1. Compositionality

1.1 Frege's Principle

Any decent language, whether natural or artificial, contains more than just finitely many expressions. In order to learn and understand all these expressions, i.e. to associate a meaning with each of them, it does not suffice to learn just a vocabulary list: there must be some systematic way of associating meanings with forms – just like there is a systematic way of construing these forms, viz. syntactic rules. Indeed, it seems plausible to assume that there is some connection between the rules that govern what an expression is and those that say what it means. An initially plausible assumption on how this connection is made is a principle that has been attributed to Frege but which, for reasons of philological accuracy, should perhaps better be called by some neutral term like:

The Principle of Compositionality

The meaning of an expression is uniquely determined by the meanings of its parts and their mode of combination.

Something should be said about the meanings of the expressions used in stating this principle. As it stands, the principle presupposes some part/whole relation among linguistic expressions. It should be clear that this must be provided by some syntactic account of the language in question. Thus, e.g., under a straightforward constituent analysis the sentence (1) contains (2), but certainly not (3) as a part, even though (1) does contain the former in some other, 'physical' sense:

(1) ***Whenever Tom sees a lollipop he wants to eat it.***
(2) ***Tom sees a lollipop.***
(3) ***Tom sees a lollipop he wants to eat.***

More complicated part/whole relations may arise out of more involved syntactic descriptions. It may thus not be entirely implausible to engage some transformational analysis that partly reduces (1) to something like (4), which would then, in a sense, be a part of (1):

(4) ***Tom wants to eat the lollipop.***

We do not seriously want to defend any such analysis but merely point out that the notion of a *part* of an expression is not as innocent and straightforward as it may appear from the above formulation of Frege's Principle.

A second possible misunderstanding of the Principle of Compositionality concerns *ambiguous* expressions like the following two:

(5) ***Linguists discovered a hidden ambiguity resolution method.***
(6) ***Not all rulers are straight.***

Since (5) has more than one meaning, we cannot expect 'it' to be (uniquely) determined by whatever the principle assumes. So we must either put the whole principle in the plural, or exclude ambiguous expressions, or understand 'expressions' to refer to *underlying (syntactic) structures* which we may assume to be unambiguous. We take the third option, extending it even to clear cases of *lexical ambiguity* which we also assume to be resolved on some syntactic level at which, e.g., subscripts distinguish at least two different nouns ***ruler***.

A third source of possible unclarity is a meaning's being uniquely *determined by* something else. In the present context, this should be taken in the weakest sense possible, viz. as functional dependence: the meaning of the whole is uniquely determined by the meanings of the parts if there is a function taking the meanings of the parts as its arguments and yielding the meanings of complex expressions as its values. Whether and in what sense this function is known to the language users and how complex it is (whether it is recursive etc.) will be of no concern to us here.

If we know all parts of a complex expression and at the same time know how they combine, we know the expression itself. Given this reasoning, it might appear that the Principle of Compositionality applies vacuously: the parts of an expression and the way they are combined determine the expression which in turn determines its meaning in the weak sense that the latter is unique. However, Frege's Principle is stronger than that. For it is not the parts themselves that we have to combine but their *meanings*. In order to see the exact meaning of this, consider two distinct expressions the corresponding parts of which have the same meanings:

(7) ***My brother, who lives in Aachen, is an oculist.***
(8) ***My brother, who dwells in Aix-la-Chappelle, is an eye-doctor.***

Let us, for the purpose of this discussion, imagine that each word in (7) has exactly the same meaning as the corresponding word in (8). Since the way the words are combined in these two sentences is obviously the same, the Principle of Compositionality immediately implies that (7) and (8) must have the same meaning. Clearly, this is a non-trivial claim.

Not trivial, but not very exciting either; for all it says is that replacing one or more words (ultimate constituents) by synonyms results in a synonymous complex expression. And it is easily seen that this is the only kind of prediction one gets by literally applying Frege's Principle. However, there is a different reading of the term *parts* that leads to a considerable strengthening of the principle: if by a *part* of an expression we mean an *immediate part* of it, we get slightly more substantial claims about synonymies. From the (assumed) synonymy of the noun phrases ***Tom*** and ***everyone who is identical with Tom*** we may, e.g., conclude that (9) and (10) must have the same meaning:

(9) ***Tom is asleep.***
(10) ***Everyone who is identical with Tom is asleep.***

At least under a straightforward analysis, the immediate parts of (9) are the subject ***Tom*** and the predicate ***is asleep***; those of (10) are the subject ***everyone who is identical with Tom*** and the same predicate as in (9). Since the subjects are supposed to be synonymous and the predicates are anyway, Frege's Principle implies that so are the sentences. But it should be noted that this implication only holds under the strong reading of 'part' as 'immediate part'; otherwise the internal combination of the lexical material in (9) and (10) would make the principle inapplicable.

A generalization of this kind of argument reveals an important implication of Frege's Principle, viz. the *Substitutivity of Synonyms*: replacing one part of an expression by a synonymous one results in a synonymous expression. (9) vs. (10) is already an example and by embedding it we see the validity of the general principle:

(11) ***Alain erroneously believes that Tom is asleep.***
(12) ***Alain erroneously believes that everyone who is identical with Tom is asleep.***

Having established the synonymy of (9) and (10) we can now go on like this: since (9) is an immediate part of ***that Tom is asleep*** and (10) is an immediate part of ***that everyone who is identical with Tom is asleep***, the two ***that***-clauses must be synonymous, again by the Principle of Compositionality. Moreover, the complex verb ***erroneously believes*** is clearly synonymous with itself and, once more, combining it with either ***that Tom is asleep*** or the synonymous ***that everyone who is identical with Tom is asleep*** must result in two synonymous verb phrases. One more step like that and we're home; we leave it to the reader. The general conclusion to be drawn from this is that the Principle of Compositionality implies:

The Substitution Principle

Synonymous parts may be substituted for each other without changing the meaning of the complex expression in which they occur.

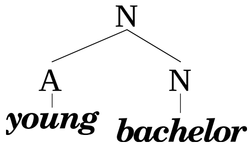
The Substitution Principle will prove to be helpful in evaluating Frege's Principle. However, it must be kept in mind that any apparent counterinstance to the Substitution Principle does not necessarily constitute an argument against compositionality but may equally well serve as evidence against one of its underlying assumptions like, e.g., a certain syntactic analysis or the notion of meaning involved. Concerning the latter, we will gradually see that the Principle of Compositionality can be regarded as a restriction on what meanings are; some simplistic concepts of meaning are incompatible with Frege's Principle.

Before looking at specific examples and counter-examples, we must finally get clear about the *status* of the Principle of Compositionality: is it an empirical hypothesis, a methodological assumption, or what? This depends on how exactly we want to understand it. On one, rather strong reading the principle says that the only (empirically correct) way of assigning meanings to complex expressions is compositional. It turns out that this reading of the principle immediately leads to problems: as we will see in section 1.3, a very straightforward interpretation of predicate logic is non-compositional and it is not unlikely that the characteristic features responsible for this failure carry over to natural language. So we will not favour an empirical reading of Frege's Principle but rather a weaker, methodological one: a compositional (and empirically correct) way of assigning meanings to expressions is to be preferred to its non-compositional rivals. Since it can be shown that, under reasonable assumptions, there is always *some* (empirically correct) compositional way of assigning meanings to complex expressions, the Principle has a universal applicability. But why should we adopt it? Because it teaches us something about the complexity of meaning. This will best be understood from some standard examples to which we will now turn.

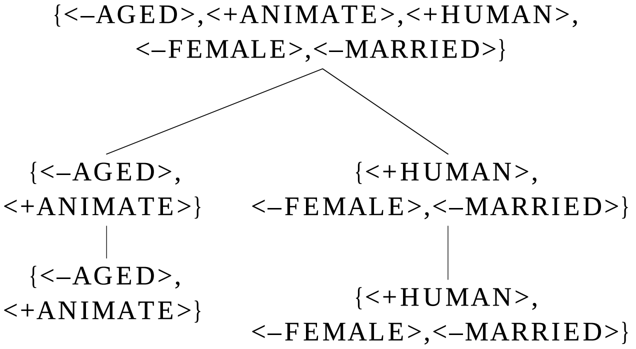
1.2 Compositional meaning assignments: some examples

In order to get some feeling for the content of Frege's Principle, let us first look at some cases in which it clearly applies. The easiest one is a traditional analysis of meanings in terms of *semantic features* or *components*. According to this approach, meanings are sets (or maybe lists) of semantic features which themselves are the ingredients of some complex universal conceptual structure. The basic idea is that meaning relations hold among expressions in virtue of the conceptual relations holding among the features that these expressions mean. Thus, e.g., one reading of the word ***bachelor*** may be analyzed into the components <+HUMAN>, <–FEMALE>, and <–MARRIED>, and its semantic relation to one reading of ***man*** can be read off the latter's presumed meaning {<+HUMAN>, <–FEMALE>}: subsethood amounts to *hyponymy*. (Note that, while we *represent* these components in some binary feature notation mainly derived from English, they *are* abstract conceptual entities.) Given this approach, it is at least tempting to analyze meanings of complex expressions like (13) as the unions (or concatenations) of the meanings of their parts, as shown in (13'):

(13)



(13')


{<–AGED>,<+ANIMATE>,<+HUMAN>, <–FEMALE>,<–MARRIED>}
{<–AGED>, <+ANIMATE>}
{<+HUMAN>, <–FEMALE>,<–MARRIED>}
{<–AGED>, <+ANIMATE>}
{<+HUMAN>, <–FEMALE>,<–MARRIED>}

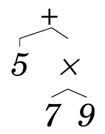
(Note that, due to *redundancy rules* governing conceptual structure, the top node of (13') might turn out to be equivalent to {<–AGED>, <+HUMAN>,<–FEMALE>,<–MARRIED>}.) It is obvious that this analysis, however naive and narrow in its scope, is in accord with the Principle of Compositionality.

An equally simple and familiar example of compositional semantics is provided by the interpretation of (constant) arithmetic terms built up from numerals *1*, *2*, *3*, etc. and the symbols '+' and '×' for addition and multiplication. If we take the numerals to be lexical expressions (or ultimate constituents) and identify the meaning $[\![\delta]\!]$ of such a term δ with the number it denotes, we have a simple compositional meaning assignment, as exemplified in (14):

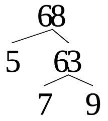
(14) $[\![\mathit{5} + \mathit{(7} \times \mathit{9)}]\!] = [\![\mathit{5}\ ]\!] + [\![\mathit{(7} \times \mathit{9)}]\!] = 5 + ([\![\mathit{7}]\!] \times [\![\mathit{9}]\!]) = 5 + (7 \times 9) = 68.$

In (14), the difference between the numeral '*5*' and the number 5 as well as the corresponding difference between the symbol '+' and the arithmetical operation of addition (denoted by '+') should be noted. Moreover, the bracketing is essential in that it indicates the underlying syntactic structure. What (14) shows is that, if the meanings of complex arithmetical terms are numbers, they can be computed by applying arithmetical operations to the meanings of their immediate parts. How do we know which operation is to be applied in a given case? This obviously depends on the term's *mode of composition*, i.e. whether it is an additive term (= one of the form $\ulcorner(\delta_1 + \delta_2)\urcorner$) or a multiplicative one. Using a tree notation as in (13') and omitting the surface brackets, we thus have:

(14')

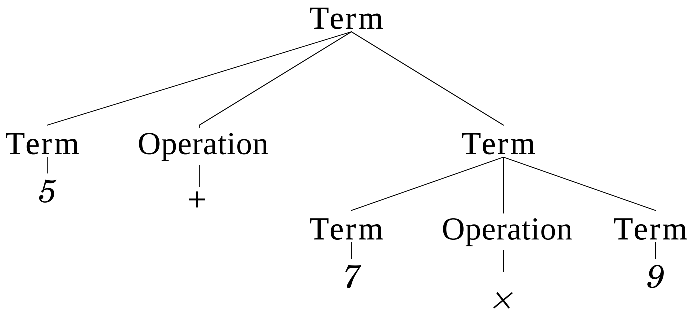


(14")

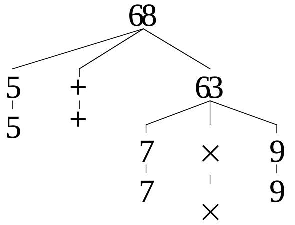


According to the analysis (14'), addition and multiplication are two different term constructions and, following (14"), each of them semantically corresponds to a different operation on numbers or *semantic combination*. Of course, this is not the only way of syntactically analyzing such terms. One may also think of $\ulcorner(\delta_1 + \delta_2)\urcorner$ as consisting of three immediate constituents, the two sub-terms δ_1 and δ_2 plus the operation symbol '+'. (According to (14'), '+' does not really constitute a part of the term but merely indicates the syntactic construction; traditionally speaking, it is analyzed as a *syncategorematic* expression.) In that case we have an equally straightforward and, indeed, very similar compositional interpretation:

(14''')



(14'''')



Thus time the function symbols '+' and '×' are interpreted as denoting operations, whereas the combination of two term meanings (i.e. numbers) and an operation is interpreted by applying the operation to the two term meanings. As far as compositional interpretation is concerned, there is therefore no essential difference between treating the function symbols as syncategorematic or letting them denote (or 'mean') something. We will encounter many analogous situations as we go along.

The third example, the truth-table interpretation of propositional logic, is very similar to the arithmetical one. This time our meanings are the truth-values 0 (= false) and 1 (= true). The meanings of complex expressions can be computed by applying truth-functions to the meanings of their parts:

(15)

$[\![(p\wedge \neg p)]\!]^g = 1$
iff $[\![p]\!]^g = 1$ and $[\![\neg p]\!]^g = 1$
iff $[\![p]\!]^g = 1$ and $[\![p]\!]^g = 0$

The main difference between this example and the preceding ones is that in the case of propositional logic, meaning (truth or falsity) is no longer absolute but rather depends on some function g assigning a truth-value to any basic (atomic) expression. However, the formula in (15) will be assigned the truth-value 0, no matter which function g we may pick.

1.3 Non-compositional meaning assignments: examples

It is now time to look at two counter-examples to the Principle of Compositionality, each of which are, in a sense, extensions of propositional logic. In the first case, we add a one-place operator **T** (that syntactically behaves like negation) and the following interpretation clause:

(16) $[\![\mathbf{T}\varphi]\!]^g = 1$ iff $[\![\varphi]\!]^h = 1$, for all h.

The new operator thus expresses that the formula embedded under it is a tautology in the truth-functional sense. Under the assumption that meanings are truth-values, (16) is a non-compositional rule: we can have: $[\![p]\!]^g = [\![\neg(p\wedge\neg p)]\!]^g = 1$, i.e. p and $\neg(p\wedge\neg p)$ have the same meaning, but only the latter is a tautology and thus:

(17) $[\![\mathbf{T}p]\!]^g = 0 \neq [\![\mathbf{T}\neg(p\wedge\neg p)]\!]^g = 1$.

This example is simple and admittedly artificial. So let us try a real-life case, predicate logic. Again we assume that meanings are truth-values. The interpretation of atomic formulae and connectives are then as in propositional logic. For the quantifiers we have:

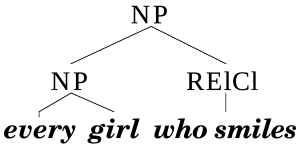
(18) $[\![(\exists x)\varphi]\!]^g = 1$ iff $[\![\varphi[^x/_a]]\!]^h = 1$, for some individual constant a.

