses the primitive types: np, n, S. First let us define a morphism from syntactic types to semantic types: these semantic types are formulae are define from two types e (entities) and t (truth values or propositions) with the intuitionistic implication $\to$ as their only connective:

$$types \quad ::= \quad e \quad | \quad t \quad | \quad (types \to types)$$

Thus a common noun like *chair* or an intransitive verb like *sleep* have the type $e \to t$ (the set of entities which are chairs or who sleep) a transitive verb like *takes* is a two place predicate of type $e \to (e \to t)$ (the pairs of entities such that the first one takes the second one) etc.

Thus we can define a morphism from syntactic types to semantic types.

(Syntactic type)*	= Semantic type	
S^*	$= t$	a sentence is a proposition
np^*	$= e$	a noun phrase is an entity
n^*	$= e \to t$	a noun is a subset of the set of entities
$(a \backslash b)^* = (b\,/\,a)^*$	$= a^* \to b^*$	extends $(\_)^*$ to all syntactic types

[7] We will ignore the product formula $a \bullet b$ in this chapter, though this is just to keep the exposition of the lambda-calculus simple. The Curry-Howard isomorphism for the implication/conjunction fragment of intuitionistic logic is discussed in detail in (Girard et al, 1988) and carries over to the Lambek calculus without problems (see Abramsky, 1993; Moortgat, 1997, for two different solutions).

The lexicon associates to each syntactic type $t_k \in \mathrm{Lex}(m)$ of a word m a λ-term τ_k *whose type is precisely* t_k^*, the semantic counterpart of the syntactic type t_k.

As discussed in Section 3.2.4, we need constants for the usual logical operations like quantification, conjunction etc.:

Constant	Type
$\exists$	$(e \to t) \to t$
$\forall$	$(e \to t) \to t$
$\wedge$	$t \to (t \to t)$
$\vee$	$t \to (t \to t)$
$\Rightarrow$	$t \to (t \to t)$

and proper constants for the denotation of the words in the lexicon:

likes	$\lambda x \lambda y$ (likes x) y	$x : e,\ y : e,$ likes : $e \to (e \to t)$
`<< likes >> is a two-place predicate`		
Pierre	λP (P Pierre)	$P : e \to t,$ Pierre : e
`<< Pierre >> is viewed as the set of properties which hold for the entity Pierre`		

These constants can include intensionality operators ˆ and ˇ (Gamut, 1991) which we will discuss briefly in Section 3.7.

Given

- *a syntactic analysis of* $m_1, \ldots, m_n$ in the Lambek calculus, that is a proof $\mathscr{D}$ of $t_1, \ldots, t_m \vdash S$ and
- *the semantics of each word* $m_1, \ldots, m_n$, that are λ-terms $\tau_i : t_i^*$,

we obtain the semantics of the sentence by the following algorithm:

1. Replace every syntactic type in $\mathscr{D}$ with its semantic counterpart; since intuitionistic logic extends the Lambek calculus the result $\mathscr{D}^*$ of this operation is a proof in intuitionistic logic of $t_1^*, \ldots, t_n^* \vdash t = S^*$.
2. Via the Curry-Howard isomorphism, this proof in intuitionistic logic can be viewed as a simply typed λ-term $\mathscr{D}_\lambda^*$ which contains one free variable x_i of type t_i^* per word m_i.
3. Replace in $\mathscr{D}_\lambda^*$. each variable x_i by the λ-term τ_i — whose type is also type t_i^*, so this is a correct substitution.
4. Reduce the resulting λ-term: this provides the semantics of the sentence (another syntactic analysis of the same sentence can lead to a different semantics).

We use natural deduction, because natural deduction is closer to λ-terms, but computing λ-terms from sequent calculus proofs is possible too, though this essentially corresponds to translating the sequent proof into natural deduction, as we have done in Section 2.4.2. Even though we remarked in Section 2.2.1 that it was unnecessary in the Lambek calculus to mark the hypothesis of the $/_i$ and $\backslash_i$ rules, we will coindex hypotheses and introduction rules throughout this chapter: intuitionistic proofs require us to mark which hypotheses correspond to which introduction rules, and since we translate Lambek proofs to intuitionistic proofs, indicating explicitly in *both* logics which hypotheses are withdrawn will make the translation from Lambek calculus proofs to intuitionistic proofs more transparent. For the same reason, we will use the $\rightarrow_e$ rule both with the principal branch on the left (for the translation of a $/_e$ rule) and with the principal branch on the right (for the translation of a $\backslash_e$ rule): the context of the rule application always allows us to determine the principal branch of the rule unambiguously.

3.4 Some Typical Examples

Example 3.2. Consider the following lexicon:

word	***syntactic type*** u ***semantic type*** u^* ***semantics: λ-term of type*** u^* x^v ***means that the variable or constant*** x ***is of type*** v
some	$(S\,/\,(np\,\backslash\,S))\,/\,n$ $(e\rightarrow t)\rightarrow((e\rightarrow t)\rightarrow t)$ $\lambda P^{e\rightarrow t}\ \lambda Q^{e\rightarrow t}\ (\exists^{(e\rightarrow t)\rightarrow t}\ (\lambda x^e(\wedge^{t\rightarrow(t\rightarrow t)}(P\,x)(Q\,x))))$
statements	n $e\rightarrow t$ $\lambda x^e(\mathtt{statement}^{e\rightarrow t}\ x)$
speak_about	$(np\,\backslash\,S)\,/\,np$ $e\rightarrow(e\rightarrow t)$ $\lambda y^e\ \lambda x^e\ ((\mathtt{speak\_about}^{e\rightarrow(e\rightarrow t)}\ x)\ y)$
themselves	$((np\,\backslash\,S)\,/\,np)\,\backslash\,(np\,\backslash\,S)$ $(e\rightarrow(e\rightarrow t))\rightarrow(e\rightarrow t)$ $\lambda P^{e\rightarrow(e\rightarrow t)}\ \lambda x^e\ ((P\,x)\ x)$

Let us first show that "*Some statements speak about themselves*" belongs to the language generated by this lexicon. So let us prove (in natural deduction) the following:

$$(S\,/\,(np\,\backslash\,S))\,/\,n\ ,\ n\ ,\ (np\,\backslash\,S)\,/\,np\ ,\ ((np\,\backslash\,S)\,/\,np)\,\backslash\,(np\,\backslash\,S) \vdash S$$

using the abbreviations So (some) Sta (statements) SpA (speak about) $Refl$ (themselves) for the syntactic types.

$$\cfrac{\cfrac{So \vdash (S/(np\backslash S))/n \quad Sta \vdash n}{So, Sta \vdash (S/(np\backslash S))}\,/_e \qquad \cfrac{SpA \vdash (np\backslash S)/np \quad Refl \vdash ((np\backslash S)/np)\backslash(np\backslash S)}{SpA, Refl \vdash (np\backslash S)}\,\backslash_e}{So, Sta, SpA, Refl \vdash S}\,/_e$$

Using the homomorphism from syntactic types to semantic types we obtain the following intuitionistic deduction, where So^*, Sta^*, SpA^*, Refl^* are abbreviations for the semantic types respectively associated with the syntactic types: So, Sta, SpA, Refl.

$$\dfrac{\dfrac{\mathit{So}^* \vdash (e{\to}t){\to}(e{\to}t){\to}t \quad \mathit{Sta}^* \vdash e{\to}t}{\mathit{So}^*, \mathit{Sta}^* \vdash (e{\to}t){\to}t}\to_e \qquad \dfrac{\mathit{SpA}^* \vdash e{\to}e{\to}t \quad \mathit{Refl}^* \vdash (e{\to}e{\to}t){\to}e{\to}t}{\mathit{SpA}^*, \mathit{Refl}^* \vdash e{\to}t}\to_e}{\mathit{So}^*, \mathit{Sta}^*, \mathit{SpA}^*, \mathit{Refl}^* \vdash t}\to_e$$

The λ-term representing this deduction simply is

$$((\mathit{some\ statements})\ (\mathit{themselves\ speak\_about}))\ \text{of type } t$$

where *some*, *statements*, *themselves*, *speak_about* are variables with respective types So^*, Sta^*, Refl^*, SpA^*. Let us replace these variables with the semantic λ-terms (of the same type) which are given in the lexicon. We obtain the following λ-term of type t (written on two lines) that we reduce:

$$\Big(\big(\lambda P^{e\to t}\ \lambda Q^{e\to t}\ (\exists^{(e\to t)\to t}\ (\lambda x^e (\wedge (P\ x)(Q\ x))))\big)\big(\lambda x^e (\mathtt{statement}^{e\to t}\ x)\big)\Big)$$
$$\Big(\big(\lambda P^{e\to(e\to t)}\ \lambda x^e\ ((P\ x)x)\big)\big(\lambda y^e\ \lambda x^e\ ((\mathtt{speak\_about}^{e\to(e\to t)}\ x)y)\big)\Big)$$

$$\downarrow \beta$$

$$\big(\lambda Q^{e\to t}\ (\exists^{(e\to t)\to t}\ (\lambda x^e (\wedge^{t\to(t\to t)} (\mathtt{statement}^{e\to t}\ x)(Q\ x))))\big)$$
$$\big(\lambda x^e\ ((\mathtt{speak\_about}^{e\to(e\to t)}\ x)x)\big)$$

$$\downarrow \beta$$

$$\big(\exists^{(e\to t)\to t}\ (\lambda x^e (\wedge (\mathtt{statement}^{e\to t}\ x)((\mathtt{speak\_about}^{e\to(e\to t)}\ x)x)))\big)$$

This term represent the following formula of predicate calculus (in a more pleasant format):

$$\exists x : e\ (\mathtt{statement}(x) \wedge \mathtt{speak\_about}(x,x))$$

This is the semantics of the analyzed sentence.