('$\varphi[^x/_a]$' denotes the result of replacing all free occurrences of the variable x in the formula φ by the constant a.) This clause, the so-called substitutional interpretation of quantification, only works if everything in the domain of discourse has a name, but at least for the sake of the argument we may assume this to be the case. As a homework exercise will show in detail, (18) is non-compositional, as long as we are identifying meanings and truth-values. We thus face a *compositionality problem*: an initially plausible way of interpreting a certain construction (in our case: quantification) turns out to violate Frege's Principle. There are various things one can do about such a problem, depending on how severe it is and what is intuitively felt as its cause. Let us look at two interesting strategies, one of them concerning quantification.

1.4 Compositionality problems and how to solve them

Suppose we agree on the following branching for restrictive relative clauses:

(A1)



Assume, moreover, that (A2) NPs in general are interpreted as (*generalized*) *quantifiers*, i.e. as sets of sets of individuals. Thus, e.g., we would have:

(19) $\llbracket$***every girl***$\rrbracket = \{X \subseteq D \mid \llbracket$***girl***$\rrbracket \subseteq X\}$
$\llbracket$***no girl***$\rrbracket = \{X \subseteq D \mid \llbracket$***girl***$\rrbracket \cap X = \emptyset\}$
$\llbracket$***some girl***$\rrbracket = \{X \subseteq D \mid \llbracket$***girl***$\rrbracket \cap X \neq \emptyset\}$
$\llbracket$***Jane***$\rrbracket = \{X \subseteq D \mid j \in X\}$,
etc.

(D is the domain of individuals, j is Jane.) Thirdly, it seems to be quite straightforward to have (A3) relative clauses denote sets. So $\llbracket$***who smiles***$\rrbracket = \llbracket$***smiles***$\rrbracket$, etc. We might now wonder whether (?) there is any systematic way of deriving the denotation of a complex NP containing a relative clause from its parts.

(?) is one kind of compositionality problem: we know (or suppose we know) what the meanings of the parts of a given kind of expressions are, we also know what the meaning of the whole expression should be, but we do not know how to combine the parts in order to obtain the resulting meaning. As we will see in a homework exercise, under the assumptions (A1) - (A3), there is no way of combining the meanings of relative clauses and NPs to obtain the desired result: any compositional treatment of (restrictive) relative clauses will have to give up at least one of the assumptions.

The standard way of avoiding (?) is give up the syntactic assumption (A1) and adopt the following bracketing instead: (***every***(***girl***(***who smiles***))), so that the relative clause combines with the noun. One welcome consequence of this strategy is that, according to this analysis, proper names cannot take restrictive relative clauses: they do not contain proper nouns. But then neither would lexical quantifiers like ***nobody***, unless we reanalyzed them as complex (lexical decomposition into ***no*** + ***body***); this is certainly a less welcome result.

An alternative way to compositionally interpret NPs with relative clauses consists in giving up (A2). A brief discussion of that strategy will have to wait till we get to categorial techniques!

Let us now turn to the compositionality problem discussed in the previous section, the interpretation of variable-binding quantifiers. We have seen that the so-called substitutional interpretation of quantification is non-compositional. However, this does not mean that there is no compositional interpretation of predicate logic. One way of getting one (due to Tarski) is by giving up the assumption that meanings (of formulae) are truth-values. The basic idea behind Tarski's construction is that the truth value of $\ulcorner(\exists x)\varphi\urcorner$ does not only depend on φ's *actual* truth-value but rather on all *possible* truth-values that φ might have. The actual truth value depends on what x (actually) refers to whereas other, possible truth-values are determined like the actual one – except that x might refer to something else; in order to define alternative truth-values, we thus need the notion of a *variable assignment*, i.e. a function determining the referent of every variable (including x). The actual truth-value of the formula φ then generally depends on the actual variable assignment and what it does to x; the (actual) truth-value of $\ulcorner(\exists x)\varphi\urcorner$ depends on φ's truth-value at other possible assignments g' that let x refer to alternative referents:

(20) $[\![(\exists x)\varphi]\!]^g = 1$ iff $[\![\varphi]\!]^{g'} = 1$, for some g' that differs from g in (at most) x's value (i.e. $g'(y) = g(y)$ whenever $y \neq x$).

Note that, unlike (!), the actual truth-value of a quantified formula is related to the (possible) truth-values of its (immediate) *parts*. However, this does not quite prove that (20) is compositional: we still need a suitable notion of *meaning*. It cannot be the actual truth-value because then (20) would be non-compositional! How about the possible truth-values, then? Can they be used as the meanings in a compositional interpretation of predicate logic, along the lines of (20)? The answer to this question partly depends on what precisely we mean by 'possible truth-values'. For if we simply mean the set of all truth-values that a formula can possibly have, won't get much further than with the actual truth-value. So a more sophisticated notion is needed. Here it is: the meaning $[\![\varphi]\!]$ of a formula φ of predicate logic is φ's truth-value as it depends on all possible variable assignments or, mathematically speaking, the function taking each variable assignment g to φ's truth-value at g. We can now show that (20) is compositional in that it can be reformulated as a combination of meanings of formulas and variables. (The meaning $[\![x]\!]$ of a variable x is, of course, x's denotation as depends on the assignments, i.e. the function taking each g to g(x).

(21) $[\![(\exists x)\varphi]\!](g) = 1$ iff $[\![\varphi]\!](g') = 1$, for some g' such that $g'(y) = g(y)$, whenever $[\![y]\!] \neq [\![x]\!]$.

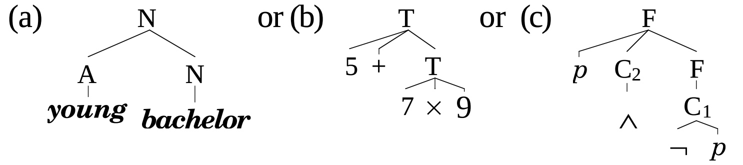
(21) implicitly presupposes that $[\![(\exists x)\varphi]\!]$ is a function from variable assignments to truth-values; thus its value on a given g will be 0 if the above condition is not met. Note that we could now define $[\![(\exists x)\varphi]\!]^g$ as an alternative notation for $[\![(\exists x)\varphi]\!](g)$, i.e. $[\![(\exists x)\varphi]\!]$'s value on g. The only difference between (20) and (21) would then be in the reformulation of the condition '$y{\neq}x$': (21) gives this condition in terms of x's meaning $[\![x]\!]$, as required by compositionality.

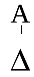
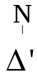
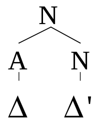
Are (20) and (21) equivalent? In order to see that they actually are, one must first investigate the somewhat neurotic case that the universe of discourse contains exactly one element. (By definition, it cannot be empty.) It is easily seen, that in that case there is only one variable assignment and hence the two clauses amount to: $[\![\varphi]\!](g) = 1$. But if there are (at least) two distinct individuals a and b, we could define an assignment g* by putting: $g^*(x) = a$ and $g^*(y) = b$ for any $y \neq x$; then (20)'s condition $y \neq x$ implies: $[\![y]\!](g^*) = g^*(y) = b \neq a = g^*(x) = [\![x]\!](g^*)$, and hence $[\![y]\!] \neq [\![x]\!]$. Since the other direction (if $[\![y]\!] \neq [\![x]\!]$, then $y \neq x$) is obvious, we have actually proved that (20) can be reformulated in a compositional way.

1.5 Homomorphisms

Up to now we have relied on an intuitive understanding of the main concepts surrounding the Principle of Compositionality. There is, however, a very elegant and general way of making this principle precise without dramatically changing its content. In order to see how it works, we need a general and precise notion of the kind of entities the principle should be applied to, i.e. a universal notion of an (underlying) *structure*. In the examples discussed above, structures were usually represented by *trees* like:

(22)



Note how each structure encodes the way it has been construed using structure-building rules. Thus, e.g., (22a) was built up from the basic (lexical) items ***young*** and ***bachelor*** by applying two lexical rules (R_1: A → ***young***, R_2: N → ***bachelor***) and combining the results with the the attributive-adjective-rule (R_3: N → A + N). We might thus think of the rules as *operations* combining structures. R_3, e.g., combines arbitrary structures of the form  and  into more complex structures  ; the combined sub-structures Δ and Δ' might or might not be lexical items. The tree (a) can thus be rewritten as: R_3(***young***, ***bachelor***). The latter is an algebraic term denoting the result of applying the operation R_3 to two specific lexical structures. In that sense the structure tree (22a) is, or closely corresponds to, an algebraic term.

So instead of having syntactic operations build up trees we might as well think of them as building up terms describing those trees. Each term is either lexical – in which case it cannot be obtained by any operation – or complex and there is only one way of construing it:

(23)

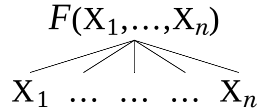
(i) If Y is a lexical term, then there are no (n-place) operations F and terms $X_1,\ldots,X_n$ such that $F(X_1,\ldots,X_n) = Y$.

(ii) If $F(X_1,\ldots,X_n) = G(X'_1,\ldots,X'_n)$, then $F = G$, $n = m$, and $X_i = X'_i$ for all $i \leq n$.

If we replaced 'terms' by 'trees', (23i) would say that the lexicon shouldn't contain any structures that can be construed using the construction rules for logical forms (or whatever the syntactic input to our compositional semantics may be); similarly, (23ii) imposes a restriction on complex structures: they must not be construable in more than one way. Taken together, (23i) and (23ii) would then rule out any kind of ambiguous structures. Thus, e.g., tree transformations yielding the same output for different inputs would violate these conditions. However, the corresponding *terms* describing the *transformational history* of a given tree structure would again be unambiguous in the sense of (i) and (ii): even if the transformation F yields the same tree when applied to distinct structures Δ and Δ', the terms 'F (Δ)' and F (Δ')' would be distinct.

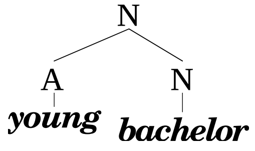
We thus may and, in fact, will assume that the syntactic input to a compositional semantic framework always consists of a collection A of structures and operations satisfying (23i) and (23ii). Given these restrictions, any such structure can be thought of as exactly corresponding to the term describing it: a tree is (almost) a term. Consequently, such A are called *term algebras* (or *free algebras*) in mathematics and they are usually presented as n+1-tuples or sequences of the form A = (A;F_1,…,F_n), where A is the domain of structures (or terms) and the F_1,…,F_n are the structure-building operations. Even though the structures of a term algebra are essentially terms, in order to get a better understanding of the algebraic reconstruction of the compositionality principle, it is helpful to think of them as trees again: a tree is a term is a tree. To every term 'F (X_1,…,X_n)' there is the corresponding tree:

(24)



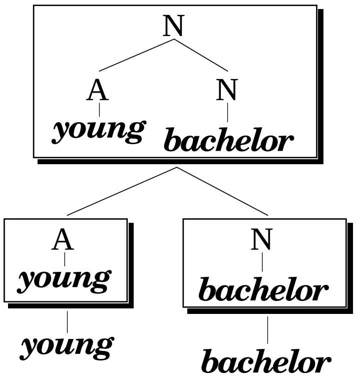
If our initial system of syntactic inputs (logical forms etc.) already met the disambiguity requirements (23i) and (23ii), the trees (24) would closely correspond to the original structures. Thus, e.g., the (24) corresponding to the context-freely generated tree structure:

(25)



would be the hybrid tree:

(25)



which, for all practical purposes, could and should be identified with the original tree. However, the detour via corresponding term algebras is more general in that it can also handle (i.e. differentiate between) identical structures with different histories.

What does a compositional meaning assignment do to a tree? The examples in 1.2 reveal that every node gets its meaning by combining the meanings of its daughters in an appropriate way. Two questions remain: (a) what happens if there are no daughters, i.e. with terminal nodes? And (b) what are the appropriate ways of combining meanings? The answer to (a) is that, in order to make the compositional apparatus work, we will have to *assume* that each lexical item is given its meaning independently, by some *lexical meaning assignment f*. As to (b), appropriateness seems to depend on the particular 'mode of combination', i.e. on the structure-building operation applied in arriving at that node. In other words, a tree of the form (24) would be interpreted by combining the meanings of $X_1,\ldots,X_n$ in the way corresponding to F. In addition to the lexical meaning assignment f we will thus have to assume that, to each structure-building operation F, there corresponds a *semantic operation* G (with the same number of places as F) that assigns meanings to meanings. We may now collect all our semantic operations G and the meanings B to obtain a semantic algebra B = (B;$G_1,\ldots,G_n$). We are then in a position to characterize the main property of a compositional meaning assignment $[\![\ ]\!]$:

(*C*) $[\![F_i(X_1,\ldots,X_n)]\!] = G_i([\![X_1]\!],\ldots,[\![X_n]\!])$.

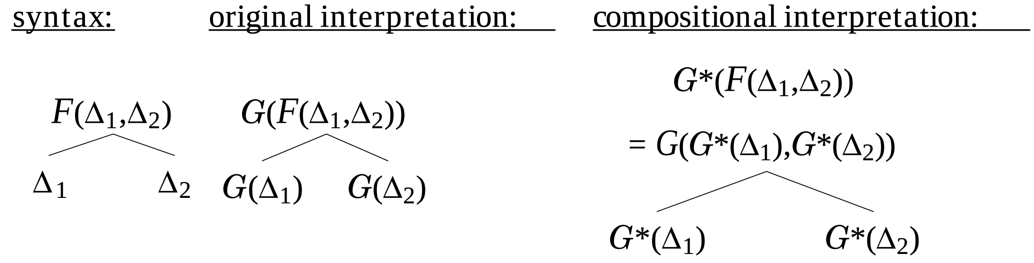
Mathematically speaking, (*C*) says that $[\![\ ]\!]$ is a *homomorphism* from the term algebra A = (B;$G_1,\ldots,G_n$) to the semantic algebra B = (B;$G_1,\ldots,G_n$). The algebraic characterization of compositionality, then, is: *a meaning assignment is a homomorphism from a free algebra to a semantic algebra.*

Intuitively speaking, all we need to know in order to compositionally determine the meaning of a complex structure is the meaning of its ultimate constituents and the relevant semantic operations. In other words, given f and B, the meanings of all structures are uniquely determined by (C) and the fact that $[\![\ ]\!]$ *extends* f (i.e. $f \quad \subseteq [\![\ ]\!]$). This could actually be proved in the algebraic framework sketched here, but we will not do it. We should however, be aware of the fact that the proof essentially depends on the syntactic structures' satisfying (23i) and (23ii).

Does the semantic algebra have to satisfy (23i) and (23ii)? One might expect this to be the case simply because of the existence of a homomorphism. However, this is not so, as a simple example shows. Even though the trees corresponding to the formulae $(p \wedge p)$ and p are clearly different (one is basic, the other is complex), they are always assigned the same truth-value, i.e.: $[\![(p \wedge p)]\!] = [\![p]\!] \times [\![p]\!] = [\![p]\!]$, and hence the meaning of the lexical expression p can be obtained by applying the operation ($\times$) corresponding to conjunction ($\wedge$) to the same expression, thus violating (23i). A similar example can be used to show that the semantic algebra of propositional logic violates (23ii). In fact, it can be shown that, due to the unambiguous nature of term algebras, *any* algebra (with the correct number of operations of the correct numbers of places) can be made the target of a compositional 'meaning' assignment, so that (C) imposes no restriction on possible meanings and their combinations.

We end this part with some speculations about status of the Principle of Compositionality as captured by the algebraic notion of a homomorphism. Let us first observe that in this framework it can be shown that, in a sense, compositionality does not impose any restriction on what an interpretation is; more dramatically: *any meaning assignment can be made compositional*. One proof of this quite surprising result involves a fairly straightforward generalization of Tarski's interpretation of quantification by means of variable assignments. Loosely speaking, the (conceptually) simple strategy of substitutional quantification did not satisfy the compositionality requirement because, in predicate logic, there is more to a variable than just denotation. So the notion of meaning has to be adapted to this case by taking into account everything that the variable might denote, i.e. the function taking every denotation assignment (for variables) to denotation under that assignment. Now we only have to observe that the original, non-compositional notion of meaning in substitutional semantics coincided with denotation. So Tarski's construction replaced non-compositional meanings d (= denotations) of expressions (variables) Δ by functions D_Δ taking arbitrary meaning assignments g (for variables) to the meanings the Δs would have under these assignments g: $D_\Delta(g) = g(\Delta)$. If we now generalize this construction by performing the same trick on arbitrary expressions rather than just variables, it turns out that any meaning assignment, whether compositional or not, will be turned into a compositional one. More precisely, if we are given a function G assigning to every structure Δ of a certain language L a meaning $G(\Delta)$ taken from a set M, we let the corresponding *compositional meaning* $G^*(\Delta)$ of any Δ be that function from M-assignments to members of M that yields $F(\Delta)$ whenever applied to an M-assignment F. (An *M-assignment* is a function from arbitrary structures Δ of L to members of M.) It is easy to check that the compositional meanings of any two structures must be distinct, as long as M has at least two members. (Otherwise compositionality isn't a problem anyway.) In particular, there is a function G taking the compositional meanings $G^*(\Delta_1),\ldots,G^*(\Delta_n)$ of the parts $\Delta_1,\ldots,\Delta_n$ of a complex expression $F(\Delta_1,\ldots,\Delta_n)$ to the compositional meaning of $F(\Delta_1,\ldots,\Delta_n)$: G's arguments $G^*(\Delta_1),\ldots,G^*(\Delta_n)$ uniquely determine $\Delta_1,\ldots,\Delta_n$, to which F and then G^* can be applied: $G(G^*(\Delta_1),\ldots,G^*(\Delta_n)) = G^*(F(\Delta_1,\ldots,\Delta_n))$. Thus G may be regarded as the semantic operation corresponding to the syntactic construction F. Schematically we have:

(26)



Clearly, the new, compositional meanings are much more abstract than the ones that we started with; but, just as in the case of predicate logic there is a simple and canonical way to recover the old, intended meaning of an expression from its new one; we only have to apply the latter to the interpretation function G:

(27) $G(\Delta) = G^*(\Delta)(G)$

However, the price to pay for this achievement is rather high. For it is easily seen that semantic algebra arrived at by this construction is, in effect, a term algebra, isomorphic to the algebra of syntactic structure. This means that instead of applying the complicated procedure indicated in (26), we may as well regard the syntactic trees themselves as 'abstract' meanings and replace the canonical rule (27) by the somewhat less exciting trivial equation (27'), which may be read: the meaning of a syntactic structure *qua* compositional meaning is the original meaning of the same structure:

(27') $G(\Delta) = G(\Delta)$

Of course, one difference between (27) and (27') might be that the former is less trivial and hence more interesting. But there is more to the generalization of Tarski's construction. For, as the case of predicate logic shows, it sometimes helps to apply the procedure 'locally', i.e. only to a certain class of expressions (e.g. variables) and (partial) interpretations (like variable assignments). Now, the general idea underlying the Tarskian strategy is to retain as much as possible of the original, simple but non-compositional analysis and only apply the procedure (26) where the old interpretation leads to non-compositionality. We cannot go into the details of this strategy here but will nevertheless try to indicate its significance. For although it generally leads to more complicated and messy interpretations, it may be of use to assess and compare the complexities of meaning assignments. The more the compositional interpretation obtained by applying Tarski's construction diverges from the original one, the more complex the new meanings will become and, *a fortiori*, the more complex the original interpretation. In that sense compositionality may not always be a desirable property of a meaning assignment; but the Tarski-type compositional interpretations may be regarded as normal forms of semantic theories, and their purpose lies in helping us to understand their complexities.

-

Exercises

1. Show that, under the assumption that predicate logic formulae denote truth-values, the substitutional interpretation (!) of quantification is non-compositional. Hint: Assume that the extension of a given predicate P is neither empty nor identical to the universe of discourse D and consider the formulae $(\exists x)\ P(x)$ and $(\exists x)\ \neg P(x)$.

2. Show that there is no compositional treatment of relative clauses that meets (A1) - (A3) and is equivalent to the standard bracketing plus interpretation, according to which ⟦Det + (N +RelCl)⟧ is obtained by combining ⟦Det⟧ with ⟦N⟧ ∩ ⟦RelCl⟧. Hint: Assume that the extension of the noun ***president*** is a singleton {b} and that its intersection with that of the relative clause ***is wise*** is empty. Then consider the NPs ***every president***, ***some president***, ***every president who is wise*** and ***some president who is wise***.

2. Model-Theoretic Semantics

2.1 Logic and Semantics

Model-theory is a branch of mathematical logic that has proved to be very helpful as a tool of natural language semantics. The main purpose of this chapter is to introduce some of the central ideas and concepts of model theory. But before we even start doing this, let us briefly discuss in what way a mathematical discipline can be of help to the task of describing the meaning of natural language expressions: *Why does logic matter to semantics?* There are at least two reasons:

(a) Logic is concerned with *truth*, semantics with *meaning*, and the two concepts are related, at least by what Cresswell has called the Most Certain Principle: *difference in truth value implies difference in meaning*.

(b) The task (or one task) of a semantic theory is to correctly predict *meaning relations* (or *sense relations*) among expressions. But all meaning relations are (or are reducible to, or variants of) logical relations. Thus, e.g.:

- Hyponymy is (schematic) implication: ***green*** HYP ***coloured*** because *x* ***is green*** (logically?) implies *x* ***is coloured***, whatever *x* is.

- Incompatibility is reducible to implication (and negation): ***green*** INCOMP ***red*** because *x* ***is green*** implies *x* ***is not red***, i.e. the negation of *x* ***is red***.

A *logical relation* among linguistic expressions is one that can be reduced to the notion of (logical) implication. Logical semantics can be understood as the attempt to describe all (relevant) semantic relations in terms of logical relations (and leave the rest to pragmatics). Here are some examples of logical relations:

entailment: this is just the special case of logical implication holding among two sentences.

validity: a valid sentence is one that is implied by any sentence whatsoever.

self-contradictory: a property that a sentence φ has just in case φ implies any sentence whatsoever.

negation: a sentence φ (classically) negates a sentence ψ iff φ and ψ together imply a self-contradiction and the only sentences both of them imply individually are valid.

As many approaches to natural language semantics, logical semantics faces a very general problem that we will refer to as Mates's Trap: it seems impossible to escape the conclusion that synonymy is logical indiscernibility and thus (presumably) weaker than identity. But this seems to contradict the fact that, for any two distinct expressions α and β, the first of the following sentences should be true whereas the second appears to be false:

(M_1)***It is possible for someone to believe that*** α ***is the same as*** α ***without believing that*** α ***is the same as*** β.

(M_2)***It is possible for someone to believe that*** α ***is the same as*** α ***without believing that*** α ***is the same as*** α.

(Note that the precise form of these sentences might have to be changed according to the syntactic categories of α and β.) By the Most Certain Principle, then, (M_1) and (M_2) differ in meaning and thus a compositional meaning assignment would have to assign different meanings to α and β.

In a sense, this problem reveals the limits of logical semantics. This does not necessarily imply that the logical approach to semantics only gives a rough picture of meaning that should ultimately be replaced by something more sensitive (and sensible). It might be that the meaning difference between (M_1) and (M_2) should be analyzed in pragmatic, rather than semantic, terms.

2.2 Models (and Classes of Them)

One leading idea behind the model-theoretic approach is that logical implication is *schematic*, i.e. it holds between sentences if these are instances of general patterns. Thus, e.g., the fact that ***This book is heavy*** implies ***Something is heavy*** is not an isolated observation about these two English sentences. Rather, sentences of the form ⌜NP VP⌝ always imply the corresponding *existential generalization* ⌜***Something*** VP⌝ (provided that the subject NP is referential).

The relevant *inference scheme* is:

(I1) Whenever φ is a true sentence of the form ⌜$\alpha\beta$⌝, where α is a referential NP and β is a VP, then ⌜***Something*** β⌝ is also a true sentence.

We might thus say that (T) ***This book is heavy*** logically implies (∃) ***Something is heavy*** because the two sentences are instances of a general pattern of inference, viz. (I1).

Given the Most Certain Principle (MCP), meaning determines truth-value in the sense that two sentences with the same meaning will have the same truth-value. So to every sentence meaning $[\![\varphi]\!]$, there corresponds a truth-value $V([\![\varphi]\!])$. Moreover, by the principle of compositionality, $[\![\varphi]\!]$ can be obtained by breaking up φ into its parts $\gamma_1,\ldots,\gamma_n$ and combining their meanings $[\![\gamma_1]\!],\ldots,[\![\gamma_n]\!]$ in the way G determined by φ's structure: $[\![\varphi]\!] = G([\![\gamma_1]\!],\ldots,[\![\gamma_n]\!])$. Schemes like (I1) can thus be reformulated with reference to meanings rather than expressions:

(I2) Whenever b_n and b_v are meanings of referential NPs and VPs, respectively, and $V(G(b_n,b_v)) = 1$, then $V(G'([\![\textbf{\textit{something}}]\!]^{M},b_v)) = 1$.

G' occurs instead of G in order to allow for the possibility that the structure of the quantified conclusion differs from that of the premiss with the referential subject; this does not exclude an analysis according to which $G' = G$.

The formulation (I2) of the inference scheme shows that one does not have to know what the meanings of ***this book*** and ***is heavy*** are to see that (T) implies its existential generalization (∃). In other words, whichever meanings b_n and b_v we may choose to assign to the two expressions, the conclusion will be true if the premiss is. The technical term for a way of assigning meanings to (lexical) expressions is *model*. We have thus seen that (T) logically implies (∃) because, in any model M in which (T) is true, (∃) is true; for whichever meanings b_n (= $[\![\textbf{\textit{this book}}]\!]^{M}$) and

b_v (=⟦***is heavy***⟧^{M}) M may assign to ***this book*** and ***is heavy***, the truth-value of (∃) is 1 if only (T) is true:

(I3) If M is any model such that $V(G_M($⟦***this book***⟧^{M}, ⟦***is heavy***⟧^{M})) = 1, then $V(G'_M($ ⟦***something***⟧^{M}, ⟦***is heavy***⟧^{M}) = 1.

(I3) lets the combinations G_M and G'_M of meanings depend on the model M. We will soon see why this is necessary.

Even though (I3) was just a reformulation of (I1), it differs from it in that it can be generalized in different ways. Before we do that, let us first observe that the relation between (I1) and (I3) is analogous to that between the substitutional interpretation of quantification (!) and its more compositional, Tarskian reformulation (!!) discussed in section 1: in (I1) generality, i.e. the schematic character of the implication, is brought out by *substituting* certain expressions with others, just like (!) construes the generality expressed by the quantifier as a statement about substitutions of variables by names; and in (I3) we consider alternative meanings of the expressions actually occurring in the sentences related, just like (!!) has the quantifier look at alternative denotations of the variable it binds. We could even say that schematic implication is (universally) generalized material implication (= truth-table conditional) and that (I1) is its substitutional, (I3) its Tarskian version. [Incidentally, (I3) is close to, but not identical with, Tarski's standard notion of logical implication; (I1) is more like Bolzano's.] It should, however, be noted that the two are only equivalent on the assumption that the meanings assigned by models are precisely the meanings that can be found in some expressions (or possible substitution instances); this corresponds to the tacit assumption in (!) that everything has a name. And it is this assumption that we will give up in our final, model-theoretic account of logical implication. As a motivation of this step, we will briefly sketch an example.

Suppose we interpreted VPs like ***go to the moon*** by sets of people; this is certainly an oversimplification but it helps to see our point more clearly. Since we would not want a sentence like

(1) ***Only Armstrong went to the moon.***

to come out as self-contradictory, at least one model M_0 would have to make it true: otherwise models M that make it true would make any sentence whatsoever true, for there would not be any such M. According to M_0, the interpretation of ***went to the moon*** would be a singleton; and, more importantly for our present purposes, the interpretation of ***did not go to the moon*** would be its complement, i.e. the set C of all individuals different from Armstrong (or whoever the name ***Armstrong*** refers to in M_0.) How many elements does C have? That depends on the entire number of individuals. Let us suppose the universe of discourse contains exactly 520 individuals. Then the sentence

(519) ***Five-hundred-and-nineteen people did not go to the moon.***

would be true in M_0. In fact, it would be true in any model that makes (1) true: for such models would only differ from M_0 as to which set of individuals they assign to which lexical expression – but not with respect to what an individual is, viz. a member of the meaning of some VP. Using the same criterion as we did in (I1) - (I3), we would thus find that (1) implies (519). Well, maybe it does, but this is not what is usually taken to be a logical implication.

The example just sketched is one kind of unintuitive result we would obtain if we were to define models as ways of assigning actual meanings of expressions to (possibly different) expressions. We can avoid it by a more liberal notion of model that includes ways of assigning meanings that do not co-exist in the same model: (1) would no longer have to imply (519) because we might now include models based on a different domain of individuals. (There are other ways of avoiding the result just sketched, based on alternative ways of defining logical implication. We might return to these matters in part 5.) It is this new possibility of allowing models with totally different domains and, in general, ‘mutually incompatible’ meanings that forces us to let the semantic operations depend on models.

In general, then, a *model* has (at least) the following ingredients:

- a realm of meanings;
- a lexical meaning assignment;
- semantic operations;
- a way V to determine truth-values for sentence meanings.

Here are two examples from formal logic:

propositional logic:

- meanings: {0,1};
- lexical meaning assignment: truth-value assignment to atomic formulae;
- semantic operations: truth-tables;
- V_M is identity: truth-values are meanings.

predicate logic:

- meanings: functions from variable assignments to extensions (e.g. truth-values);
- lexical meaning assignment: the mapping from variable assignments g to g(x) + stipulations about predicates and individual constants;
- semantic operations: (!!!) + appropriate clauses for propositional connectives;
- V_M gives the truth value of any given formula under a fixed assignment g_M.

Since we assume the Principle of Compositionality and every model M contains a lexical meaning assignment (plus a specification of the semantic operations), the meaning of any expression α is uniquely determined by M; we will continue to denote this meaning by $[\![\alpha]\!]^M$. Using this notation, we are now in a position to give a general, model-theoretic characterization of logical implication (due to Tarski):

A set of sentences Σ (of a given language L) *logically implies* a sentence φ (of L) iff in every (intended) model M in which all of Σ is true (i.e. $V_M([\![\psi]\!]^M) = 1$, whenever $\psi \in \Sigma$), φ is true (i.e. $V_M([\![\varphi]\!]^M) = 1$).