Example 3.3 (Sorts and Adverbs). The following example illustrates the use of a multisorted logic with sorts h (human beings) v (events) — and possibly others — and the possible treatment of higher order predicate by reification.

(3.3) Michele works hard.

When properties likes “works” can be the argument of a higher order predicate like the adverb “hard” it is common to introduce an event variable to designate ”work” to which we apply the adverb “hard”, thereby avoiding the need for to analyze “hard” as a property of a property. In this case, the event variable should be bound, hence, one should translate the syntactic type S by $v \rightarrow \mathbf{t}$, rather than by $\mathbf{t}$, and at the end of the computation an existential quantifier over events should be applied to event variables.

word	***syntactic type*** u ***semantic type*** u^* ***semantics: λ-term of type*** u^* x^v ***means that the variable or constant*** x ***is of type*** v
Michele	np h $\mathtt{M}^h$
works	$np \backslash S$ $h \rightarrow (v \rightarrow t)$ $\lambda u^h\ (\mathtt{works}^{h\rightarrow(v\rightarrow t)}\ u)$
`hard`	$(np \backslash S) \backslash (np \backslash S)$ $(h\rightarrow(v\rightarrow t))\rightarrow(h\rightarrow(v\rightarrow t))$ $\lambda P^{h\rightarrow(v\rightarrow t)}\ \lambda x^h\ \lambda e^v(\wedge((P\ e)x)(\mathtt{hard}^{v\rightarrow t} e))$

Here is the syntactic analysis of the sentence:

$$\dfrac{np \qquad \dfrac{(np \backslash S) \quad (np \backslash S) \backslash (np \backslash S)}{(np \backslash S)} \backslash_e}{S} \backslash_e$$

This Lambek proof can be turned into an intuitionistic proof:

$$\dfrac{h \qquad \dfrac{(h \rightarrow (v \rightarrow S)) \quad (h \rightarrow (v \rightarrow S)) \rightarrow (h \rightarrow (v \rightarrow S))}{(h \rightarrow (v \rightarrow S))} \rightarrow_e}{(v \rightarrow S)} \rightarrow_e$$

The lambda term issued from the syntactic analysis is $((H\ W)\ M)$, and after inserting the lambda terms provided by the lexicon it becomes

$$\Big(\Big(\lambda P^{h\rightarrow(v\rightarrow t)}\ \lambda x^h\ \lambda e^v(\wedge((P\ e)\ x)(\mathtt{hard}^{v\rightarrow t} e))\Big)\Big(\lambda u^h \lambda i^v\ ((\mathtt{works}^{h\rightarrow(v\rightarrow t)}\ u)\ i)\Big)\Big)\ \mathtt{M}^h$$

$$\Big(\lambda x^h\ \lambda e^v((\wedge(((\Big(\lambda u^h \lambda i^v\ ((\mathtt{works}^{h\rightarrow(v\rightarrow t)}\ u)\ i)\Big)\ x))\ e)(\mathtt{hard}^{v\rightarrow t} e)))\Big)\ \mathtt{M}^h$$

$$\Big(\lambda x^h\ \lambda e^v((\wedge(((\mathtt{works}^{h\rightarrow(v\rightarrow t)} x))\ e))(\mathtt{hard}^{v\rightarrow t} e))\Big)\ \mathtt{M}^h$$

$$\lambda e^v((\wedge(((\mathtt{works}^{h\rightarrow(v\rightarrow t)} \mathtt{M}^h))\ e))(\mathtt{hard}^{v\rightarrow t} e))$$

In this reified vision, in order to obtain a logical formula, it is common to consider, once the sentence or discourse has been analyzed, the existential closure of the result. Hence the semantic representation becomes

$$\exists^{(v\to\mathbf{t})\to\mathbf{t}}\left(\lambda e^{v}((\wedge(((\mathtt{works}^{h\to(v\to t)}\mathtt{M}^{h}))\ e))(\mathtt{hard}^{v\to t} e))\right)$$

which, in standard notation is

$$\exists e : \mathit{event}\ \mathtt{works}(e,\mathtt{M}^{h}) \wedge \mathtt{hard}(e)$$

There is a slightly different analysis — without reification but using a higher order predicate — of the same sentence. The lexicon is as follows:

word	***syntactic type*** u ***semantic type*** u^* ***semantics: λ-term of type*** u^* x^v ***means that the variable or constant*** x ***is of type*** v
Michele	np h $\mathtt{M}^h$
works	$np \setminus S$ $h \to t$ $\lambda u^h\ (\mathtt{works}^{h\to\mathbf{t}}\ u)$
hard	$(np \setminus S) \setminus (np \setminus S)$ $(h\to\mathbf{t})\to(h\to\mathbf{t}))$ $\lambda P^{h\to\mathbf{t}}\ (\mathtt{hard}^{(h\to\mathbf{t})\to(h\to\mathbf{t})} P)$

The syntactic analysis is just the same, yielding a term $((H\ W)\ M)$. If one inserts the new semantic lambda terms, one obtains:

$$\left(\lambda P^{h\to\mathbf{t}}\ (\mathtt{hard}^{(h\to\mathbf{t})\to(h\to\mathbf{t})} P)\right)(\lambda u^{h}\ (\mathtt{works}^{h\to\mathbf{t}}\ u))\ \mathtt{M}^{h}$$

$$(\mathtt{hard}^{(h\to\mathbf{t})\to(h\to\mathbf{t})}(\lambda u.(\mathtt{works}^{h\to\mathbf{t}}\ u))\ \mathtt{M}^{h}$$

which can be written in higher order logic as:

$$(\mathtt{hard}(\mathtt{works}))(\mathtt{M})$$

Example 3.4 (Relative pronouns and relative clauses)
The semantic effect of the pronoun is to introduce a conjunction between the relative clause of type $\mathbf{e} \to \mathbf{t}$ and the common noun it is attached to $\mathbf{e} \to \mathbf{t}$. We will use French because it makes a neater distinction between the relative pronoun acting as a subject (*qui*) and relative pronouns acting as an object (*que*).

(3.4)	*Un*	*enfant*	*qui*	*courait*	*est_tombé.*
	A	child	who	ran	fell.

'A child who ran fell.'

Such an example should be structured as follows:

$\exists\, x\, ((child(x) \wedge past\_run(x)) \wedge (past\_fall(x))$

The existential quantifier is correctly applied to the conjunction of two predicates: the restriction *((child(x)* $\wedge$ *past_run(x))* and the predicate *fall(x)*.

The lexicon should look as follows.

word	***syntactic type*** u ***semantic type*** u^* ***semantics: λ-term of type*** u^* x^v ***means that the variable or constant*** x ***is of type*** v
courait	$np\,\backslash\, S$ $\mathbf{e} \rightarrow \mathbf{t}$ $\lambda x^{\mathbf{e}}(past\_run(x))$
est_tombé	$np\,\backslash\, S$ $\mathbf{e} \rightarrow \mathbf{t}$ $\lambda x^{\mathbf{e}}(past\_fall(x))$
enfant	n $\mathbf{e} \rightarrow \mathbf{t}$ $\lambda x^{\mathbf{e}}\ (child(x))$
qui	$(n\,\backslash\, n)\,/\,(np\,\backslash\, S)$ $(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \rightarrow (\mathbf{e} \rightarrow \mathbf{t})$ $\lambda P^{\mathbf{e}\rightarrow\mathbf{t}} \lambda Q^{\mathbf{e}\rightarrow\mathbf{t}} (\lambda x^{\mathbf{e}} \wedge (Px)(Qx))$
un	$(S\,/\,(np\,\backslash\, S))\,/\,n$ $(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$ $\lambda P^{\mathbf{e}\rightarrow\mathbf{t}} \lambda Q^{\mathbf{e}\rightarrow\mathbf{t}} \exists (\lambda x^{\mathbf{e}} \wedge (Px)(Qx))$

The syntactic analysis is

$$\dfrac{\dfrac{(S\,/\,(np\,\backslash\, S))\,/\,n \qquad \dfrac{n \qquad \dfrac{(n\,\backslash\, n)\,/\,(np\,\backslash\, S) \qquad np\,\backslash\, S}{n\,\backslash\, n}\,/_e}{n}\,\backslash_e}{(S\,/\,(np\,\backslash\, S))}\,/_e \qquad np\,\backslash\, S}{S}\,\backslash_e$$

The corresponding proof with semantic types instead is:

$$\dfrac{\dfrac{(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow ((\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}) \qquad \dfrac{(\mathbf{e} \rightarrow \mathbf{t}) \qquad \dfrac{(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \qquad \mathbf{e} \rightarrow \mathbf{t}}{(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow (\mathbf{e} \rightarrow \mathbf{t})}\rightarrow_e}{(\mathbf{e} \rightarrow \mathbf{t})}\rightarrow_e}{((\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t})}\rightarrow_e \qquad \mathbf{e} \rightarrow \mathbf{t}}{\mathbf{t}}\rightarrow_e$$

The corresponding lambda term is *un ((qui courait) enfant) est_tombé* when one inserts the lexical lambda terms it yields:

$$((\lambda P^{\mathbf{e}\to\mathbf{t}}\lambda Q^{\mathbf{e}\to\mathbf{t}}\exists(\lambda x{:}\mathbf{e}\wedge(Px)(Qx))$$
$$((\lambda P^{\mathbf{e}\to\mathbf{t}}\lambda Q^{\mathbf{e}\to\mathbf{t}}(\lambda x^{\mathbf{e}}\wedge(P\,x)\;(Q\,x))\lambda x^{\mathbf{e}}\;(past\_run(x)))\lambda x^{\mathbf{e}}\;(child(x))))$$
$$\lambda x^{\mathbf{e}}\;(past\_fall(x)))$$