What the hell is an *intended* model? In order to get an idea about this, let us first note that the above criterion would almost collapse if we dropped this restriction: it should be intuitively clear that, for any set Σ of sentences and any sentence φ not already in Σ, we could define some 'model' M^* (with all the ingredients from our above list) that makes Σ true but φ false. Just to take an example: The set $\{\varphi\wedge\psi\}$ would not 'imply' the formula φ of propositional logic because we could, e.g., interpret the operation $\wedge$ of conjunction by the truth-table usually attributed to disjunction ($\vee$); in such a model $\varphi\wedge\psi$ would be true if only ψ is true but φ might well be false at the same time. Of course, interpreting conjunction as disjunction is no longer logic but it can be done without violating the general definition of a model of propositional logic. In order to rule out such models, we thus have to restrict our attention to those M that intuitively do the right thing. And such M are called *intended models*. This is, of course, not a precise definition but merely a hint that every concrete application of Tarski's criterion of implication must first provide a *class* of intended models. (Note that the above examples from logic already did provide such a restricted notion of a model.) Why a class and not just a set? This has to do with the very foundations of set theory: in the case of propositional logic, a set would do (i.e. the class of intended models is a set), but in predicate logic we would like to have models based on any given (non-empty) *set* of individuals. So for any (non-empty) D the whole collection C of all intended models of predicate logic contains models containing D as one of their ingredients. C would thus have to be very large, too large indeed to be a set.

2.3 Logical constants and meaning postulates

Even if the class of all (possible) models of a language is generally too large to be of interest, one might attempt to find a general definition of intended models – or at least some conditions that the class of intended models of a language should meet. Several such conditions have actually been proposed in the logical literature (on what is called *abstract model theory*). The most straightforward one, presumably inspired by the notion of an intended model of predicate logic, is this:

Neutrality of Subject-Matter

Given two possible models M and M' that differ with respect to their ingredients but not with respect to their internal structures, then M is an intended model iff M' is and, moreover, the two should make the same sentences true.

To take a concrete example: If a model of predicate logic has the set {1,2,3} as its domain and assigns to the only (unary) predicate P the extension {3} and 1, 2, and 3 to the (only) constants a, b, and c respectively, then it makes the same formulae true as a model based on the domain {Alfred, Bertrand, Charles} that assigns {Alfred} to P and Charles, Bertrand, and Alfred to a, b, and c respectively. The example hopefully reveals where the principle got its name from.

Another, very general principle to be satisfied by any class of intended models might be called *closure under sub-topics*: whenever the meanings in a model allow the expressions to refer to or make statements about objects in a certain domain D, then there should be analogous models for each of D's subdomains. We will not state this principle here explicitly because it would appear even more vague and obscure than the previous one and in the attempt to make it more precise we would face a lot of technical complications. But these examples may suffice to give a flavour of the whole enterprise of finding general restrictions for intended classes of models. In particular, it appears that, whatever the value of such general principle may be, they do not seem to get us very far: more substantial and, presumably, less general restrictions will have to be imposed on the class of intended models of a given language.

In order to see what kinds of restrictions are likely to apply to intended models of a language, we will again look at the special but illuminating case of predicate logic. One reason why certain arbitrary models are unintended is that they do not provide the intuitively correct interpretations of certain expressions: any intended model should assign to a one-place predicate symbol (a constant function from variable-assignments to) a set of objects of a given domain (fixed throughout the model) and not, say, an object of that domain or a truth-value or a fifteen-place relation; similarly, in first-order logic a variable x must be assigned a function taking any variable-assignment g (over that domain) to g(x), rather than, say, a function taking g to the singleton {g(x)} or to the truth-table for conjunction. This 'minimal correctness' in what kind of objects the meanings of expressions of a given category are can be called *type matching*. A model of predicate logic thus has to assign to each (lexical) expression a meaning of the correct *type*. Types of meanings will be the object of part 3 and the general notion of type matching will again be discussed in part 4; so we will not discuss these fundamental restrictions on models at this point any further. But one should be aware of their existence.

Obviously, type matching is not enough. For as we have already seen, a (possible) model interpreting '$\wedge$' as disjunction is clearly unintended – even though conjunction and disjunction happen to be objects of the same kind, viz. truth-tables. (For ease of exposition, we now think of '$\wedge$' as a lexical expression of the category 'binary connective'; type matching restrictions would also apply if we introduced it by a syntactic operation.) So an additional restriction on the notion of an intended model of predicate logic would have to say that '$\wedge$' means (a constant function from variable-assignments to a *certain* truth-table. Similarly, we would have a restriction on the interpretation of '$\neg$' as negation (rather than the identical mapping on truth-values), '∀' as the universal (rather than the existential) quantifier, etc. On the other hand, no such restrictions apply to any individual variable or constant. The lexicon of predicate logic is thus divided between those expressions (like '∧', '¬', and '∀') whose interpretation is fixed throughout all intended models and the rest of them that behave in a more arbitrary way. Note that, strictly speaking, the meanings of logical constants are not the *same* in all intended models. Thus, e.g., what the universal quantifier refers to depends on the universe of discourse D. On the other hand, for any given D, the meaning of '∀' is literally the same throughout all intended models that have D as their universe. Moreover, the meaning of '∀' in one model *corresponds* to its meaning on any other model: it is always (the constant function from all assignments to) the singleton of the universe. This way of describing the meaning of '∀' also show that it is always a very special kind of object: it can be defined in purely structural terms ('set', universe'), without reference to any particular members of features of the universe of discourse. (This is,e.g. not true of the meaning of the quantified NP ***every bachelor***.) Similar things can be said about the other expressions whose meaning remains constant across all intended models of predicate logic. We will return to these observations towards the end of part 3.

What has been said about predicate logic also holds for model-theoretically interpreted languages in general: some expressions, the *logical constants* (or *logical words*), will have fixed meanings in all intended models. Does that mean that all other (lexical) expressions are only subject to type-matching restrictions? No. For one thing, the distinction between variables and constants is usually not taken as a type distinction. (This is, of course mainly a terminological point concerning the term *type*.) More importantly, in model-theoretic interpretations of (fragments of) natural languages, one needs many restrictions on the meanings of non-logical words, simply because without them certain logical implications could not be predicted. There are two different kinds of such non-logical restrictions, *subcategorial* and *relational* ones. A restriction of the first kind is, e.g., the requirement that proper names are to be interpreted as quantifiers of a very special kind: each of them must denote the set of those sets of individual that contain one particular individual (intuitively speaking, the bearer of that name) as a member. Note that this requirement does not fix the denotation of the proper name: there are always many quantifiers of that kind (= *subtype*). An example of a relational restriction on non-logical lexical items is the (over-simplified) *lexical decomposition* of ***kill*** into ***cause to die***: it establishes the relation of identity among the meaning of ***kill*** and a combination of the meanings of ***cause*** and ***die***.

Restrictions on the interpretation of non-logical words usually have an effect on the implications and other logical relations holding in virtue of what counts as an intended model. Thus, e.g., the fact that the sentence ***John sleeps*** logically implies ***Someone sleeps*** can be seen to be partly a consequence of the above-mentioned restriction on proper names (provided one chooses to interpret them as quantifiers at all, that is). Similarly, given a suitable analysis of gerunds and certain other constructions, the decomposition of ***kill*** may have the consequence that (*) will come out as valid:

(*) ***Killing someone is causing that person to die.***

In fact, it is to be expected that the models in which (*) holds are precisely those that satisfy the decomposition restriction. So instead of putting a restriction on the interpretation of the lexical items ***kill***, ***cause***, and ***die***, we might as well say that in any intended model the truth-value of (*) must be 1. Such sentences whose truth is, by definition, a necessary condition on intended models are called (*direct*) *meaning postulates*. The use of meaning postulates is thus a technique of imposing restrictions on the non-logical part of the lexicon. However, not all such restrictions can be expressed by (direct) postulates: under certain background assumptions about the class of intended models, the restriction that proper names be interpreted as quantifiers of a certain kind cannot be expressed by a direct meaning postulate. However, we will see (in part 4) that we will be able to express this restriction by an *indirect meaning postulate*, i.e. one to be expressed in a (logical) language different from (and more powerful than) the one to be analyzed.

2.4 A word about alternative approaches

It must be emphasized that the model-theoretic approach to meaning, although standard, is by no means the only one to have ever been suggested. In this section, some of its most popular alternatives will be mentioned. Some of these alternatives can be motivated by Mates’s trap and similar alleged shortcomings of model-theoretic semantics. A radically different approach to meaning would seek to avoid these problems by stipulating that meanings themselves are expressions of some language (of thought). Any such semantic theory can be called *representational*. In one of its more naïve forms representationalism may favour the trees of componential analysis mentioned in part 1. Note that at least some meaning relations can be defined on such tree representations without further interpreting them (model-theoretically or otherwise). In particular, hyponymy might be reduced to the subset (or sublist) relation. To be sure, more sophisticated notations will be needed to account for the semantic structure of any significant portions of natural languages but the semantic features give at least an idea: a representationalist analysis semantically analyses natural language expressions (or utterances) by relating them to representations, i.e. expressions of some other language; and all meaning relations are directly defined on these representations without any model-theoretic detour. One might wonder what these representations are: are they supposed to be or somehow represent neurological structures or processes, are they universal innate ideas or combinations thereof, or are they merely nice little formulas whose very existence is justified in their success (if any) in descriptive semantics? The answer to this question depends on the particular representationalist approach. So does any criticism from a model-theoretic point of view.

If meanings are more subtle (or 'fine-grained') than whatever logical semanticists take them to be, this does not mean that they have to be expressions of another language. Maybe they are objects in their own right, irreducible to the stuff model-theoretic semantics is made of and following their own rules. In that case it would be surprising if the logical approach to meaning would be of any help in developing a theory of these semantic objects. But how could one arrive at such a theory? One approach would be by listing as many basic general facts about meanings as one can find and as one needs for natural language semantics and then try to get everything into a neat (usually axiomatic) theory. Such is an *intensionalist* approach to meaning (or my caricature of it). Quite surprisingly, the outcome very often not only closely resembles the results of model-theoretic semantics, the methods used in establishing them are even essentially the same. But there are differences: intensionalists usually have no problems with the subtleties of meaning that lead more conventional semanticists into Mates's and other peoples' traps. Moreover, intensionalist accounts of even very simple semantic phenomena tend to be very complex. Finally, whether any intensionalist theory will arrive at the same descriptive breadth and compatibility with linguistics (and philosophy of language) remains to be seen.

Even if we accept the fundamental connection between meaning and truth and agree that semantics should be based on logic, we might still look for alternative approaches to logic itself. It is well-known that, in the case of first-order logic, logical implication can be defined without any reference to truth, reference, or models: a first-order implication holds if and only if it can be established by the use of some suitable proof-procedure. (This is the content of Kurt Gödel's famous *Completeness Theorem*.) The essential point about these proof-procedures is that they are purely formal (or 'syntactic', as logicians say) in the sense that they are manipulations of formulas that only depend on their structure. So couldn't natural language semantics be based on proofs rather than models? This has, in fact, been suggested by various logicians but again, a concrete application to any significant fragment of some language has yet to be elaborated. But the program of *proof-theoretic semantics* has been around for a while. Three things about it should be noted. The first is its representationalist flavour: like the representationalist definitions of semantic relations, proof-theoretic techniques are purely syntactic. Secondly, the Completeness Theorem holds for first-order logic but nowhere beyond. (This can be seen as a consequence of the *Undefinability of Arithmetical Truth* established by Tarski.) Now there is some indication to the effect that natural language should be analyzed as being of higher order. Although this would by no means imply that proof-theoretic techniques are unavailable, their scope would be more limited than in the first-order case. Thirdly, something positive must be said about the proof-theoretic approach: even if, in its orthodox form, it is completely equivalent to the model-theoretic approach (to first-order logic), it generalizes in quite different ways. In particular, some notorious difficulties with model-theoretic analyses of natural language sentences can be overcome by switching to proof-theory and slightly adjusting some techniques; this is, at least, what some logicians have suggested.

There are less radical alternatives to model-theoretic semantics in the sense discussed above. One of them, *realism,* should be mentioned explicitly because it is easily confused with what we have done in section 2.3. However, we will only be able to discuss it in part 5.

Exercise

3. Show that, at least for classical propositional logic, the definition of negation given on p. 10f. is correct: any formula φ negates a formula ψ if and only if φ is *logically equivalent to* $\neg\psi$, i.e. if $[\![\varphi]\!]^g = [\![\neg\psi]\!]^g$ for any truth-value assignment g. You may assume that self-contradictions always get the truth-value 0 and that the valid formulae are the tautologies. Hint: One direction is simple. The other one is to show that $[\![\varphi]\!]^g = [\![\neg\psi]\!]^g$ for any g; it is best to distinguish the cases $[\![\varphi]\!]^g = 1$ and $[\![\varphi]\!]^g = 0$ and use one of the two properties of negation in each case.

3. Extensional Types

3.1 Motivation

Although we have seen that compositionality forces us to use highly complicated objects as semantic values (or meanings), we will try to avoid them as often as we can. In particular, they will not play any role in the classification of meanings according to their types. Instead, we will concentrate on the *extensions* of expressions. What are extensions? There are various non-equivalent ways of defining this notion. A good characterization (for our purposes) is this: the extension of an expression is that object in the world that the expression corresponds to. In the case of a referential NP, the extension is thus the referent; the extension of a VP is the set of individuals having the property (engaging in the activity, …) described by it; the referent of a sentence is its truth-value, etc. (The latter option can be justified by thinking of sentences as 0-place predicates denoting one of the two sets of 0-tuples, Ø and {Ø}, i.e. 0 and 1.) If we want to apply it to the expressions of predicate logic, extensions are thus whatever we get when we apply meanings to a particular, given variable assignment; extensions are thus model-dependent. Our classification of extensions will then give rise to a classification of meanings according to which types of extensions will be their values in any given models. Similarly, we will think of natural-language expressions as having extensions that are determined by their meanings and whose classification carries over to these meanings.

We have already seen that different kinds of expressions of predicate logic differ with respect to the kinds of extensions that they have: some of them, variables and individual constants, refer to individuals (from a given universe of discourse), others (predicates) correspond to relations of various arities, formulas (or sentences) ‘denote’ truth-values, connectives) stand for truth-tables, etc. Although these differences in extensions do not reflect all semantic distinctions to be found among the expressions of predicate logic, they will be sufficiently rich and interesting to provide a basis for our classification. But what is the use of such a classification? One, very basic application has already been announced in the previous part: it will allow us to define a notion of *type matching* that is useful in the definition of a (class of) intended models. A more substantial applications of the classification of extensions into types is the theory of meaning combinations (or semantic operations), i.e. the general structure of semantic algebras in the sense of section 1.3.

3.2 Types and Ontologies

The type of object that may serve as the extension of a predicate logic expression obviously depends on its syntactic category (variable, individual constant, formula, etc.). In other words, to each syntactic category there corresponds a kind of extension. the same seems be true of natural language expressions, or so we will assume for the time being. Here is one plausible correspondence covering the most obvious cases:

(1)

Category	*Kinds of extensions*
S	truth-values
NP_{ref}	individuals
VP	sets of individuals
N	sets of individuals
Det	binary relations among sets of individuals
NP_{quant}	sets of sets of individuals
V_{trans}	binary relations among individuals

Since sets and binary relations closely correspond to characteristic functions, all extensions appearing below the second line of the above table can be thought of as functions with domains that can be built up from truth-values and individuals, i.e. the extensions of sentences and referential NPs. Using t as short-hand for 'truth-values', e for 'individuals' (or 'entities'), and '(ab)' for 'function from a to b', the above table can be rewritten in the following way:

(2)

Category	*Kinds of extensions*
S	t
NP_{ref}	e
VP	(et)
N	(et)
Det	$((et)((et)t))$
NP_{quant}	$((et)t)$
V_{trans}	$(e(et))$

Tables like (2) (or, rather, their completions) are called *type assignments* or *type correspondences*: they assign to each syntactic category the corresponding type of meanings (or extensions) of the expressions of that category. A straightforward way of specifying conditions of type-matching, then, is by stipulating that all intended models are subject to a given type assignment. Given the compositional approach to semantics, it would, of course, suffice to postulate that the lexical meaning assignment satisfies the type assignment and that the semantic operations G are in accord with it: whenever F is a syntactic operation combining (structures of) expressions $X_1,\ldots,X_n$ of categories $\kappa_1,\ldots,\kappa_n$ into $F\,(X_1,\ldots,X_n)$ of category κ_{n+1} and $\kappa_1,\ldots,\kappa_n,\kappa_{n+1}$ are assigned the types $a_1,\ldots,a_n,a_{n+1}$, then $G_M\,([\![X_1]\!]^M,\ldots,[\![X_n]\!]^M)$ is of type a_{n+1}. (G_M is, of course the semantic operation corresponding to F, according to the model M.) Note that while to every syntactic category there corresponds a type, there may be types corresponding to no or more than one category: sets of individuals, i.e. objects of type (et), are extensions of both nouns and VPs and, at least according to the assignment (2), there is no category whose expressions denote functions from truth-values to individuals, i.e. objects of type (te).

Given these preliminaries, we can now define the set T of *types of extensions* or *extensional types* as containing (exactly) everything that can be obtained by starting from the *basic types* e and t and taking pairs; T is generated by the context-free grammar $\{S \to e, S \to t, S \to (S + S)\}$, where '$e$', '$t$', ')', and '(' are terminal symbols and 'S' is the non-terminal start-symbol. Note that the types themselves are only symbols, labels, names. Which objects they correspond to varies with the universe of discourse D. But the set D_a of *possible extensions of type a* (*given* a non-empty universe D) can again be defined recursively:

(3) $D_e = D$;
$D_t = \{0, 1\}$;
$D_{(ab)} = D_b{}^{D_a}$, i.e. the set of those functions f that are defined for any $u \in D_a$ and assign to it a value $f(u) \in D_b$.

A system or family $(D_a)_{a\in T}$ of such typed domains will be called an *ontology* and the individual D_a are the *ontological layers*. Ontologies are thus uniquely determined by their universe of discourse (= layer of individuals of type e). Apart from truth-values and individuals, which layers does an ontology satisfying (3) consist of? In order to answer this question, it is best to concentrate on such *normal* ontologies whose domain D of individuals contains neither functions nor truth-values; then any of its objects is of exactly one type, i.e. it occurs in exactly one layer. As already mentioned, for any type a, $D_{(at)}$ contains all characteristic functions χ^a_A of subsets A of D_a. For most purposes, these χ^a_A can be identified with the corresponding sets A. However, sometimes one must take a little care. A case in point is the empty set Ø whose characteristic function can be found in every layer of the form $D_{(at)}$ – simply because Ø is a subset of every D_a. But if $a \neq b$, then $D_a \neq D_b$ and hence no χ^a_A is identical with any χ^b_B: their domains do not coincide. In particular, then, $\chi^a_Ø \neq \chi^b_Ø$, even though there is only one empty set.

Under the normality condition, Ø is the only set that gets characterized by more than one function in one ontology. Keeping this exception in mind, it is therefore not too dangerous to actually think of layers $D_{(at)}$ as power sets. Other identifications are equally harmless. In particular, to every function f of a type $(b(at))$ (i.e. $f \in D_{b(at)}$) there corresponds exactly one binary relation $R \subseteq D_a \times D_b$: u$R$v – i.e. (u,v) $\in R$ – iff f(v)(u) = 1, i.e. if the result of applying the function f to v characterizes a set of individuals among which u can be found. f(v) thus characterizes the set of v's R-predecessors. There is obviously something arbitrary in this way of identifying binary relations with functions: why should we get the pair (u,v) by first applying to u, and then to v? Why not the other way round? The reason for this particular choice lies in a nice consequence of it: we can continue to think of (at least certain) transitive verbs as denoting binary relations and then analyze sentences like ***John loves Mary*** by subsequently applying the extension ($l \in D_{e(et)}$) of the verb to that of the object (m $\in D_e$) and the subject (j $\in D_e$): l(m)(j). Unlike the first-order formalization, this semantic analysis reflects (part of) the constituent structure of the sentence. – The correspondence between $D_{b(at)}$ and $D_a \times D_b$ can obviously be generalized to n-place relations and we will later make use of this possibility without further warning. But it should be kept in mind that all such identifications must be taken with care.

3.3 Coverage and Size

Since there are infinitely many types but only finitely many syntactic categories (in a given language), it is clear that not every type can correspond to a category (of that language). In particular, then, not every object in a given ontology can be the extension of an expression of a given language). If, on the other hand, some type a does correspond to a syntactic category C, one might expect C to *cover* a in the sense that, in any ontology, every object u of type a (i.e. u $\in D_a$) is a possible extension of at least some expression of category C. But is this really so? Let us check the type assignment (2)! In the case of sentences, coverage of t is obvious: some (English) sentences are true, others are false. The case of referential noun phrases is also quite straightforward: as we have seen in section 2.3, it should not really matter which individual is denoted an individual constant in predicate logic; and, at least if we assume that proper names are analyzed in an analogous fashion, a name like ***John*** should be able to denote any individual whatsoever – depending on the particular model. As to VP and N, it also seems plausible to assume that any set of individuals of any ontology can, in principle serve as the extension of a lexical item like ***sleep*** or ***woman***. Similarly, it appears reasonable to assume that there should be no restriction on which binary relation Rx could be denoted by the lexical verb ***kill***; even the restriction that R must at the same time be the extension of ***cause to die*** does not bear on the issue who actually kills whom.

In the remaining two cases, the situation is different. Many determiners (like ***every***, ***no***, ***some***, etc.) are logical words and their meanings is thus fixed once we are given a set of individuals, Moreover, it should be clear that, in any ontology, there exist lots of objects of type $(et)((et)t)$ that are not denoted by any of these logical determiners. But even the non-logical determiners (like ***many*** or ***John*** maybe) satisfy certain restrictions that prevent them from denoting arbitrary relations among sets of individuals. The most obvious such restriction (but possibly not the only one) is their *conservativity* that holds because each δ of them satisfies:

$$\text{(CONS)} \quad \delta(A)(B) = \delta(A)(B \cap A)$$

(Note that in (CONS), $\cap$ combines the characteristic functions of two sets into the characteristic function of their intersection.) The conservativity constraint reflects the basic observation that any sentence of the form ⌜Det N VP⌝ has the same truth conditions as the corresponding sentence ⌜Det N ***is a*** N ***and*** VP⌝: ***no woman is asleep*** means ***no woman is a woman and is asleep***, etc. Now, clearly, in any ontology one can find relations δ of type $(et)((et)t)$ that do not satisfy (CONS). Thus, if every determiner satisfies (CONS), Det does not cover $(et)((et)t)$.

How about $\mathrm{NP}_{\mathrm{quant}}$, then? The question of whether every set of sets may serve as a possible extension is rather difficult to answer and so we will be happy with one simple observation: unlike $\mathrm{NP}_{\mathrm{ref}}$, VP, N, and TV, the category $\mathrm{NP}_{\mathrm{quant}}$ does not contain any such *flexible lexical items* that can denote any object whatsoever of the corresponding type: the meaning of ***everybody*** is bound to be a restricted universal quantifier and, e.g., cannot turn out to be $\{\emptyset\}$. And even if we take all lexical $\mathrm{NP}_{\mathrm{quant}}$s together, their extensions will only cover a small percentage of all objects of type $(et)t$. Moreover, the inclusion of 'lifted' proper names, i.e. quantifiers of the form $\{S \subseteq D \mid u \in S\}$ (for some fixed $u \in D_e$) would not give us full *lexical coverage* either: all quantifiers of the form $\{S \subseteq D \mid u \notin S\}$ would still be left out of the spectrum.

One may wonder whether there is any way of predicting whether a given syntactic category covers or even lexically covers its corresponding type. One attempt at finding such a criterion is:

Keenan's Thesis

If a category C lexically covers a type a, then a must be small.

Intuitively speaking, a small type a is one with relatively few elements in D_a. In order to make this notion more precise, one would first have to compare different ontological layers with respect to their sizes, i.e. cardinalities. In order to do this, two things should be noted. The first is that the size of some layers does not depend on the particular ontology: whatever the domain D of individuals is, D_{tt} will always contain exactly two elements, the size of D_t will be 4, etc. In fact, if a is any type containing only ts, then D_a does not depend on D_e and consequently its cardinality is fixed (and finite). The second observation to be made is that, no matter what a and b are, the cardinality of $D_a{}^{D_b}$ is $|D_a|^{|D_b|}$ where $|D_a|$ and $|D_b|$ are D_a's and D_b's respective cardinalities. If both $|D_a|$ and $|D_b|$ are natural numbers, this can be proved by induction on $|D_b|$. (The essential point is that each function from a domain with $|D_b|$ members can be extended to $|D_a|$ new functions on a domain with $|D_b|$ +1 members.) If one of the two is infinite, '$|D_a|^{|D_b|}$' denotes the cardinality of $D_a{}^{D_b}$ (almost) by definition. In particular, then, the power set $D_{(et)}$ of the layer D_e contains exactly $2^{|D_e|}$ members, whereas its power set, $D_{e(et)}$, contains 2^{2^n}, where $n = |D_e|$ is the number of individuals. Now the former is the cardinality of the type lexically covered by VP, whereas the latter is the number of generalized quantifiers, a type that might is not lexically covered. So, if the above thesis is right, the borderline between small and large types must lie between the two. In order to determine it, one must keep in mind, that a cardinality such as 2^n still depends on the number n of individuals. So what we are comparing are functions from such n to natural numbers. The general pattern for determining these functions is:

$$\#_n(t) = 2;$$
$$\#_n(e) = n;$$
$$\#_n((ab)) = \#_n(b)^{\#_n(a)}.$$

We may call the function taking any n to $\#_n(a)$ the *size of* type a. So when is a small, in the sense of Keenan's Thesis? One answer that draws the line where we want it to be is this: its size must not exceed the $2^{\phi(n)}$, where Φ is a fixed polynomial, i.e. an algebraic term construed with +, ×, fixed numerals, and n. Let us check this for the type $e(et)$ corresponding to TV:

$$\#_n(e(et)) = \#_n(et)^{\#_n(e)} = \#_n(et)^n = [\ \#_n(t)^{\#_n(e)}]^n = [2^n]^n = 2^{n\times n},$$

so the polynomial $\Phi(n)$ is: $n\times n$. In order to show that the types corresponding to Det and $\mathrm{NP}_{\mathrm{quant}}$ are not small, one would have to prove that their sizes cannot be represented in the form '$2^{\phi(n)}$', which can indeed be done by purely mathematical methods outside the scope of this course.

Even if Keenan's Thesis (or something like it) were correct, wouldn't it just be an odd observation about natural language? Or, worse, wouldn't it just be an observation about this type-theoretic framework? We will not answer these questions here but merely indicate that Keenan himself has worded his thesis more carefully and at least tried to explain it (or make it plausible) by considerations on the learnability of lexical items and their meanings. Whatever the merits of these attempts, it would presumably be more satisfactory to somehow *reduce* Keenan's Thesis (if correct and meaningful) to some more basic principles about the ranges of denotations and the logical behaviour of lexical items. This has not (yet) been done.

3.4 Combinations of Types

One of the major advantages of a type-theoretic framework lies in the fact that it naturally leads to a theory (or a variety of theories) of *possible semantic operations*, i.e. combinations of meanings corresponding to structure-building operators. We start with a simple observation. In many cases, syntactic combinations semantically amount to functional application, i.e. applying the extension of one expression to the extension(s) of the other(s) that are combined with it. Thus, as was already mentioned above, the truth-value of ***John loves Mary*** can be computed by applying the extension of ***loves Mary*** to that of ***John***; and the former is obtained by applying $\llbracket\textit{\textbf{loves}}\rrbracket$ to $\llbracket\textit{\textbf{Mary}}\rrbracket$:

$$\begin{aligned} &\llbracket(_{\mathrm{S}}\ \textit{\textbf{John}}\ (_{\mathrm{VP}}\ \textit{\textbf{loves Mary}}))\rrbracket\\ =\ &\llbracket\textit{\textbf{loves Mary}}\rrbracket(\llbracket\textit{\textbf{John}}\rrbracket)\\ =\ &\llbracket\textit{\textbf{loves}}\rrbracket\ (\llbracket\textit{\textbf{Mary}}\rrbracket)\ (\llbracket\textit{\textbf{John}}\rrbracket),\end{aligned}$$

which is 1 iff the pair $(\llbracket\textit{\textbf{John}}\rrbracket,\llbracket\textit{\textbf{Mary}}\rrbracket)$ is in the relation described by the $e(et)$)-function $\llbracket\textit{\textbf{loves}}\rrbracket$. Similarly, combining a quantified NP with a VP can be interpreted by applying the denotation of the $\mathrm{NP}_{\mathrm{quant}}$ to $\llbracket\mathrm{VP}\rrbracket$:

$$\begin{aligned} &\llbracket\textit{\textbf{everyone sleeps}}\rrbracket\\ =\ &\llbracket\textit{\textbf{everyone}}\rrbracket\ (\llbracket\textit{\textbf{sleeps}}\rrbracket),\end{aligned}$$

which is 1 iff $\llbracket\textit{\textbf{sleeps}}\rrbracket \subseteq \llbracket\textit{\textbf{person}}\rrbracket$. So to each of the structure-building principles:

(4) $\mathrm{S} \rightarrow \mathrm{NP}_{\mathrm{ref}}\ \mathrm{VP}$
(5) $\mathrm{VP} \rightarrow \mathrm{TV}\ \mathrm{NP}_{\mathrm{ref}}$
(6) $\mathrm{S} \rightarrow \mathrm{NP}_{\mathrm{quant}}\ \mathrm{VP}$

there corresponds the semantic combination of functional application:

(4') $\llbracket\mathrm{S}\rrbracket = \llbracket\mathrm{VP}\rrbracket(\llbracket\mathrm{NP}_{\mathrm{ref}}\rrbracket)$
(5') $\llbracket\mathrm{VP}\rrbracket = \llbracket\mathrm{TV}\rrbracket(\llbracket\mathrm{NP}_{\mathrm{ref}}\rrbracket)$
(6') $\llbracket\mathrm{S}\rrbracket = \llbracket\mathrm{NP}_{\mathrm{quant}}\rrbracket(\llbracket\mathrm{VP}\rrbracket)$

Remember that, in the framework discussed in section 1.5, the context-free rules (4) - (6) can be thought of as binary operations on trees. The order in which these operations take their arguments does not necessarily coincide with the order in which they arrange them. So in all three cases the first argument might as well correspond to the category corresponding to the function type *ab*. The semantic operation would then always be the very same one, viz. application of the first argument to the second.