This reduces to:

$$(\lambda P^{\mathbf{e}\to\mathbf{t}}\lambda Q^{\mathbf{e}\to\mathbf{t}}\exists(\lambda x^{\mathbf{e}}\wedge(P\,x)\;(Q\,x))(\lambda x^{\mathbf{e}}(\wedge(past\_run(x))child(x)))(\lambda x^{\mathbf{e}}\;(past\_fall(x))))$$

an finally to:

$$\exists(\lambda x^{\mathbf{e}}(\wedge(\wedge(past\_run(x)\;child(x))\;(past\_fall(x))))$$

In other words:

$$\exists x(past\_run(x)\wedge child(x)\wedge past\_fall(x))$$

Example 3.5 (Quantifier scope). A classical example is scope ambiguity in a sentence like

(3.5) Every child ate a pizza.

word	***syntactic type*** u
	semantic type u^*
	semantics: λ-term of type u^*
	x^v ***means that the variable or constant*** x ***is of type*** v
every	$(S\,/\,(np\,\backslash\,S))\,/\,n$ (subject)
	$((S\,/\,np)\,\backslash\,S)\,/\,n$ (object)
	$(e\to t)\to((e\to t)\to t)$
	$\lambda P^{e\to t}\;\lambda Q^{e\to t}\;(\forall^{(e\to t)\to t}\;(\lambda x^e(\Rightarrow^{t\to(t\to t)}\;(P\,x)(Q\,x))))$
a	$((S\,/\,np)\,\backslash\,S)\,/\,n$ (object)
	$(S\,/\,(np\,\backslash\,S))\,/\,n$ (subject)
	$(e\to t)\to((e\to t)\to t)$
	$\lambda P^{e\to t}\;\lambda Q^{e\to t}\;(\exists^{(e\to t)\to t}\;(\lambda x^e(\wedge^{t\to(t\to t)}(P\,x)(Q\,x))))$
child	n
	$e\to t$
	$\lambda x^e(\mathtt{child}^{e\to t}\;x)$
pizza	n
	$e\to t$
	$\lambda x^e(\mathtt{pizza}^{e\to t}\;x)$
ate	$(np\,\backslash\,S)\,/\,np$
	$e\to(e\to t)$
	$\lambda y^e\;\lambda x^e\;((\mathtt{ate}^{e\to(e\to t)}\;x)y)$

There are two possible syntactic analyses that will lead to the two different readings. Let us completely compute the first analysis and the first semantic representation.

One of the two possible syntactic analyses is:

$\exists\forall$

$$\dfrac{\dfrac{\dfrac{\dfrac{(S\,/\,(np\,\backslash\,S))\,/\,n \quad n}{(S\,/\,(np\,\backslash\,S))}\,/_e \qquad \dfrac{(np\,\backslash\,S)\,/\,np \quad [np]^1}{(np\,\backslash\,S)}\,/_e}{S}\,/_e}{S\,/\,np}\,/_i(1) \qquad \dfrac{((S\,/\,np)\,\backslash\,S)\,/\,n \quad n}{(S\,/\,np)\,\backslash\,S}\,\backslash_e}{S}\,\backslash_e$$

The corresponding semantic analysis is the following:

$\exists\forall$

$$\dfrac{\dfrac{\dfrac{\dfrac{\overset{every}{(\mathbf{e}\to\mathbf{t})\to(\mathbf{e}\to\mathbf{t})\to\mathbf{t}} \quad \overset{child}{(\mathbf{e}\to\mathbf{t})}}{(\mathbf{e}\to\mathbf{t})\to\mathbf{t}}\to_e \qquad \dfrac{\overset{ate}{\mathbf{e}\to\mathbf{e}\to\mathbf{t}} \quad \overset{o}{[\mathbf{e}]^1}}{\mathbf{e}\to\mathbf{t}}\to_e}{\mathbf{t}}\to_e}{\mathbf{e}\to\mathbf{t}}\to_i(1) \qquad \dfrac{\overset{a}{(\mathbf{e}\to\mathbf{t})\to(\mathbf{e}\to\mathbf{t})\to\mathbf{t}} \quad \overset{pizza}{(\mathbf{e}\to\mathbf{t})}}{(\mathbf{e}\to\mathbf{t})\to\mathbf{t}}\to_e}{\mathbf{t}}\to_e$$

The corresponding lambda terms is:

$\exists\forall = (a\ pizza)(\lambda o^{\mathbf{e}}(every\ child)(ate\ o))$

where words have to be replaced with the corresponding lambda terms.

Let us first compute the following lambda terms

$(a\ pizza)$
$= (\lambda P^{e\to t}\ \lambda Q^{e\to t}\ (\exists^{(e\to t)\to t}\ (\lambda x^e(\wedge^{t\to(t\to t)}(P\,x)(Q\,x)))))(\lambda z^e(\mathtt{pizza}^{e\to t}\ z))$
$= (\lambda Q^{e\to t}\ (\exists^{(e\to t)\to t}\ (\lambda x^e(\wedge^{t\to(t\to t)}((\lambda z^e(\mathtt{pizza}^{e\to t}\ z))\ x)(Q\,x)))))$
$= (\lambda Q^{e\to t}\ (\exists^{(e\to t)\to t}\ (\lambda x^e(\wedge^{t\to(t\to t)}((\mathtt{pizza}^{e\to t}\ x)))(Q\,x))))$

$(every\ child)$
$= (\lambda P^{e\to t}\ \lambda Q^{e\to t}\ (\forall^{(e\to t)\to t}\ (\lambda x^e(\Rightarrow^{t\to(t\to t)}\ (P\,x)(Q\,x)))))(\lambda u^e(\mathtt{child}^{e\to t}\ u))$
$= (\lambda Q^{e\to t}\ (\forall^{(e\to t)\to t}\ (\lambda x^e(\Rightarrow^{t\to(t\to t)}\ ((\lambda u^e(\mathtt{child}^{e\to t}\ u))\ x)(Q\,x)))))$
$= (\lambda Q^{e\to t}\ (\forall^{(e\to t)\to t}\ (\lambda x^e(\Rightarrow^{t\to(t\to t)}\ (\mathtt{child}^{e\to t}\ x)(Q\,x)))))$

$(every\ child)(ate\ o) =$
$(\lambda Q^{e\to t}\ (\forall^{(e\to t)\to t}\ (\lambda w^e(\Rightarrow^{t\to(t\to t)}(\mathtt{child}^{e\to t}\ w)(Q\,w)))))((\lambda y^e\ \lambda x^e\ ((\mathtt{ate}^{e\to(e\to t)}\ x)\ y))\ o)$
$= (\lambda Q^{e\to t}\ (\forall^{(e\to t)\to t}\ (\lambda w^e(\Rightarrow^{t\to(t\to t)}\ (\mathtt{child}^{e\to t}\ w)(Q\,w)))))(\lambda x^e\ ((\mathtt{ate}^{e\to(e\to t)}\ x)\ o))$
$= \forall^{(e\to t)\to t}\ (\lambda w^e(\Rightarrow^{t\to(t\to t)}\ (\mathtt{child}^{e\to t}\ w)((\lambda x^e\ ((\mathtt{ate}^{e\to(e\to t)}\ x)\ o))\ w)))$
$= \forall^{(e\to t)\to t}\ (\lambda w^e(\Rightarrow^{t\to(t\to t)}\ (\mathtt{child}^{e\to t}\ w)(((\mathtt{ate}^{e\to(e\to t)}\ w)\ o))))$

$(a\ pizza)(\lambda o\ (every\ child)(ate\ o))$
$= (\lambda Q^{e\to t}\ (\exists^{(e\to t)\to t}\ (\lambda x^e(\wedge^{t\to(t\to t)}((\mathtt{pizza}^{e\to t}\ x)))(Q\,x))))$
$(\lambda o \forall^{(e\to t)\to t}\ (\lambda w^e(\Rightarrow^{t\to(t\to t)}\ (\mathtt{child}^{e\to t}\ w)(((\mathtt{ate}^{e\to(e\to t)}\ w)\ o)))))$
$= (\exists^{(e\to t)\to t}\ (\lambda x^e(\wedge^{t\to(t\to t)}((\mathtt{pizza}^{e\to t}\ x)))$
$((\lambda o \forall^{(e\to t)\to t}\ (\lambda w^e(\Rightarrow^{t\to(t\to t)}\ (\mathtt{child}^{e\to t}\ w)(((\mathtt{ate}^{e\to(e\to t)}\ w)\ o)))))\ x)))$

$= (\exists^{(e\to t)\to t}\ (\lambda x^e(\wedge^{t\to(t\to t)}((\mathtt{pizza}^{e\to t}\ x)))$
$\quad (\forall^{(e\to t)\to t}\ (\lambda w^e(\Rightarrow^{t\to(t\to t)}\ (\mathtt{child}^{e\to t}\ w)((\mathtt{ate}^{e\to(e\to t)}\ w)\ x))))))$
that one usually writes:

$$\exists x.\ pizza(x) \wedge \forall w.\ (child(w) \Rightarrow ate(w,x))$$

The other possible syntactic analysis is:
$\forall\exists$

$$\dfrac{\dfrac{(S\,/\,(np\,\backslash\,S))\,/\,n \quad n}{(S\,/\,(np\,\backslash\,S))}\,/_e \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[np]^1 \quad \dfrac{(np\,\backslash\,S)\,/\,np \quad [np]^2}{(np\,\backslash\,S)}\,/_e}{S}\,\backslash_e}{S\,/\,np}\,/_i(2) \quad \dfrac{((S\,/\,np)\,\backslash\,S)\,/\,n \quad n}{(S\,/\,np)\,\backslash\,S}\,\backslash_e}{S}\,\backslash_e}{np\,\backslash\,S}\,\backslash_i(1)}{S}\,/_e$$