Let us now consider quantified direct objects:

(7) $\mathrm{VP} \rightarrow \mathrm{TV}\ \mathrm{NP}_{\mathrm{quant}}$

An interesting effect of our type-theoretic framework is that it tells us that, whatever the semantic operation corresponding to (7') may be, it cannot be functional application (*FA*), as in the first cases. Why not? Because *FA* only applies if one of the extensions is of a type *ab* and the other is of the argument type *a*:

(8) $(ab), a \underset{FA}{\overrightarrow{\quad}} b$

(8) says that *FA* combines objects of type *ab* with objects of type *a* into objects of type *b*, but according to our type assignment (2), the types in (7) are *e*(*et*) and (*et*)*t*, and they don't match the pattern (8). But maybe there is another, equally straightforward way of combining them:

(9) $(e(et)), (et)t \underset{XY}{\overrightarrow{\quad}} et$

To be sure, there is no problem in defining the particular combination *XY* needed in our special case: given the extension $R \in D_{e(et)}$ of a transitive verb and the extension $\mathrm{Q} \in D_{(et)t}$ of a quantified NP, the result of combining the two expressions by (7) should denote the set of all individuals that bear R to a quantity in the denotation of Q:

(10) $XY(R,\mathrm{Q}) = \{\mathrm{x}{\in}D_e \mid \{\mathrm{y}{\in}D_e \mid \mathrm{x}R\mathrm{y}\} \in \mathrm{Q}\}$

(Note that we have identified functions with the sets and relations they characterize.) Applying (10) to examples like ***sell everything*** shows that it does make the correct predictions: $\{\mathrm{x}{\in}D_e \mid \{\mathrm{y}{\in}D_e \mid x[\![\textbf{\textit{sell}}]\!]y\} \in \{D_e\}\}$ $= \{\mathrm{x}{\in}D_e \mid \{\mathrm{y}{\in}D_e \mid x[\![\textbf{\textit{sell}}]\!]y\} = D_e\} = \{\mathrm{x}{\in}D_e \mid (\forall \mathrm{y}{\in}D_e)x[\![\textbf{\textit{sell}}]\!]y\}$. But whereas *FA* seemed to be the natural thing to do given the input and output types of (7), *XY* looks much more idiosyncratic. Or is the difference between *FA* and *XY* only an illusion caused by the complicated nature of the types in (9)?

The type-theoretic approach to delimiting the kinds of semantic operations to be met in natural languages is to associate with any combination of types the most natural combination(s) of meanings (extensions) of those types. One would thus try to *predict* that, under the type assignment (2), rules (4) - (6) should be interpreted by functional application, whereas (7) has *XY* as at least one natural interpretation. Using the arrow notation introduced in (8), we can characterize this task as that of supplying naked *type combinations* of the form (11) with suitable operations *G*, as noted in (12):

(11) $a_1, \ldots, a_n \rightarrow a_{n+1}$

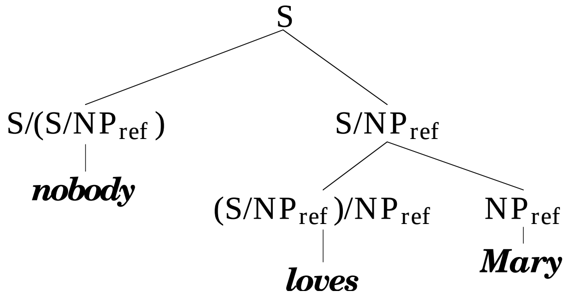
(12) $a_1, \ldots, a_n \xrightarrow[G]{} a_{n+1}$

There are two natural starting points for attacking this problem, and we will discuss both of them. The first one will be introduced in this section and continued in part 4: it consists in finding independent evidence (and methods) for narrowing down the possible naked type combinations (11) and only later worry about how to obtain the *G*. The second possibility is to look for plausible general criteria that semantic operations *G* should meet and only then wonder which type combinations they correspond to. One such method will be discussed in section 3.5, another one will naturally come up in part 4.

A straightforward formal procedure for defining all plausible (naked) type combinations is to generate them from some axiomatic system. How could such a system be construed? An answer to this question comes from formal syntax theory. For there is a close analogy between types and the categories of *(classical) categorial grammar*: expressions denoting functions of type *ab* may be thought of as *incomplete* (or, in Frege's terms: *unsaturated*) terms denoting *b*-type objects: they lack an *a*-type argument-term. So according to classical categorial analysis (due to Ajdukiewicz), they would be classified as being of a *functor category* B/A, which is read 'B over A'. (Note the unfortunate inversion). Generalizing this idea then gives us a one-one correspondence between our functional types and the categories to be construed from

$\mathrm{NP_{ref}}$ and S by slashing. Thus, e.g., the types $(et)t$ and $e(et)$ correspond to the categories $\mathrm{S/(S/NP_{ref})}$ and $\mathrm{NP_{ref}/(NP_{ref}/S)}$, respectively. From a categorial point of view, we might thus be tempted to read '$\mathrm{NP_{quant}}$' and 'TV' as abbreviations of the latter functor categories. An obvious result of such an analysis would be the prediction that ***Nobody loves Mary*** is a sentence:

(13)



(13) is a well-formed *categorial tree* because it is based on the principle *C* of classical categorial grammar that combines expressions of functor categories A/B with their arguments of category B, resulting in complex expressions of category A. (Note that *C* can be expressed by a scheme of a context-free rules 'A → A/B B'.) Clearly, *C* closely corresponds to *FA* in that the latter is applicable in just the same situations (given our correspondence between types and the categories of categorial grammar). And, as one can easily check, *FA* is actually what we need in order to compute the extension of (13).

In the attempt of applying categorial techniques to natural language syntax, it has been observed that the principle *C* is insufficient. For, as we have just seen in terms of combining types by means of *FA*, we would not be able to derive sentences with quantified direct objects with the kind of lexicon underlying (13). Of course, we may enrich our lexicon by a double classification of, say, ***loves***: $\mathrm{VP/NP_{quant}}$ → ***loves*** [i.e.: $\mathrm{(S/NP_{ref})/(S/(S/NP_{ref})}$ → ***loves***] would do for our present purposes. But then we would get the same kind of lexical ambiguity all over the place; and it certainly is not the only one to be encountered. So instead of admitting ever more homonyms, one should try to find a more systematic account of what is going on. And one way of arriving at such an account is to replace the classical principle of combination *C* by a more liberal system of possible combinations. The most celebrated of these enriched systems of categorial grammar is *Lambek's Calculus* to which we will now briefly turn.

3.5 Lambek's Calculus

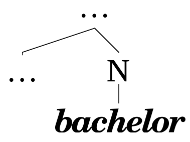
It must be kept in mind that our interest in Lambek's Calculus is not to repair classical categorial analysis but rather to find a theory of possible type combinations in the sense of (11). So we might as well use the type notation directly and immediately forget about the slash categories. The calculus then provides a system of *deriving* certain type combinations from a collection of basic ones, the *axioms*. The latter are easily introduced and motivated:

(L0) $a \rightarrow a$

(L1) $(ab), a \rightarrow b$; $a, (ab) \rightarrow b$

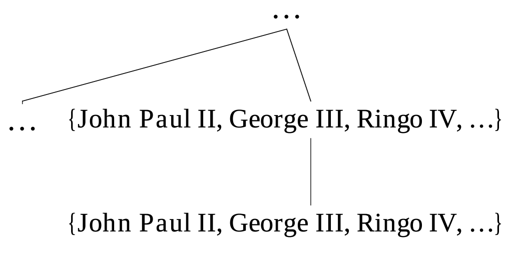
The first axiom is absolutely trivial: as a principle of type combination it says that a (unary) semantic operation may map extensions of a given type to extensions of the same type. A typical case of (L0) is the interpretation of a lexical insertion rule: the meaning of the lexical item is passed on to the node immediately dominating it: the extension of a tree like

(14)



can be determined by:

(14')


…
…
{John Paul II, George III, Ringo IV, …}
{John Paul II, George III, Ringo IV, …}

Note that the semantic operation corresponding to the lexical rule 'N → ***bachelor***' is the identical mapping *ID*. So we already know how to complete (L1) by a suitable semantic operation:

(L0') $a \underset{ID}{\rightarrow} a$

However, at this point it is not clear whether *ID* should always be considered the only possible semantic operation taking us from a to a. And, as was already mentioned, the problem of associating semantic operations with type combinations will not be attacked (systematically) before part 4.

There is not much to be said about the axiom schemata (L1): they express that whatever functional application does is a possible type combination. The reason why we have two versions of this scheme is that we do not want to be biased as to the order in which functor and argument are supplied by the syntactic construction. Of course, when read as a categorial axiom scheme, the fact that (L1) goes both ways is less trivial. Indeed, in more syntactically oriented versions of Lambek's Calculus one will find more sophisticated formulations of (L1). In any case, the scheme will always be a version of the fundamental law C of classical categorial grammar.

Note that, on our type reading of Lambek's Calculus, it is again unclear whether the completion (8) of (L1) by functional application should always be the only way of providing a corresponding semantic operation; but it is certainly always a straightforward possibility.

(L0) and (L1) are the only axioms. All other combinations will be derived from them by means of three schematic *inference rules*. Before we go into them, we must say something about their format. One might expect the rules allow us to deduce the possibility of some type combination $a_1, \ldots, a_n \rightarrow a_{n+1}$ from that of some other, previously established type combination(s). However, the actual rules are somewhat more general. To motivate their format, let us look at a very simple example. We certainly would like the following to be a possible type combination:

(15) $e, e, e(et) \rightarrow t$

(15) states that the extension of a transitive verb can be combined with two NP$_{\text{ref}}$-extensions so that the result will be a truth-value. Although there might be no immediate use for this kind of combination, we certainly do not want to rule it out as an impossible combination of types. Now note that in our above discussion there was no need for (15) because we got the same result by iterating FA. But this fact in itself might be seen as a justification of (15): the type combination should be possible, *because* we could get it by iteration of other, possible type combinations. In order to turn such a justification into a formal proof (from the axioms

of our Calculus, that is), we could argue along the following lines: by (L1) we know that we can get the type *et* of the VP out of the TV's type *e*(*et*) and the referential object's type *e*:

(i) $e(et), e \rightarrow et$

This combination should also be applicable in the situation (15), where there is still another *e* around (corresponding to the referential subject). From the possibility of (i) we should thus be able to infer the possibility of:

(ii) $e, e(et), e \rightarrow e, et$

Note that the type *e* of the subject still appears on the right hand-side: we do not want to combine the meanings of transitive verb, subject, and object into that of the VP. But what we get as the result from (ii) may now be fed into the scheme (L1) again:

(iii) $e, et \rightarrow t$

So, the argument goes on, we can plug (ii) and (iii) together and obtain (15).

An important point about this derivation of (15) is the intermediate step (ii) which is not of the general form (11) of a (naked) type combination but rather of the more general form. i.e. it contains more than one resulting type:

(16) $a_1, \ldots, a_n \rightarrow b_1, \ldots, b_m$

The above example hopefully illustrates that we can understand these more general combinations as steps in a derivation. In the syntactic interpretation of Lambek's Calculus these intermediate steps could be made more sense of: (16) can be read as stating that a string of expressions of categories $a_1,\ldots,a_n$ can always be analyzed as consisting of expressions of categories $b_1,\ldots,b_m$.

A closer look at the derivation of (15) reveals that it involved two general principles, corresponding to the first two inference rules of Lambek's Calculus. In order to get from (i) to (ii) we argued that a type possible combination should remain possible in the presence of other types. We thus have the following principle of 'contextual freedom' of type combinations:

(L2) If

$$a_1, \ldots, a_n \to b_1, \ldots, b_m$$

is a possible type combination and $c_1,\ldots,c_k$, $d_1,\ldots,d_l$ are types, then

$$c_1, \ldots, c_k, a_1, \ldots, a_n, d_1, \ldots, d_l \to c_1, \ldots, c_k, b_1, \ldots, b_m, d_1, \ldots, d_l$$

is a possible type combination.

(We tacitly adopt the convention that something like $c_1,\ldots,c_k$ might be an empty string of types.) If we denote arbitrary strings of types by capital letters, (L2) can be expressed in a shorter form that we will adopt:

(L2) If A $\to$ B, then: C, A, D $\to$ C, B, D .

The second principle of argumentation that we have used above was in plugging (ii) and (iii) together to obtain (15). We have thus made implicit reference to a 'transitivity principle' for type combinations:

(L3) If A $\to$ B and B $\to$ C then: A $\to$ C.

In order to motivate the final inference rule, we have to look at another example, the interpretation of the word ***not*** in the quantified NP ***not every student***. As one can easily show, a plausible and straightforward interpretation of this negation of NP$_{\text{quant}}$ is by taking it to denote the complement of its argument: ⟦***not every student***⟧ = $D_{(et)t}$ \ ⟦***every student***⟧, which in our type-theoretic framework amounts to:

⟦***not every student***⟧(P) = 1 iff ⟦***every student***⟧(P) = 0.

Or, equivalently:

⟦***not every student***⟧(P) = ¬(⟦***every student***⟧(P))

where ¬ is the (metalinguistic way of referring to) ordinary truth-functional connective of negation. Given this connection, one might wonder whether there is no direct way of combining $\neg \in D_{tt}$ with quantifiers. We would thus need the type combination:

(17) $tt, (et)t \rightarrow (et)t$

Here is how we could argue for (17): given the axioms and rules already introduced, it is easily seen that (18) is a possible type combination:

(18) $tt, (et)t, et \rightarrow t$

Now note the categorial reading of (18): a predicate (category S/NP_{ref}), a quantified NP ($S/(S/NP_{ref})$) and negation (S/S) combine into a sentence. So if we omit the predicate we get a sentence lacking a predicate, i.e. something of category $S/(S/NP_{ref})$, the category of quantified NPs corresponding to type $e(et)$. But this is precisely what (17) says. The general principle underlying this derivation of (18) from (17), then, is:

(L4) If A, $b \rightarrow c$, then: A $\rightarrow bc$, whenever A is non-empty;
similarly, if b, A $\rightarrow c$, then: A $\rightarrow bc$, whenever A is non-empty.

One might conjecture that (L4) can also be used to explain the use of ***or*** in, e.g., ***reads or writes*** as an instance of the type combination:

(19) $t(tt), e(et), e(et) \rightarrow e(et)$

However, it turns out that (19) is not derivable in Lambek's Calculus. We will return to this problem in part 4.

(L0) - (L4) constitute Lambek's Calculus. We will now check that it does give us the type combination needed for the operation XY that combines transitive verbs and their quantified objects into VPs:

(20) $(e(et)), (et)t \rightarrow et$

In order to derive (20), we first note that we can get it from (21) by (L4):

(21) $e, (e(et)), (et)t \rightarrow t$

So the task is to derive (21), which is not hard: the name type e combines with $e(et)$ into et, by (L1), and this combination is also possible in the presence of the quantifier $(et)t$ to the right, by (L2), giving us $et, (et)t$, which in turn can be reduced to t, by (L1). Plugging the last two results together by means of (L3) the gives us the desired combination (21). The whole derivation of (20) can now be represented in a (hopefully self-explanatory) *proof tree* form:

(22)

$$\dfrac{\dfrac{\dfrac{\text{L1: } e, e(et) \to et}{\text{L2: } e, e(et), (et)t \to et, (et)t} \;;\quad \text{L1: } et, (et)t \to t}{\text{L3: } e, e(et), (et)t \to t}}{\text{L4: } e(et), (et)t \to et}$$

Here is a more complicated derivation of the same combination:

(23)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{L1: } e(et), e \to et}{\text{L2: } e, e(et), e \to e, et} \;;\quad \text{L1: } e, et \to t}{\text{L3: } e, e(et), e \to t}}{\text{L4: } e, e(et) \to et}}{\text{L2: } e, e(et), (et)t \to et, (et)t} \;;\quad \text{L1: } et, (et)t \to t}{\text{L3: } e, e(et), (et)t \to t}}{\text{L4: } e(et), (et)t \to et}$$

We will later see that different derivations in Lambek's Calculus may correspond to different completions of the type combinations derived. In this particular case, (23) corresponds to the operation *XY* defined in (10) above, but (22) doesn't. Although the systematic reason for this difference can only be fully understood on the background of the material in part 4, there is already an intuitive way of seeing the crucial difference between the two derivations above. For we may distinguish different occurrences of *e* in (22) and (23) according to whether they correspond to the subject or object positions of the transitive verb. In fact, we may think of them as corresponding to slightly different types of individuals, e_s and e_o. Thus, e.g., the first line of (22) becomes:

$$e_o, e_o(e_st) \to e_st$$

because the object is supposed to correspond to the outermost argument, according to our characterization of binary relations. If we carry this indexing through both derivations, we will find that the result of the combination in (23) is a set of subject *e*s, whereas the *e* on the right side of the result in (22) must be co-indexed with the object:

(22') $e_o(e_st), (e_st)t \rightarrow e_ot$
(23') $e_o(e_st), (e_ot)t \rightarrow e_st$

Intuitively speaking, (23') says that the $\mathrm{NP}_{\mathrm{quant}}$ quantifies over the object position, so that the result of the combination will be the set of individuals satisfying the subject position; this is exactly what *XY* does. But in (22') it's just the other way round: the quantifier binds the subject position, and the combination leaves us with the set of individuals that fill the object position. Note that, while this is not what we wanted as our interpretation of the rule that combines transitive verbs with their quantified *objects,* this semantic operation might still be of some use: it does, e.g., open up the possibility of allowing a quantified subject to take narrow scope with respect to the quantified object. We will return to this.

The type combination (20) can be seen as a special case of the general scheme (24) of combinations derivable in Lambek's Calculus:

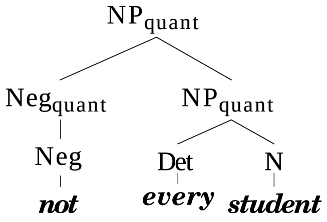
(24) $ab, bc \rightarrow ac$

If we replace all occurrences of '*et*' in (22) by a schematic *b* and then the remaining *t*s and *e*s by *c*s and *a*s, respectively, we can check that the derivation still goes through. (The one in (23), however, does not.) This type combination seems to be particularly fruitful. Note that, if we permute the input types, (17) also turns out to be a special case of this scheme; and the derivation we gave for it has the same structure as that in (22). Another application of (24) is in the derivation of a famous categorial principle called *Geach's Shift* that we can obtain by applying (L4) to (24):

(GS) $bc \rightarrow (ab)(ac)$

The categorial reading of (GS) is that we can *recategorize* functions of some type *bc* into functions of a type (*ab*)(*ac*). On our type-theoretic interpretation of Lambek's Calculus it says that we can have unary syntactic operations to this effect. Thus, e.g., we may analyze negated quantified NPs in this way:

(25)



where we assume the type assignment (2) plus:

(2')

Category	*Kinds of extensions*
Neg	*tt*
$\text{Neg}_{\text{quant}}$	((*et*)*t*)(*et*)*t*

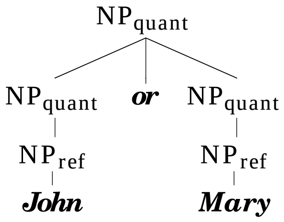
Then the rule '$\text{Neg}_{\text{quant}} \to$ Neg' could be taken care of by (GS). Though the precise semantic content of (GS) will only be discussed in section 4, it should be clear that in the case of negation the result will always be the complement operation.

Unary type combinations like (GS) are usually called *type shifts*. If the input type is smaller or equal in size to the output, the corresponding operations can (but need not) be 1-1 functions, in which case they are usually referred to as (*type*) *liftings*. We will, e.g., see that the most obvious interpretation of Geach's Shift is a type-lifting operation. Another famous lifting is Montague's re-categorization of referential NPs as quantifiers:

(MS) $e \xrightarrow[ML]{} (et)t$

where $ML(u)$ is defined to be $\{P \subseteq D_e \mid u \in P\}$. In the presence of the type assignment (2), (MS) can be used for construing and interpreting coordinations like ***John or Mary*** as NP_{quant}s:

(26)



The interpretation of this particular co-ordination $F(\mathrm{NP}_{\mathrm{quant}}, \mathrm{NP}'_{quant})$ would be: $\{P \subseteq D_e \mid [\![\mathrm{NP}_{\mathrm{quant}}]\!](P) = 1$ or $[\![\mathrm{NP}'_{\mathrm{quant}}]\!](P) = 1\}$, and it would be the same operation as in ***some girl or every boy***.

(MS) is only a special case of a general scheme that is easily derivable in Lambek's Calculus:

(27)
$$\frac{\text{L1: } ab, a \to b}{\text{L4: } a \to (ab)b}$$

Just put $a = e$, $b = t$, and you'll get (MS). Again, we will see that (27) gets the interpretation *ML*.

If we let (GS) and (MS) interact, we may find a solution for the compositionality problem about relative clauses discussed in section 1.4. Before we can show this, a difficulty concerning the type of (restrictive) relative clauses must be overcome. For up to now we have assumed that their extensions are sets so that they would be assigned the type *et*. However, this type-assignment is, in a sense, incompatible with Lambek's Calculus, because it would not even make the (Det (N RelCl)) analysis possible: the type combination $et, et \to et$ is not derivable – even though it does have the natural interpretation of intersection. Although this is a serious problem for the approach under discussion, we shall not deal with it at this point but rather assume that relative clauses should be typed differently in the first place, viz. as *modifiers* of nouns, i.e. functions taking sets to sets. The type corresponding to RelCl would thus be (*et*)(*et*), and a clause like ***who loves John*** would be interpreted by that function in $D_{(et)(et)}$ that takes arbitrary $P \subseteq D_e$ to $P \cap [\![$***loves John***$]\!]$. (The intersection has thus been built into the relative clause meaning.) Clearly, the combination of the noun with the relative clause now is only a matter of functional application. But how about the other bracketing? We could get it by recategorizing the noun as something that still needs to be modified by a relative clause, i.e. a function of type ((*et*)(*et*))(*et*). This kind of type shift is just a special case of (MS). In order to still be able to combine the determiner with this re-categorized noun, we can have Professor Geach lift it:

(28) $(et)((et)t) \rightarrow (r(et))(r((et)t))$,

where *r* is the type (*et*)(*et*) of the relative clause. (Thank you, Sir!) So we are in a position to combine the extension of (lifted) determiner and (lifted) noun by functional application. The result is an NP of the odd type (*r*((*et*)*t*)) of something that is still waiting for a noun modifier in order to become an extension of a (quantified) noun phrase. Of course, the noun modifier is supplied by the relative clause that can now be attached by functional application.

This specific rebracketing analysis, whose interpretation we will have to check later, is due to van Benthem, but the general idea underlying it is a standard categorial technique that has, e.g., been excessively used in generalized phrase structure grammar. Its major complication is that it multiplies syntactic categories and the types corresponding to them, so that people favouring it have tried to do away with global type assignments like (2) and argued for a policy of *type driven interpretation* according to which semantic combinations should be predictable from the type combinations involved. However, time, space, and the teacher's lack of competence force us to leave this strategy out of consideration.

Before we go on to another type-theoretic approach to constraints on semantic operations, a word should be said about the relation on Lambek's Calculus to *propositional logic*. The general format of the axioms and rules introduced in this section is strikingly similar to a certain proof-theoretic approach to (propositional and predicate) logic, viz. Gentzen's *Sequent Calculus*, which is designed to derive all valid logical implications among logical formulae. Closer inspection of the analogy reveals that Lambek's types (or categories) correspond to Gentzen's formulae, the type combination arrow '→' plays the analogous role of logical implication ('⊨'), and functor types *ab* behave essentially like material implications ($\varphi \rightarrow \psi$). In particular, any Lambek-derivable type combination can be translated into a valid implication among formulae of propositional logic that are built up from the atomic formulae *e* and *t* using only material implication. Our example (21) of combining the types of a proper name, a transitive verb and a quantified NP into *t* would become:

(29) $e, e\rightarrow(e\rightarrow t), (e\rightarrow t)\rightarrow t \models t,$

which states that the three formulae to the left of the '|=' logically imply the atomic proposition *t*, which certainly is the case. And in general, every type combination derivable in Lambek's Calculus corresponds to a logical implication in propositional logic. On the other hand, there are many instances of valid inferences whose proofs transcend the axioms and rules allowed by Lambek. A trivial example is the underivable type combination (19) above that would be needed for combining the type *t*(*tt*) of binary connectives with transitive verbs. It corresponds to a trivial implication in propositional logic:

(30) $t\rightarrow(t\rightarrow t), e(e\rightarrow t), e(e\rightarrow t) \models e(e\rightarrow t)$

The fact that (the type combination corresponding to) this implication cannot be derived in Lambek's Calculus shows its *incompleteness* with respect to (the →-fragment of) propositional logic. This in itself is, of course, not necessarily a defect; for it was not propositional logic that the calculus was designed for. However, there is some evidence for the conjecture that any way of strengthening Lambek's Calculus for getting a more adequate theory of semantic operations must get it closer to logical implication. (30) already points in that direction. More evidence will be found as we go along.

3.6 Logicality

As was already mentioned in section 3.4, another approach to a theory of semantic operations is to look for properties all existing cases (and, plausibly, all possible ones) share. One such property is *logicality*, i.e. the fact that they are all purely structural operations, that their values for specific arguments only depends on their logical relations to each other and on nothing else. So while there might be semantic operations that give us the intersection of two sets X and Y, we never seem to get the oldest members of X and Y, or their grandmothers. Maybe this is not the only property semantic operations have in common, but it seems to be hard to find any plausible counter-examples. So logicality is a good guess for at least one constraint on semantic operations.

The problem is how to make these ideas precise. But then an advantage of the type-theoretic framework is that it naturally leads to at least one necessary condition on logicality. (Moreover, we will see that it might also throw some light on other approaches to constraining semantic operations.) Before we start looking at this condition, we should note that the kinds of operations we are interested in can all be found in the ontologies themselves! For if G appears in a type combination like:

(12) $a_1, \ldots, a_n \underset{G}{\rightarrow} a_{n+1}$,

it is a function taking us from $D_{a_1} \times \ldots \times D_{a_n}$ to $D_{a_{n+1}}$. Now, using the same kind of correspondence as we did between $D_{e(et)}$ and $\wp(D_e \times D_e)$, we may think of G as a function of type $(a_n(\ldots(a_1 a_{n+1})\ldots)$, viz. as that f_G satisfying: $f(\mathrm{u}_n)\ldots(\mathrm{u}_1) = G(\mathrm{u}_1,\ldots,\mathrm{u}_n)$. So our operation XY combining binary relations and quantifiers into sets of individuals could be found in type $(((et)t)((e(et))(et)))$. Consequently, instead of looking for logicality among semantic operations, we may as well search an ontology for its *logical objects*: if the operation G is logical, then so is its counterpart f_G, and vice versa. In what follows, we will think of logicality as a property of objects in an (arbitrary) ontology.

In order to motivate our criterion of logicality, we will fix an ontology and successively look at some of its layers D_a trying to figure out which of its elements are logical. Our hope is that some pattern will emerge. So let us start with type t. D_t contains the truth-values 0 and 1, and they certainly are logical objects. So if we write 'L_a' for the logical objects of type a (in our ontology), it seems plausible to assume that $L_t = D_t$. Next, consider tt, the type of unary truth-functions of which there are exactly four:

$$\mathbf{F} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}; id_t = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}; \neg = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Again it seems clear that all of the above are logical objects: negation and identity are anyway, and there is certainly something logical about constant functions to logical objects, or so we will assume. Hence $L_{tt} = D_{tt}$. If we now look at the type $t(tt)$ of binary truth-functions, i.e. the extensions of binary connectives, we will again find that there is something logical about all of them: $\wedge$, $\vee$, $\rightarrow$, etc. are logical constants, and since each of them, together with $\neg$, can be used to define all other binary truth-tables, the latter should all count as logical: for it seems safe to assume that whatever can be defined solely in terms of logical objects is again logical. (The latter assumption will actually be questioned later on; but these doubts will not affect the present considerations.) So we may adopt the equation $L_{t(tt)} = D_{t(tt)}$. By the same kind of reasoning it follows that all truth-tables should count as logical objects, i.e. $L_{t(\ldots(tt)\ldots)} = D_{t(\ldots(tt)\ldots)}$. In fact the criterion of logicality to be developed will tell us that $L_a = D_a$ whenever a is made out of ts only; this principle also covers more exotic types like $(tt)(tt)$, $(t((t(tt))(tt)))$, etc.

So how about the other types? The simplest of them is e, and this time it is obvious that it does not contain *any* logical objects at all: the ontological framework treats individuals as objects without any internal structure and so, in particular, none of them has any logical structure that distinguishes it from the others of its type. (A little care must be taken if the ontology is not normal; for then an individual might be a logical object of a type other than e. Still it wouldn't be logical *qua* individual, i.e. it would not be in L_e.) We thus conclude that $L_e = \emptyset$.

What is true of individuals, is not true of sets of them. For the type *et* does contain at least two members that are determined by their structure, viz. Ø and D_e(or, more accurately, their characteristic functions). The former is that function that assigns to any individual whatsoever the logical object 0, and the latter does the same thing with 1. In particular, these two objects can be described without any reference to what an individual is, how many of them there are, and how they can be distinguished from each other. And that clearly indicates that both Ø and D_e are logical objects of type (*et*). Are there more? Apparently not. For consider a set like $\{u, v, w\} \subseteq D_e$. What should be logical about it? The fact that it contains u? No, because u is just any old individual and containing it as a member is not a *structural* feature of $\{u, v, w\}$. How about the fact that this set contains three members? Well, if D_e happens to coincide with $\{u, v, w\}$, then this may actually be taken as a purely structural feature that can be used to single out this set. But then it would be in L_{et} anyway. If, on the other hand, D_e is larger, then the property of having 3 members is shared by, say, $\{u, v, w'\}$ and so it cannot be used to distinguish $\{u, v, w\}$ from the latter. The conclusion to be drawn from these considerations is that $L_{et} = \{\emptyset, D_e\}$.

Checking the examples discussed so far, one might guess that logical objects are either truth-values or constant functions to logical objects. However, things are more complicated, as a quick glance at the type *ee* of functions from individuals to individuals shows. For although the range type *e* does not contain any logical objects (i.e. $D_e = \emptyset$), one logical function of type *ee* certainly exists, viz. the identity function over id_e that maps every individual to itself. Again we may wonder whether id_e is the only element of L_{ee}. And again the answer is positive. To see this consider a non-identity $f \in D_{ee}$ on the very small domain $\{u, v, w\}$:

$$\begin{bmatrix} u & v \\ v & u \\ w & w \end{bmatrix}$$

A purely structural description of *f* is easily given: *f* is a 1-1 function mapping exactly one individual onto itself. (This is only one possible description of *f*'s structural features.) The following function *g* meets the same criteria:

$$\begin{bmatrix} u & u \\ v & w \\ w & v \end{bmatrix}$$

Since *f* and *g* are distinct but structurally identical, none of them can be identified by its structural features, and so none of them is a logical object. Now it should not come as a great surprise that the same kind of reasoning can be turned into a general argument to the effect that no non-identity is logical. Hence we conclude that $L_{ee} = \{id_e\}$.