It corresponds to the following analysis:
$\forall\exists$

$$\dfrac{\dfrac{\overset{\mathit{every}}{(\mathbf{e}\to\mathbf{t})\to(\mathbf{e}\to\mathbf{t})\to\mathbf{t}} \quad \overset{\mathit{child}}{(\mathbf{e}\to\mathbf{t})}}{(\mathbf{e}\to\mathbf{t})\to\mathbf{t}}\to_e \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{s}{[\mathbf{e}]^1} \quad \dfrac{\overset{\mathit{ate}}{\mathbf{e}\to\mathbf{e}\to\mathbf{t}} \quad \overset{o}{[\mathbf{e}]^2}}{\mathbf{e}\to\mathbf{t}}\to_e}{\mathbf{t}}\to_e}{\mathbf{e}\to\mathbf{t}}\to_i(2) \quad \dfrac{\overset{a}{(\mathbf{e}\to\mathbf{t})\to(\mathbf{e}\to\mathbf{t})\to\mathbf{t}} \quad \overset{\mathit{pizza}}{(\mathbf{e}\to\mathbf{t})}}{(\mathbf{e}\to\mathbf{t})\to\mathbf{t}}\to_e}{\mathbf{t}}\to_e}{\mathbf{e}\to\mathbf{t}}\to_i(1)}{\mathbf{t}}\to_e$$

This proof corresponds to the lambda term

$$\forall\exists = (every\ child)(\lambda s.\ (a\ pizza)(\lambda o\ ((ate\ o)\ s)))$$

where the words are to be replaced with the lambda terms from the lexicon and that we are going to reduce.

Let us compute the following lambda terms, using the terms we computed before for $(a\ pizza)$ and $(every\ child)$.

$\quad (a\ pizza)(\lambda o\ ((ate\ o)\ s))$
$= (\lambda Q^{e\to t}\ (\exists^{(e\to t)\to t}\ (\lambda x^e(\wedge^{t\to(t\to t)}((\mathtt{pizza}^{e\to t}\ x)))(Q\,x))))$
$(\lambda o\ (((\lambda y^e\ \lambda x^e\ ((\mathtt{ate}^{e\to(e\to t)}\ x)\ y))\ o)\ s)))$
$= (\lambda Q^{e\to t}\ (\exists^{(e\to t)\to t}\ (\lambda x^e(\wedge^{t\to(t\to t)}((\mathtt{pizza}^{e\to t}\ x)))(Q\,x))))(\lambda o\ ((\mathtt{ate}^{e\to(e\to t)}\ s)\ o))$
$= (\exists^{(e\to t)\to t}\ (\lambda x^e(\wedge^{t\to(t\to t)}((\mathtt{pizza}^{e\to t}\ x)))((\lambda o\ ((\mathtt{ate}^{e\to(e\to t)}\ s)\ o))\ x)))$
$= (\exists^{(e\to t)\to t}\ (\lambda x^e(\wedge^{t\to(t\to t)}((\mathtt{pizza}^{e\to t}\ x)))((\mathtt{ate}^{e\to(e\to t)}\ s)\ x)))$

$$\forall\exists = (every\ child)(\lambda s.\ (a\ pizza)(\lambda o\ ((ate\ o)\ s)))$$
$$= (\lambda Q^{e\to t}\ (\forall^{(e\to t)\to t}\ (\lambda u^{e}(\Rightarrow^{t\to(t\to t)}\ (\mathtt{child}^{e\to t}\ u)(Q\ u)))))$$
$$(\lambda s.\ (\exists^{(e\to t)\to t}\ (\lambda x^{e}(\wedge^{t\to(t\to t)}((\mathtt{pizza}^{e\to t}\ x)))((\mathtt{ate}^{e\to(e\to t)}\ s)\ x))))$$
$$= (\forall^{(e\to t)\to t}\ (\lambda u^{e}(\Rightarrow^{t\to(t\to t)}\ (\mathtt{child}^{e\to t}\ u)$$
$$((\lambda s.\ (\exists^{(e\to t)\to t}\ (\lambda x^{e}(\wedge^{t\to(t\to t)}((\mathtt{pizza}^{e\to t}\ x)))((\mathtt{ate}^{e\to(e\to t)}\ s)\ x))))\ u)))))$$

$$= (\forall^{(e\to t)\to t}\ (\lambda u^{e}(\Rightarrow^{t\to(t\to t)}\ (\mathtt{child}^{e\to t}\ u)$$
$$(\exists^{(e\to t)\to t}\ (\lambda x^{e}.\ (\wedge^{t\to(t\to t)}((\mathtt{pizza}^{e\to t}\ x)))((\mathtt{ate}^{e\to(e\to t)}\ u)\ x))))))$$

which is usually written as:

$$\forall u.\ child(u) \Rightarrow \exists .x\ pizza(x) \wedge ate(u,x)$$

3.5 Determiners, Quantifiers and Type Raising

So far we did not interpret the definite article. To be simple, one can use Hilbert's $\tau : (\mathbf{e} \to \mathbf{t}) \to \mathbf{e}$ which, given a predicate $P^{\mathbf{e}\to\mathbf{t}}$ picks up an element satisfying P. As a definite article needs to be first introduced it would be better to replace logic in the lambda calculus by Discourse Representation Theory (DRT) in the lambda calculus. DRT correctly handles many puzzles concerning discourse referents. We discuss this compositional version of DRT, as well as several motivating examples, in Section 3.6.

Hitherto we stuck to a strict correspondence (often called homomorphism) between syntactic categories and semantic types. As we have seen, this complicates the syntactic categories for quantifiers and determiners somewhat, since they depend on the syntactic position. Why should the syntactic category of *someone* be different in *Someone forgot his wallet.* and in *I met someone*? Here, we will discuss alternative semantic solutions, which also have their own drawbacks.

Systematic Type Raising

The first alternative is to translate *np* into $(\mathbf{e} \to \mathbf{t}) \to \mathbf{t}$ rather than into $\mathbf{e}$. Thus the semantics of "*Peter*" is $\lambda P^{\mathbf{e}\to\mathbf{t}}.\ P(peter)$ The advantage of this solution is that that quantifiers have the same syntactic category no matter where they appear. Determiners will therefore get a lifted type: since "*the cat*", of syntactic type *np* should be, of semantic type $(\mathbf{e} \to \mathbf{t}) \to \mathbf{t}$, the determiner "*the*", of syntactic type $np\,/\,n$, should be of type $(\mathbf{e} \to \mathbf{t}) \to ((\mathbf{e} \to \mathbf{t}) \to \mathbf{t})$ and the term should be $\lambda P^{\mathbf{e}\to\mathbf{t}}\lambda Q^{\mathbf{e}\to\mathbf{t}}Q(\tau(P))$. As a consequence, the semantic types of verbs become more complicated: "*sleeps*", of syntactic type $np \setminus S$, should be of semantic type $((\mathbf{e} \to \mathbf{t}) \to \mathbf{t}) \to \mathbf{t}$ with term $\lambda R^{(\mathbf{e}\to\mathbf{t})\to\mathbf{t}}R(\lambda x^{\mathbf{e}} sleeps(x))$, so we simplify the syntactic types but complicate the semantic types and, consequently, the lambda terms.

A problem of this approach is that we no longer generate two distinct readings for "every child ate a pizza": in Example 3.5 we obtained two proofs for this sentence, each of which corresponded to a different semantic reading. The use of systematic type raising would require us to find another way to generate the two readings;

one possibility would be to assign to different lambda-terms to the verb, but the treatment of quantifier scope as verbal polysemy seems hardly appropriate.

Substituting terms for terms rather than terms for variables

The second alternative, which also provides the same syntactic category for quantifiers wherever they appear in the sentence, is to change the way we substitute the lexical lambda terms: as we have used it before, a Lambek calculus derivation corresponds to a lambda-term with a free variable w for each of the hypotheses and we replace these free variables by lexical lambda terms of the same type. We can make this replacement operation a bit more complicated by allowing the replacement of any lambda term of the correct type provided that its only free variable is the lexical variable w. Let us take a simple example:

word	***syntactic type*** u ***semantic type*** u^* ***semantics:*** $\boldsymbol{\lambda}$***-term of type*** u^* x^v ***means that the variable or constant*** x ***is of type*** v
everyone	np $((e \to t) \to t)$ $\lambda Q^{e \to t}\ (\forall^{(e \to t) \to t}\ (\lambda x^e (Q\ x)))$
sleeps	$(np \backslash S)$ $(e \to t)$ $\lambda x^e\ (\mathtt{sleep}^{e \to t}\ x)$

(3.6) Everyone sleeps

Note the type clash between the syntactic type np, corresponding to the semantic type e, and the semantic type $((e \to t) \to t)$ which is assigned to it in the lexicon. This sentence can be analyzed by a non normal proof in the Lambek calculus, with a subproof of conclusion $(S\,/\,(np \backslash S))$ and a single free variable (or hypothesis) np (observe that it is not an analysis in AB grammars because of the introduction rule).

$$\dfrac{\dfrac{\dfrac{np \qquad [np \backslash S]_1}{S}\,\backslash_e}{S\,/\,(np \backslash S)}\,/_i \qquad np \backslash S}{S}\,/_e$$

The semantic translation of this proof is:

$$\dfrac{\dfrac{\dfrac{\mathbf{e} \qquad [\mathbf{e} \to \mathbf{t}]_1}{\mathbf{t}}\,\to_e}{(\mathbf{e} \to \mathbf{t}) \to \mathbf{t}}\,\to_i \qquad \mathbf{e} \to \mathbf{t}}{\mathbf{t}}\,\to_e$$

The corresponding lambda term is $(\lambda P^{\mathbf{e}\to\mathbf{t}}\, P(\texttt{everyone}))^{(\mathbf{e}\to\mathbf{t})\to\mathbf{t}}\texttt{sleep}$ and the first part only has everyone as a free variable. With the relaxed rule, one can replace the whole subterm by the semantic lambda term associated with *everyone*,

$$\lambda Q^{e\to t}\ (\forall^{(e\to t)\to t}\ (\lambda x^{e}(Q\,x)))$$

thus obtaining as the semantics of the sentence the expression:

$$(\lambda Q^{e\to t}\ (\forall^{(e\to t)\to t}\ (\lambda x^{e}(Q\,x))))(\lambda x^{e}\ (\texttt{sleep}^{e\to t}\ x))$$

which reduces to.

$$(\forall^{(e\to t)\to t}\ (\lambda x^{e}(\texttt{sleep}\,x))))$$

Because this analysis forces us to consider non normal proofs with a subproof of the semantic type, it is not very satisfactory.

To have a single syntactic category for quantifiers wherever they are located, and to handle discourses referents properly, one may go for categorial minimalist grammars, with lambda-mu DRT for computing the semantics (Amblard et al, 2010), but this is goes beyond the scope (!) of the present book.

We will give another solution for unifying the two type assignments in Section 5.1.1.

3.6 Lambek Calculus and Discourse Representation Theory

Though the link between the Lambek calculus and Montague grammar is traditional and well-known, a lot of modern research has been done in what is called *dynamic* semantics. Dynamic semantics takes into account the *context* of an utterance and models how this utterance changes the context. One of the typical applications of this point of view is the semantics of anaphors: anaphors, like "he" or "she" must — in a coherent discourse — refer to a previously introduced individual, as indicated by the following example discourse.

(3.7) The president of the board/A friend from Amsterdam/Thomas arrived yesterday.
(3.8) I'll meet him for lunch today.

Here the first sentence is interpreted as adding a new element to the context[8], corresponding to "the president of the board", "a friend from Amsterdam" or "Thomas". In each of these cases it is possible for the second sentence to refer back to this previously introduced entity by means of the pronoun "him".

On the other hand, it is well-known that we cannot just refer back to *any* noun phrase or quantifier of the preceding discourse, as indicated by the following discourse ("#" denotes the sentence is incoherent in the given discourse).

[8] We stay, for the moment, deliberately vague as to the nature of the context and the elements in it. We will give a more concrete interpretation in what follows.

(3.9) Four of my friends/nobody/everyone arrived yesterday.
(3.10) # I'll meet him for lunch today.

Here none of the expressions "four of my friends", "nobody" or "everyone" can be referred to by the pronoun "him" and a theory of anaphoric reference needs to explain this. Though the contrast between plural and singular explains at least part of the difference, it does not explain all of it: neither "him" nor "them" can refer to "nobody".

As another set of examples (after van Eijck and Kamp, 2011), look at the following contrast.

(3.11) Someone didn't smile.
(3.12) He had a terrible headache.

(3.13) Not everyone smiled.
(3.14) # He had a terrible headache.

Though from a truth-conditional point of view "Someone didn't smile" and "Not everyone smiled" are logically equivalent, they are clearly different in terms of the anaphoric possibilities they allow, as shown by the second sentence of each of the two minimal discourses above.

As one final example, let's return to Example 3.5, repeated as the first sentence below.

(3.15) Every child ate a pizza.
(3.16) It had lots of ham and mozzarella.

In this case, continuing the discourse with the second sentence forces us to give the first sentence the interpretation where there is a single pizza which was eaten by all the children.

One very successful theory of dynamic semantics which gives an account of all these and many more phenomena is Discourse Representation Theory (Kamp and Reyle, 1993). The basic objects of Discourse Representation Theory (DRT) are Discourse Representation Structures. Dynamic semantics and Discourse Representation Theory are both fields with a long history and a large volume of research and this short section cannot really do more than give a flavor of some of the very basic ideas. For a more detailed treatment of these subjects, the reader is referred to (Kamp and Reyle, 1993; Muskens et al, 2011; van Eijck and Kamp, 2011; Kamp et al, 2011).

Figure 3.1 shows two examples Discourse Representation Structures (DRSs).

It shows the two readings of "Every child ate a pizza" which we computed for Example 3.5 in DRS form. Each DRS is drawn as a box, divided into two parts by a horizontal line. Above the line, there is a list of variables which are called the *discourse referents*. The set of discourse referents is sometimes called the *universe* of the DRS. Below the line are the *conditions* of the DRS: these can be predicates, like $pizza(x)$ but also complex conditions, which recursively contain other DRSs. In the example there is an implication $\Rightarrow$ between two DRSs, with the obvious

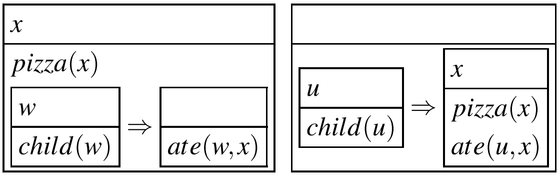

Fig. 3.1. Two example Discourse Representation Structures corresponding to the two readings of Example 3.5

intended meaning that *if* the DRS on the left of the arrow evaluates to true *then* the DRS on the right of the arrow evaluates to true. The intuitive meaning of a DRS is that the discourse referents are existentially quantified variables (and, by duality, the discourse referents on the left hand side of an implication are universally quantified).

Definition 3.6 (DRS). *A discourse representation structure is a pair $\langle V, C\rangle$ such that V is a set of variables and C is a set of conditions.*

A condition is either an atomic condition, a predicate $p(x_0,\dots,x_n)$ where all the x_i are discourse referents (we treat $x = y$, often called anaphoric link, as a two-place predicate which is interpreted as the identity relation) or a complex condition: if K_1 and K_2 are DRSs then $K_1 \Rightarrow K_2$, $K_1 \vee K_2$ and $\neg K_1$ are DRS conditions.

One of the crucial notions is the notion of *accessibility* of discourse referents: if we add a sentence containing an anaphoric pronoun, such as “he”, “she” or “it” to a DRS, we need to link it to an accessible discourse referent. We say a discourse referent x is accessible from DRS K if:

1. x is a member of the universe of K,
2. K is a DRS which occurs as a condition $\neg K$, $K \vee K_1$, $K_1 \vee K$, $K \Rightarrow K_1$ or $K_1 \Rightarrow K$ in a DRS K_p and x is accessible from K_p,
3. K is a DRS which occurs as a condition $K_1 \Rightarrow K$ and x is accessible from K_1.

So, intuitively, from a DRS K we can “see” all discourse referents which occur in a DRS which contains K or which occur to the left of an implication from K. If x_i is an accessible discourse referent occurring in an atomic condition $p(x_0,\dots,x_n)$ we will call x_i *bound*[9]; if not, we will call x_i *free*. A DRS without free variables is called a *proper* DRS.

Given this definition of accessibility, we see that for the example DRSs of Figure 3.1, the outermost level of the leftmost DRS has only the discourse referent x which is accessible in the “outer” DRS, whereas w is accessible for the DRS to the left of the implication and both w and x are accessible DRS to the right of the implication. The rightmost DRS has no discourse referents accessible at the outer level, u is accessible in the DRS left of the arrow and both u and x are accessible right of the arrow.

[9] It is possible to distinguish between two types of bound occurrences (see van Eijck and Kamp, 2011, for further details and discussion), however, in order to keep the current discussion simple, we will not do so here.

As a slight abuse of notation, we will write $x = ?$ for an unresolved anaphoric link, indicating that we need to substitute an accessible discourse referent for '?' before we can interpret the DRS. In other words, the actual resolution of anaphora, though subject to syntactic constraints, is handled in the semantics. The alternative is to handle binding in the syntax, by coindexing an anaphor and its antecedent, then ensuring in the translation that expressions labeled by the same index are translated by the same variable (see Muskens, 1996, for example). We will have a bit more to say on this topic when we talk about the semantics assigned to "he".

A useful operation on two DRSs K_1 and K_2 is the *merge* operation. The idea behind the merge operation is that we add the information of K_2 to the information of K_1 to obtain a new DRS K. We can see K_1 as the DRS corresponding to the preceding discourse, K_2 as the DRS corresponding to the sentences which have just been uttered and their merge K as the DRS corresponding to the resulting discourse. Defining merge is easy when the universes of K_1 and K_2 are disjoint and none of the conditions of either DRS contains free variables. However, this seems much too restrictive. In order to give an appropriate account of binding phenomena, we allow the discourse referents of K_1 to bind the variables of the conditions in K_2, but not vice versa and we rename the variables in the universe of K_2 in order to avoid naming conflicts with variables of the universe K_1, as indicated by the following definition (this is not the only possibility; see van Eijck and Kamp, 2011, for a discussion of different strategies for defining merge).

Definition 3.7 (Merge). *Let K_1 and K_2 be two Discourse Representation Structures with variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ respectively. We define the* merge *of K_1 and K_2, written $K_1 \oplus K_2$ as follows*

- *we replace all free variables x_i of K_1 which are identical to one of the y_i of K_2 by unused variables v_i, obtaining a new DRS K_1',*
- *we replace all bound variables y_i of K_2 which are identical to one of the x_i of K_1 by unused variables z_i, obtaining a new DRS K_2',*
- *$K_1 \oplus K_2$ is defined as the union of the discourse referents of K_1' with the discourse referents of K_2' and the union of the conditions of K_1' with the conditions of K_2'.*

Note that the merge of two DRSs is always defined and unique up to variable renaming, but asymmetric: discourse referents of K_1 can bind variable occurrences of conditions of K_2 (and this is, of course, the essence of anaphoric reference) but discourse referents of K_2 can never bind conditions of K_1; free variables of K_1 will be free variables of $K_1 \oplus K_2$ by definition.

The relation between Discourse Representation Structures and first-order logic is rather transparent, as shown by the following translations.

Definition 3.8. *We define two translations: one from first-order logic — where we restrict the translation to formulae which use a different variable for each quantifier and to formulae for which no free variable uses the same variable name as a quantified variable — to Discourse Representation Structures, shown in Table 3.1 and one from Discourse Representation Structures to first-order logic, shown in*

Table 3.2. Since Discourse Representation Structures and conditions are defined by mutual recursion, the translation from DRSs and conditions is defined by two mutually recursive function d (from DRSs to formulas of first-order logic) and c (from DRS conditions to formulas of first-order logic) as well.

Table 3.1. Translating first-order formulae into Discourse Representation Structures

$$\|\exists x.F\| = \begin{array}{|c|}\hline x \\ \hline \\ \hline\end{array} \oplus \|F\|$$

$$\|\forall x.F\| = \begin{array}{|c|}\hline \\ \hline \begin{array}{|c|}\hline x \\ \hline \\ \hline\end{array} \Rightarrow \|F\| \\ \hline\end{array}$$

$$\|F_1 \wedge F_2\| = \|F_1\| \oplus \|F_2\|$$

$$\|F_1 \Rightarrow F_2\| = \begin{array}{|c|}\hline \\ \hline \|F_1\| \Rightarrow \|F_2\| \\ \hline\end{array}$$

$$\|F_1 \vee F_2\| = \begin{array}{|c|}\hline \\ \hline \|F_1\| \vee \|F_2\| \\ \hline\end{array}$$

$$\|\neg F\| = \begin{array}{|c|}\hline \\ \hline \neg\ \|F\| \\ \hline\end{array}$$

$$\|P\| = \begin{array}{|c|}\hline \\ \hline P \\ \hline\end{array}$$

Table 3.2. Translation functions d from Discourse Representations structures to first-order formulae and c from DRS conditions to first-order formulae

$$d(\langle[x_0,\ldots,x_n],[C_1,\ldots,C_m]\rangle) = \exists x_0,\ldots,x_n.c(C_1)\wedge\ldots c(C_m)$$

$$\begin{aligned}
c(p(x_0,\ldots,x_n)) &= p(x_0,\ldots,x_n)\\
c(D_1 \vee D_2) &= d(D_1) \vee d(D_2)\\
c(\langle[x_0,\ldots,x_n],[C_1,\ldots,C_m]\rangle \Rightarrow D) &= \forall x_0,\ldots,x_n(c(C_1),\ldots,c(C_m) \Rightarrow d(D))\\
c(\neg D) &= \neg d(D)
\end{aligned}$$

Muskens (1994, 1996) shows that DRT semantics is fully compatible with both Montague semantics and the Lambek calculus, and that the combination of both approaches into a unified theory based on the simply typed lambda calculus allows us to use the strong points of both formalisms. For example, as pointed out by Muskens (1996), the combined calculus handles the interaction of anaphors with coordination rather easily. In addition, this combination of Montague grammar and DRT, being based on the simply type lambda calculus, can easily handle DRT semantics within the Lambek calculus (as shown by Muskens, 1994).

The essential idea is very simple: we replace all terms of type t by DRSs, which are of type $i \rightarrow (i \rightarrow t)$, where i is an information state: a function which assigns each discourse referent in the universe of the DRS an element of the domain." by "a function which assigns each discourse referent in the universe of the DRS an element of the domain (there is an additional subtlety: Muskens has a type distinction between discourse referents and entities; however, for ease of exposition we will only use the type entity in our discussion). Given that the discourse we have processed before the current sentence may have already assigned some values to discourse referents, what a sentence does is *update* this information — possibly adding new discourse referents and discarding old ones. The interpretation of the conditions of the DRS must therefore be a term of type $i \rightarrow t$, a function which given an information state i — that is, an assignment to the variables in the universe of the DRS — returns true or false; when occurring inside a DRS this term is therefore applied to the current context i to give a term of type t. So the intuition behind the semantics of a DRS in this calculus is that a DRS is interpreted as a relation between input context c_i and an output context c_o, such that all the variables $x_0, \dots, x_n$ are added to c_o (there are different opinions about what to do if any of the x_i is already assigned a value in the assignment c_i, we will only assume here that in the context which follows x_i is treated just like a free variable) and that all the conditions $C_0, \dots, C_n$ are true for the assignment c_o. In case there is no assignment c_o satisfying all C_i then the DRS evaluates to false, otherwise it evaluates to true. If we make the intuitions of this paragraph formally precise, we can — as shown in Muskens (1996, Section III.2) — treat the DRSs as *abbreviations* for terms which manipulate contexts directly. In what follows, we follow the simpler strategy where boxes are simply objects of type $i \rightarrow (i \rightarrow t)$ and conditions are terms of type t — ie. the application of a term of type $i \rightarrow t$ applied to the current context c. This choice means that the accessibility relation and context updates are handled at the level of the DRSs and not at the level of the lambda terms. Though the solution of coding the accessibility in the lambda terms is rather neat, the current solution produces lambda-terms which convert to Discourse Representation Structures and therefore permits a more transparent comparison with DRS semantics.

In what follows, to simplify the notation of types, we will often abbreviate the type $i \rightarrow (i \rightarrow t)$ as σ. The new constants of DRS semantics have the following types.

Constant	**Type**	**Simplified type**
$\neg$	$(i \rightarrow i \rightarrow t) \rightarrow t$	$\sigma \rightarrow t$
$\Rightarrow, \vee$	$(i \rightarrow i \rightarrow t) \rightarrow (i \rightarrow i \rightarrow t) \rightarrow t$	$\sigma \rightarrow \sigma \rightarrow t$
$\oplus$	$(i \rightarrow i \rightarrow t) \rightarrow (i \rightarrow i \rightarrow t) \rightarrow (i \rightarrow i \rightarrow t)$	$\sigma \rightarrow \sigma \rightarrow \sigma$

Example 3.9. Translating the lexicon of Example 3.5 to a DRS lexicon gives us the following.

word	***syntactic type*** u ***semantic type*** u^* ***semantics:*** $\boldsymbol{\lambda}$***-term of type*** u^* x^v ***means that the variable or constant*** x ***is of type*** v
every	$(S\,/\,(np\,\backslash\,S))\,/\,n$ (subject) $((S\,/\,np)\,\backslash\,S)\,/\,n$ (object) $(e\to\sigma)\to((e\to\sigma)\to\sigma)$ $\lambda P^{e\to\sigma}\lambda Q^{e\to\sigma}\ \boxed{\begin{array}{l} \\ \hline (((\Rightarrow^{(\sigma\to\sigma\to t)}\ ((\oplus^{\sigma\to\sigma\to\sigma}\ \boxed{\begin{array}{c} x \\ \hline \ \end{array}})\ (P\,x)))\ (Q\,x)) \end{array}}$
a	$((S\,/\,np)\,\backslash\,S)\,/\,n$ (object) $(S\,/\,(np\,\backslash\,S))\,/\,n$ (subject) $(e\to\sigma)\to((e\to\sigma)\to\sigma)$ $\lambda P^{e\to\sigma}\ \lambda Q^{e\to\sigma}\ ((\oplus^{\sigma\to\sigma\to\sigma}\ ((\oplus^{\sigma\to\sigma\to\sigma}\ \boxed{\begin{array}{c} x \\ \hline \ \end{array}})\ (P\,x)))\ (Q\,x))$
child	n $e\to\sigma$ $\lambda x^e\ \boxed{\begin{array}{c} \\ \hline child^{e\to t}(x) \end{array}}$
pizza	n $e\to\sigma$ $\lambda x^e\ \boxed{\begin{array}{c} \\ \hline pizza^{e\to t}(x) \end{array}}$
ate	$(np\,\backslash\,S)\,/\,np$ $e\to(e\to\sigma)$ $\lambda y^e\ \lambda x^e\ \boxed{\begin{array}{c} \\ \hline ate^{e\to e\to t}(x,y) \end{array}}$
Leon	$S\,/\,(np\,\backslash\,S)$ (subject) $(S\,/\,np)\,\backslash\,S$ (object) $(e\to\sigma)\to\sigma$ $\lambda P^{e\to\sigma}((\oplus^{\sigma\to\sigma\to\sigma}\ \boxed{\begin{array}{l} x \\ \hline Leon(x) \end{array}})(P\,x))$
he	$S\,/\,(np\,\backslash\,S)$ $(e\to\sigma)\to\sigma$ $\lambda P^{e\to\sigma}((\oplus^{\sigma\to\sigma\to\sigma}\ \boxed{\begin{array}{l} x \\ \hline x=? \end{array}})(P\,x))$

Apart from the fact that we have replaced type t by type σ, the types of the lambda-terms have remained them same. Words like “pizza” and “child” are now functions from entities to DRSs with the condition $pizza(x)$ (resp. $child(x)$). The interesting cases are the quantifiers “a” and “every”: “every” is a function from two properties P and Q — that is, two functions from entities to DRSs — to a DRS. The resulting DRS will have a single complex condition of the form $K_1 \Rightarrow K_2$, where the left hand side K_1 will be the merge of a DRS containing only the unused discourse referent x with the DRS (Px) and the right hand side K_2 will be the DRS (Qx). The DRS introducing the unused discourse referent x will therefore bind the occurrences of x in both K_1 and K_2. The existential quantifier "a" will simply merge a DRS containing an unused discourse referent x with the DRSs obtained by applying properties P and Q to x.

While the above lexicon corresponds rather closely for quantifiers and common nouns, we need to change the way we analyze noun phrases a bit. Before, we assigned proper names the syntactic type np, corresponding to the semantic type e. However, in the standard DRS account of names, we want a name *Name* to introduce both a discourse referent x corresponding to this name into the universe of the DRS[10] *and* an atomic condition of the form $Name(x)$. When we lift the np type to $S\,/\,(np \setminus S)$, the semantic type becomes $(e \to \sigma) \to \sigma$ and we can introduce a discourse referent x, with the condition that $Name(x)$, to the universe as shown above.

As for the pronoun "he", it introduces a new discourse referent x, but introduces an anaphoric link as well, requiring this discourse referent to be bound to one of the accessible discourse referents — which we can decide only in the context of a larger DRS, of course. As we said at the beginning of the chapter, many authors consider finding the reference for x to be a question of *syntax*, typically noted by coindexing the pronoun "he" with a noun phrase elsewhere in the discourse. Here, we will treat the resolution of the anaphor to be a *semantic* problem (though subject to various well-known syntactic constraints). The lexical entry, with the condition $x = ?$ notes this unresolved anaphor, indicating it needs to be resolved in order for the final DRS in which it occurs to make sense.

3.7 A Word about Intensional Logic

Let us briefly explain how the method we give for computing semantic representations straightforwardly extends to the intensionality operators popularized in semantics by Montague and his followers.

It is quite common to consider Kripke models which can be referred to within the syntax itself with type $\mathtt{s}$ for worlds and with the $\hat{}$ and $\check{}$ operators. Possible worlds are just common classical models, with an accessibility relation between them, usually a partial order. Phenomena whose truth in a given world depend on truth in other worlds are said to be *intensional*. For instance, to say that A is possible (resp. necessary) at world w means that for some (resp. every) world w' accessible from w we have that A is true in w'.

One is thus able to model modalities and to deal with opaque contexts like belief verbs. For instance, if "Max believes that Chomsky is a computer scientist" is

[10] Actually, names should be added to the *outermost* DRS. However, will ignore this complication here.

true, than we cannot infer that Chomsky is a computer scientist. It actually has little to do with the truth of the believed sentence. A common interpretation is that in any possible world where the beliefs of "Max" are true, "Chomsky" happens to be a computer scientist. In intensional logic, we therefore distinguish between the truth value of a proposition and the proposition itself: we will interpret a proposition as a set of worlds, a term of type $\mathtt{s} \rightarrow \mathbf{t}$: the set of worlds in which the proposition is true. Its truth value is therefore this term of type $\mathtt{s} \rightarrow \mathbf{t}$ applied to the current world w.

In order to provide an account of intensional phenomena, the syntax of the lambda calculus is commonly enriched with two operators $\hat{}\ : \mathtt{s} \rightarrow a$ and $\check{}\ : (\mathtt{s} \rightarrow a) \rightarrow a$ where $\mathtt{s}$ is the type for the (indices of the) possible worlds with $\check{}\hat{}\,u$ reducing to u.

Gallin (1975) proposes to translate the intensional operators in a two sorted logic with two types $\mathtt{s}$ and $\mathbf{e}$ (in addition to $\mathbf{t}$), with the particularity that there is only a single variable of type $\mathtt{s}$. It is possible to see $\hat{}\,t^a$, of type $\mathtt{s} \rightarrow a$, as an abbreviation of $\lambda w^{\mathtt{s}} t^a$ and $\check{}\,t^{\mathtt{s}\rightarrow a}$, of type a, as an abbreviation of $(t^{\mathtt{s}\rightarrow a}\ w^{\mathtt{s}})$ (see also Gamut, 1991; Morrill, 1994). According to Gallin's translation, $\check{}\hat{}\,u$ becomes $(\lambda w^{\mathtt{s}}\ u)w$ which reduces to u by beta reduction. Its inverse $\hat{}\check{}\,u$ corresponds to $\lambda w^{\mathtt{s}}(uw)$, which reduces to u only if there are no free occurrences of w in u.

We have not yet talked about how to add terms with intensional types to our lexicon. There are several strategies for doing this: the simplest method is to change the type map, the morphism ".*" from syntactic types to semantic types, and let it introduce type $\mathtt{s}$. For example, we can replace the basic type $\mathbf{t}$ by $\mathtt{s} \rightarrow \mathbf{t}$, in other words, we replace truth values by propositions, as proposed by Montague (1970a) van Benthem (see also 1991, chaper 12), we will call this the Montague/EFL type map — compare this to Muskens' strategy for interpreting Discourse Representation Structures in type theory as discussed in Section 3.6, which replaces truth values $\mathbf{t}$ by state changes $i \rightarrow i \rightarrow \mathbf{t}$. The Montague/EFL map produces $S^* = \mathtt{s} \rightarrow \mathbf{t}$ and $n^* = \mathbf{e} \rightarrow \mathtt{s} \rightarrow \mathbf{t}$. As another possibility, we can change the type map for the complex types and leave the translations of the atomic types unchanged, translating $(a \setminus b)^*$ and $(b \,/\, a)^*$ as $(\mathtt{s} \rightarrow a^*) \rightarrow b^*$, as proposed by Montague (1973) (see also Gamut, 1991; Hendriks, 1993), we will call this the Montague/PTQ type map. This second map replaces individuals (of type $\mathbf{e}$) by *individual concepts*: functions from possible worlds to entities. This allows expressions such as "the president of France" and "the temperature" to have different denotations in different worlds (Montague requires proper names to always refer to the same individual by means of a meaning postulate).

So according to van Montague's EFL type map, "believes that" of syntactic type $((np \setminus S) \,/\, S)$ would be of semantic type $(\mathtt{s} \rightarrow \mathbf{t}) \rightarrow \mathbf{e} \rightarrow \mathtt{s} \rightarrow \mathbf{t}$, whereas according to Montague's PTQ type map, it would be $(\mathtt{s} \rightarrow \mathbf{t}) \rightarrow (\mathtt{s} \rightarrow \mathbf{e}) \rightarrow \mathbf{t}$.

A final strategy is to add new connectives which are interpreted intensionally by the type map. Morrill (1990) (see also Morrill, 2011, Chapter 8) proposes an extension of the Lambek calculus which adds a modal connective '□', using the logical rules of S4 for this modal operator[11], where $(\Box a)^*$ is mapped to $\mathtt{s} \rightarrow a^*$.

[11] Not to be confused with the '□' connective of Section 5.2.2, but close to the '!' connective of Section 5.2.1, which is an S4 modality as well. Morrill (1994, Chapter 5) proposes to use an S5 modality instead.

In Morrill's calculus, the introduction rule for this modal operator corresponds to ${}^{\wedge}$ and the elimination rule to ${}^{\vee}$, adapting an idea from van Benthem (1986). In Morrill's intensional system "believes that" is assigned syntactic type $\Box((np\backslash S)/\Box S)$, which corresponds to the semantic type $\mathrm{s} \to (\mathrm{s} \to \mathbf{t}) \to \mathbf{e} \to \mathbf{t}$. Note how we mark explicitly that the sentential argument of believes is intensional by assigning it the syntactic type $\Box S$.

Let us conclude this very brief indication of the relation between intensional logic and categorial grammars by giving the lexical lambda terms corresponding to "believes that" for each of the three semantic types we have seen.

Montague/EFL	$(\mathrm{s} \to \mathbf{t}) \to \mathbf{e} \to \mathrm{s} \to \mathbf{t}$	$\lambda p^{\mathrm{s}\to\mathbf{t}}\lambda x^{\mathbf{e}}\ {}^{\wedge}\mathrm{believe}(x,p)$
Montague/PTQ	$(\mathrm{s} \to \mathbf{t}) \to (\mathrm{s} \to \mathbf{e}) \to \mathbf{t}$	$\lambda p^{\mathrm{s}\to\mathbf{t}}\lambda y^{\mathrm{s}\to\mathbf{e}}\ \mathrm{believe}({}^{\vee}y,p)$
Morrill	$\mathrm{s} \to (\mathrm{s} \to \mathbf{t}) \to \mathbf{e} \to \mathbf{t}$	${}^{\wedge}\lambda p^{\mathrm{s}\to\mathbf{t}}\lambda x^{\mathbf{e}}\ \mathrm{believe}(x,p)$

3.8 Concluding Remarks

Though we have touched only very briefly on many of the topics which we have discussed in this chapter, they include many of the "classic" topics in semantics: generalized quantifiers, intensionality, anaphora and Discourse Representation Theory. Carpenter (1996) discusses many more. So while the idea of combining Montague semantics and the Lambek calculus is very simple, its applications cover a wide range of semantically interesting phenomena.

Montague grammar and Discourse Representation Semantics are active and independent fields and we recommend (Portner and Partee, 2002; Partee and Hendriks, 2011; van Eijck and Kamp, 2011) as a starting point for the reader interesed in exploring these topics further.

Exercises for Chapter 3

Exercise 3.1. Reduce/expand the following lambda-terms into β normal η long form.

1. $f^{e\to(e\to t)}$
2. $(\lambda x^{e\to t} f^{(e\to t)\to(e\to t)} x) y^{e\to t}$
3. $((\lambda x^e \lambda y^e f^{e\to(e\to t)}(x,y))\ (h^{e\to e}(y)))\ c^e$
4. $(\lambda y^e f^{e\to(e\to t)}(y,y))((\lambda x^e g^{e\to(e\to e)}(x,x))\ c^e)$
5. $(\lambda x^e f^{e\to(e\to t)} y^e)(g^{e\to e} z^e)$

Exercise 3.2. Compute the types corresponding to the following formulae.

1. $n\,/\,n$
2. $(n\,\backslash\,n)\,/\,n$
3. $(n\,\backslash\,n)\,\backslash\,(S\,/\,np)$
4. $((np\,\backslash\,S)\,/\,np)\,\backslash\,(np\,\backslash\,S)$

Exercise 3.3. Using the terms and types described in the paragraph on *Systematic Type Raising* in Section 3.5 on page 84, derive "the cat sleeps" and calculate its lambda term.

Exercise 3.4. Assign a lambda-term to "is" of syntactic type $(np\,\backslash\,S)\,/\,(n\,/\,n)$ in such a way that "Leon is vegetarian" will produce a semantics equivalent to *vegetarian*(L).

Similarly, give a lambda-term to "is" of syntactic type $(np\,\backslash\,S)\,/\,np$ in such a way that "Leon is a vegetarian" will produce a semantics equivalent to *vegetarian*(L). Use the standard semantics for "a" and assign "vegetarian" the syntactic type n. Justify your answer.

Hint: assume a constant "=" of type $\mathbf{e} \to \mathbf{e} \to \mathbf{t}$ which evaluates "$= (x,y)$" (or $x = y$ in more convenient infix notation) to true iff x and y have the same denotation.

Exercise 3.5. For the semantics of adjectives, they are often classed into different groups according to the inference patterns they allow. Look at the following examples, all of which are of syntactic type n.

(3.17) vegetarian assassin
(3.18) efficient assassin
(3.19) alleged assassin

Example sentence 3.17 is an example of what is often called an *intersective* adjective. It is intersective because a vegetarian assassin is someone who is both a vegetarian and an assassin.

Example sentence 3.18 is different: suppose Leon is not just an efficient assassin but also an amateur gardener. If we know Leon is an efficient assassin, we want to be able to infer that Leon is an assassin. However, we do *not* want to conclude that he is an efficient gardener (an assassin's life may not leave him with enough time to water his plants....).

For the final example sentence, if someone is an “alleged assassin”, then he may or may not be an assassin.

Using the same syntactic type $n\,/\,n$ for all adjectives, assign semantic terms of the right type to each word. Verify that the three different inference patterns hold for the different lambda-terms.

Note: strictly speaking, inferences are between sentences. However, the notion extends rather naturally to other expressions (Keenan and Faltz, 1985). If you have trouble convincing yourself that the given inference patterns hold, use your solution for Exercise 3.4 and prefix the nouns and adjectives with “Leon is a” (resp. “Leon is”).

Exercise 3.6. Extend the lexicon of Example 3.5 with the appropriate syntactic types and lambda-terms for the generalized quantifiers “at least two” (for the current example, treat it as a single word *at_least_two*) and “no”. Add simple lexical entries for “pizzas” and “children”.

Compute the lambda-terms which correspond to the following two sentences.

(3.20) Every child ate at least two pizzas.
(3.21) No child ate at least two pizzas.

Comment on the difference between the subject wide scope and object wide scope readings. Is one reading more plausible than another?

Exercise 3.7. Give a lexicon which assigns “John” the syntactic type np, the word “and” an instantiation of the scheme $(X \setminus X)\,/\,X$ for some X.

(3.22) Every man and every child .
(3.23) Every man and child
(3.24) John and every child

Exercise 3.8. Following Example 3.4, assign formulas and lambda terms of the correct semantic types to the lexicon in order to derive the following sentence.

(3.25) *Chaque chat que je connais dort beaucoup.*
Every cat that I know sleeps a lot.

‘Every cat I know sleeps a lot.’

Exercise 3.9. Use the lexicon of Example 3.9 to compute the DRSs corresponding to the two readings of “every child ate a pizza”

Exercise 3.10. Extend the lexicon of Example 3.9 in order to treat the following two example sentences from (Muskens, 1996).

(3.26) A cat catches a fish and eats it.
(3.27) # A cat catches no fish and eats it.

Compute the DRSs for both sentences and resolve the anaphor for “it”. Make sure that your semantics for “no” — though it must introduce a discourse referent, of course — does not introduce a discourse referent which can corefer with “it”

References

Abramsky, S.: Computational interpretations of linear logic. Theoretical Computer Science 111, 3–57 (1993)

Amblard, M., Lecomte, A., Retoré, C.: Categorial minimalist grammars: From generative syntax to logical form. In: van Benthem, J., Moortgat, M. (eds.) Linguistic Analysis — Festschrift for Joachim Lambek. Linguistic Analysis, vol. 36, pp. 273–306 (2010)

van Benthem, J.: Categorial grammar. In: Essays in Logical Semantics, ch. 7, pp. 123–150. Reidel, Dordrecht (1986)

van Benthem, J.: Language in Action: Categories, Lambdas and Dynamic Logic. Sudies in logic and the foundation of mathematics, vol. 130. North-Holland, Amsterdam (1991)

van Benthem, J., ter Meulen, A. (eds.): Handbook of Logic and Language, 2nd edn. North-Holland Elsevier, Amsterdam (2011)

Carpenter, B.: Lectures on Type-Logical Semantics. MIT Press, Cambridge (1996)

van Dalen, D.: Logic and Structure, 4th edn., Universitext. Springer (1983)

Dowty, D., Wall, R.E., Peters, S.: Introduction to Montague Semantics. In: Classic Titles in Linguistics. Springer (1981)

van Eijck, J., Kamp, H.: Representing discourse in context. In: Van Benthem and ter Meulen, ch. 3, pp. 181–252 (2011)

Gallin, D.: Intensional and Higher-Order Logic: With Applications to Montague Semantics. Elsevier (1975)

Gamut, L.T.F.: Logic, Language and Meaning, vol. 2. The University of Chicago Press (1991)

Girard, J.Y., Lafont, Y., Taylor, P.: Proofs and Types. Cambridge Tracts in Theoretical Computer Science, vol. 7. Cambridge University Press (1988)

Heim, I., Kratzer, A.: Semantics in generative grammar. Blackwell textbooks in linguistics. Blackwell, Oxford (1997)

Hendriks, H.: Studied flexibility: Categories and types in syntax and semantics. PhD thesis, University of Amsterdam, ILLC Dissertation Series (1993)

Huet, G.: Résolution d'équations dans des langages d'ordre 1,2,...,ω. PhD thesis, Université Paris VII (1976)

Jackendoff, R.: The Architecture of the Language Faculty. Linguistic Inquiry Monographs, vol. 28. MIT Press, Cambridge (1995)

Kamp, H., Reyle, U.: From Discourse to Logic. D. Reidel, Dordrecht (1993)

Kamp, H., van Genabith, J., Reyle, U.: Discourse representation theory. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. 15, pp. 125–394. Springer (2011)

Keenan, E., Faltz, L.: Boolean Semantics for Natural Language, Synthese Language Library, vol. 23. D. Reidel (1985)

Krivine, J.L.: Lambda Calcul — Types et Modèles. Etudes et Recherches en Informatique, Masson, Paris (1990)

Montague, R.: English as a formal language. In: Visentini, B. (ed.) Linguaggi nella società e nella tecnica (1970a); reprinted as Chapter 6 of Thomason (1974)

Montague, R.: Universal grammar. Theoria 36, 373–398 (1970b); reprinted as Chapter 7 of Thomason (1974)

Montague, R.: The proper treatment of quantification in ordinary English. In: Hintikka, J., Moravcsik, J., Suppes, P. (eds.) Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics. Reidel, Dordrecht (1973); reprinted as Chapter 8 of Thomason (1974)

Moortgat, M.: Categorial type logics. In: van Benthem, J., ter Meulen, A. (eds.) Handbook of Logic and Language, ch. 2, pp. 93–177. North-Holland Elsevier, Amsterdam (1997)

Morrill, G.: Intensionality and boundedness. Linguistics and Philosophy 13(6), 699–726 (1990)

Morrill, G.: Type Logical Grammar. Kluwer Academic Publishers, Dordrecht (1994)

Morrill, G.: Categorial Grammar: Logical Syntax, Semantics, and Processing. Oxford University Press (2011)

Muskens, R.: Categorial Grammar and Discourse Representation Theory. In: Proceedings of COLING 1994, Kyoto, pp. 508–514 (1994)

Muskens, R.: Combining Montague Semantics and Discourse Representation. Linguistics and Philosophy 19, 143–186 (1996)

Muskens, R., van Benthem, J., Visser, A.: Dynamics. In: Van Benthem and ter Meulen, ch. 10, pp. 587–688 (2011)

Partee, B., Hendriks, H.: Montague Grammar. In: Van Benthem and ter Meulen, ch. 1, pp. 3–94 (2011)

Portner, P., Partee, B.H. (eds.): Formal Semantics: The Essential Readings. Blackwell Publishers (2002)

Pustejovsky, J.: The generative lexicon. MIT Press (1995)

Seldin, J.P., Hindley, J.R.: To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism. Academic Press (1980)

Thomason, R. (ed.): Formal Philosophy: Selected papers of Richard Montague. Yale University Press (1974)