We are now in a position to see the general pattern behind the distinction of logical and non-logical objects. For the reason why we did not want to count $\{u, v, w\} \subseteq D_e$ as logical was that replacing w by w' would have resulted in a structurally indistinguishable set. Similarly, a comparison of the two *ee*-function *f* and *g* above shows that the latter can be obtained from the former by (simultaneously) replacing u by v, v by w, and w by u. So what makes these objects non-logical is the fact that certain replacements of individuals lead to structurally identical but distinct objects. This can never happen with a logical object, because nothing but itself is structurally identical to it: if we drew a picture containing arrows from its arguments to its values but without identifying any individual, the logical object would still be uniquely described. This is the essence of our criterion of logicality. In order to make it precise, we only have to say what it means to change one object into another one by replacing individuals. But this is not very hard. A replacement of individuals or *permutation* is simply a 1-1 function from D_e onto D_e, i.e. a function π ($\in D_{ee}$!) such that $\pi(u) \neq \pi(v)$, whenever $u \neq v$, and such that every $v \in D_e$ is the value of one $u \in D_e$, i.e. $\pi(u) = v$. Given such a permutation π, we must then describe its effect on objects of arbitrary types. This can be done by a simple recursion resulting in a family $(\pi_a)_{a \in T}$ of (structure-preserving) permutations:

(31) (a) $\pi_e(u) = \pi(u)$, if $u \in D_e$;

(b) $\pi_t(u) = u$, if $u \in D_t$;

(c) $\pi_{ab}(f) = \{(\pi_a(u), \pi_b(v)) \mid f(u) = v\}$.

Clause (b) is only needed for the induction; of course, replacing *individuals* has no effect on truth-values. Clause (c) makes use of the fact that functions are sets of ordered pairs. What happens to them can be easily visualized by representing them as matrices, as we did above: if we substitute the individuals in the left column, the corresponding changes have to be made on the right side. A not too complicated inductive proof would show that, for any type a, π_a is always a permutation on D_a, although not every permutation on D_a can be obtained by starting out from some π_e.)

Given (31), we can now *define* logicality to be the property of not being affected by permutations of individuals:

(32) $L_a = \{u \in D_a \mid \pi_a(u) = u$, for all permutations π on $D_e\}$.

In order to see that (32) matches (or comes at least close to) our previous intuitions about logical objects, we just check the types discussed above. Clearly, the truth-values come out as being logical because, due to clause (b) of (31), no permutation whatsoever affects them and hence the defining clause in (32) is always met. Moreover, this observation can be used in a homework exercise to show that $L_a = D_a$, whenever a does not contain e. So let us go on to the other types.

The type e of individuals seems to present no problem, because apparently every $u \in D_e$ can be permuted by, say:

(33)

$$\begin{bmatrix} u & v \\ v & u \\ w & w \\ w' & w' \\ w'' & w'' \\ \ldots & \ldots \end{bmatrix},$$

i.e. the function interchanging u and v and mapping everything else onto itself. However, for this trick to work, there must be at least one v different from u and thus the ontology must contain at least two individuals. This is not an undesired effect: if there is only one individual, it can be singled out by its structural property of being an (i.e.: *the*) individual! In fact, it can be shown that, in this limiting case, L_a will always coincide with D_a. However, apart from this somewhat neurotic case, we can always use functions as in (33) to show that $L_e = \emptyset$, as desired.

In order to see that sets of individuals are only logical if they happen to contain everything or nothing, we consider some set $M \subseteq D_e$ that is neither empty nor identical to D_e and show that it can be mapped onto something different from itself. M is not empty; so there must be some $u \in M$. M is not universal; so there must be some $v \in D_e \backslash M$. Surely, $u \neq v$. Now let π be the permutation depicted in (33). We want to show that π does not map M onto itself, i.e. that $\pi_{et}(\chi^e_M) \neq \chi^e_M$. It suffices to find one argument over which the two disagree; u is one:

$$\begin{array}{ll} & \pi_{et}(\chi^e_M)(u) = 1 \\ \text{iff} & (u,1) \in \pi_{et}(\chi^e_M), \text{ by notational convention,} \\ \text{iff} & (\pi_e(u),\pi_t(1)) \in \chi^e_M, \text{ by (31) (c),} \\ \text{iff} & (\pi_e(u),1) \in \chi^e_M, \text{ by (31) (b),} \\ \text{iff} & \pi_e(u) \in M, \text{ by definition of } \chi^e_M, \\ \text{iff} & v \in M, \text{ by definition of } \pi, \end{array}$$

which is not the case. So $\pi_{et}(\chi^e_M)(u) \neq 1$; but, clearly, $\chi^e_M(u) = 1$, because we picked u from M. So, as long as M is not empty or universal, we can always find a permutation π that maps it onto a different set, so that M cannot be in L_{et}. The fact that, conversely, both $\emptyset$ and D_e are in L_{et} will be shown in a homework exercise.

We finally check the type ee. If $f \neq id_e$, we must disturb it by some permutation π. But since $f \neq id_e$, we can find some distinct v and w such that $f(w) = v$. If we now pick a third element u from D_e, we can again use the permutation given in (33) to show that $\pi_{et}(f)(w) \neq v$. We will not go through this calculation, because it is similar to the above argument about $\pi_{et}(\ \chi_M^e)(u)$. (We should note that this construction depends on D_e having at least three elements; in fact, if there are only two of them, we get one more logical object, viz. that 1-1 function on D_e that is not the identical mapping!) We must also verify that $id_e \in L_{ee}$. However, this is trivial, because any permutation π and any $u \in D_e$ satisfy:

$$\begin{array}{ll} & \pi_{ee}(id_e)(u) = \mathrm{u} \\ \text{iff} & (\pi_e(u),\pi_e(u)) \in id_e \\ \text{iff} & \pi_e(u) = \pi_e(u). \end{array}$$

Moreover, this argument is quite independent of the type e, so that in general we can conclude that $id_a \in L_{aa}$. Note that this does not mean that $L_{aa} = \{id_a\}$; $a = t$ was a counter-example.

Returning to our original motivation for this investigation in the notion of logicality, it may already be intuitively clear that all semantic operations discussed so far are indeed logical objects. We only check a very simple case, functional application (FA). FA is actually schematic in the sense that it appears in different types. Thus, for any types a and b, application of ab-type functions to a-type objects corresponds to the operation $FA_{ab} \in D_{(ab)(ab)}$ defined by:

$$FA_{ab}(f)(u) = f(u).$$

So FA_{ab} assigns to any $f \in D_{ab}$ that function $g \in D_{ab}$ that assigns $f(u)$ to any $u \in D_a$. Hence f and g agree on all $u \in D_a$, which means that $f = g$! So FA_{ab} turns out to be id_{ab}, and that is in $L_{(ab)(ab)}$, as was just pointed out.

As to the other semantic operations discussed above, we will soon see a general method of proving that certain objects are logical; and although this method does not cover all possible cases, it will be applicable to them. In particular, we will see that not only the rather complicated operation *XY* is logical, but also certain operations that correspond to type combinations underivable in Lambek's Calculus. Thus, e.g., intersection of sets of individuals and the disjunction of quantified NPs both correspond to logical objects, as will become clear from the discussion in part 4. So is the criterion (32) of logicality the key to the problem of determining what a possible semantic operation is? No. For there are several indications to the effect that it is incomplete. One is that it might allow for too many semantic operations to be of interest. For as types a get larger, the cardinalities of the L_a increase considerably: it can, e.g., be proved that $L_{(et)t}$ contains more elements than there are natural numbers, provided that D_e is an infinite set. Another problem concerns combinations of two sets of individuals into another one: intersection is one logical operation, but union is another one. Unfortunately, the latter does not seem to appear in any applications and, without any additions, the approach under discussion does not say why one should be preferred over the other. The biggest problem about the definition (32) of logicality, however, is its inherent *locality*: it can only be applied within a previously fixed ontology, and there is no obvious way of generalizing it across ontologies. However, this would have to be done, given our general model-theoretic approach: functional application, even restricted to a fixed combination of types, is not just one operation but a whole family of them, with a member in each ontology. It thus seems that the logicality approach to constraining semantic operations cannot be the whole story. But to some semanticists it appears to be a promising first step; and it can also be used to learn more about the other approaches.

-

Exercises

4. Given any universe D ($\neq \emptyset$), what should be the generalized quantifier $[\![\textbf{\textit{nothing}}]\!] \subseteq \wp(D)$? Which function $f_{\textbf{\textit{nothing}}} \in D_{(et)t}$ does it correspond to?

5. The following clause relates any binary truth-functional connective K to a corresponding function f_K of type $(t(tt))$:

$$f_K\,(\mathrm{v})(\mathrm{u}) = K(\mathrm{u},\mathrm{v}), \text{ whenever u and v are truth-values.}$$

   Thus every K corresponds to a binary relation among truth-values. Specify the relations thus corresponding to $\wedge$ (conjunction), $\vee$ (disjunction),and $\rightarrow$ (material implication).

6. Here is yet another derivation of (20) in Lambek's Calculus:

   (!)

$$\begin{array}{ll}
\text{L1: } e,\, e(et) \rightarrow et & \\ \cline{1-1}
\text{L2: } e,\, e,\, e(et) \rightarrow e,\, et\ ; & \text{L1: } e,\, et \rightarrow t \\ \hline
\text{L3: } e,\, e,\, e(et) \rightarrow t & \\ \cline{1-1}
\text{L4: } e,\, e(et) \rightarrow et & \\ \cline{1-1}
\text{L2: } e,\, e(et),\, (et)t \rightarrow et,\, (et)t\ ; & \text{L1: } et,\, (et)t \rightarrow t \\ \hline
\text{L3: } e,\, e(et),\, (et)t \rightarrow t & \\ \cline{1-1}
\text{L4: } e(et),\, (et)t \rightarrow et &
\end{array}$$

   Apply the method of indexing according to object/subject positions to determine whether (!) corresponds to XY.

7. The type shift:

$$(\uparrow)\quad e(et) \rightarrow n(nt),$$

   where n is the type $(et)t$ of quantified NPs is usually attributed to Montague. Show that one can derive $(\uparrow)$ in Lambek's Calculus. Try to find a derivation that makes the leftmost n correspond to the object position. Hint: Assume the derivation (22) for

$$e(et),\, n \rightarrow et$$

   and then take the second n into account.

8. Show that $L_a = D_a$ whenever a does not contain any e. Hint: Proceed inductively starting with $a = t$; for complex types bc you may then assume that $L_b = D_b$ and that $L_c = D_c$.

9. Show that the characteristic functions of Ø and D_e are in L_{et}.

4. Indirect Interpretation

4.1 Use and abuse of notation

Indirect interpretation is a popular and powerful method of defining and presenting semantic analyses. The method basically consists in systematically assigning formulae of a suitable logical language to natural language expressions or their structures. These formulae will then be compositionally interpreted in a model-theoretic framework based on a theory of types, and this interpretation carries over to the original natural language expressions.

Although the advantages of indirect interpretation can only be fully appreciated when one has seen it at work, some *a priori* considerations about its usefulness can be used to motivate the introduction of this formal apparatus. The first point to be made is that the use of formal notation often (though by no means always) leads to a better understanding and handling of complex material problems: it would have been very hard to define and understand the semantic techniques and concepts discussed in the previous parts without the use of any quasi-mathematical notation; and the formulae to be used in indirect interpretation systematize (and reduce) our notational inventory. Secondly, a canonical notation for all sorts of semantic analyses can serve as a kind of semantic *lingua franca* that not only increases readability but may also allow for the possibility of convenient and precise comparisons of rivaling descriptions. More importantly, since the languages used in indirect interpretation are formally (i.e. mathematically) defined objects, they give rise to natural measures of semantic complexity and, thereby, to substantial constraints on the semantic structure of natural language. We will, e.g., be in a position to formulate various *definability hypotheses* to constrain the notion of a possible semantic operation as one that can be defined within (a fragment of) a certain language of indirect interpretation.

Indirect interpretation also bears its risks and dangers that one should be aware of and try to avoid. Two of them must be particularly emphasized. The first is that the method easily gives the user the impression that its product, i.e. the logical formulae, are semantic analyses or even meanings in themselves. From our present armchair perspective, this confusion might appear far-fetched: the formulae only serve as alternative means of expressing what the natural language expressions say and, as such, they must be interpreted to be made sense of; for who would say that the meaning of the German word ***Tür*** is (identical to) the English word ***door***, even though the latter may be used to explain the meaning of the former? It is still true that excessive use of indirect interpretation frequently leads to the kind of confusion indicated. The best remedy against it is a permanent awareness of the fact that formulae are formulae and only get their meanings from semantic procedures and definitions; beginners should develop and cultivate this awareness by frequent 'evaluation exercises', in which the real meanings of logical formulae are determined. The second trap of indirect interpretation is its compositionality which is only guaranteed if the whole procedure meets certain conditions. Again, experience shows that it is easy to forget about these conditions, which may result not only in non-compositional assignments but even in inconsistency. We will have to return to this point in section 4.6.

4.2 Ty1: identity, application, abstraction

We will now look at one particular logical language that has proved useful in the indirect interpretation of certain fragments of natural language. It is the language Ty1 of *one-sorted type theory*. (One-sortedness is the feature that distinguishes it from a language based on a richer hierarchy of types to be discussed in part 5.) Like first-order logic, Ty1 contains certain basic expressions (variables and constants) and ways of combining them (like the connectives and quantifiers). But unlike first-order logic, it has infinitely many syntactic categories corresponding to the functional types introduced in part 3. It is this feature and the particular choice of basic means of expressions that make it a better tool for indirect interpretation than predicate logic. Unfortunately, this choice cannot be motivated by just considering the purpose of Ty1; we will instead just introduce them in the firm hope that they will prove to be useful for us.

Since we want the syntactic categories to match the functional types, we may as well identify the two sets. So instead of the usual category labels like 'S', 'NP', etc. (in natural language) or 'wff', 'individual term', etc. (in predicate logic), Ty1 has 'e', 't', '$(et)t$', etc. And, as a *façon de parler*, we will often speak of expressions of a category $a \in$ T as expressions *of type a*. This should not give rise to confusions. (But be aware of the above warnings!) The simplest, lexical expressions of Ty1 are the variables and constants. We will assume that, for each $a \in$ T, we have infinitely many variables of type a. This may sound a bit exaggerated, but it will put us in a position to quantify over everything in the ontology (though not simultaneously) and to have arbitrarily deep nestings of such quantifiers. How many constants there are and which types they belong to is not so important. Their role will be (just like in first-order logic formalizations) to translate lexical expressions of natural language (unless we want to decompose them). What is important, though, is the fact that we think of these constants as non-logical, like the predicates and individual constants of first-order logic. Thus, there are no logical constants in Ty1. This may appear odd, but let's wait and see.

There are three ways of combining Ty1-expressions, and two of them are very easy to define and understand. The first one is the operation of *Identity* which transforms two expressions, α and β, of the same category $a \in$ T into the equation $\ulcorner(\alpha = \beta)\urcorner$, which is an expression of category t, i.e. a sentence. The other simple operation is *Application*, i.e. that syntactic operation of Ty1 that can be applied to α of category ab and β of category a and yields the expression $\ulcorner\alpha(\beta)\urcorner$ of category b. Although we we will provide the precise semantic definitions in a second, the intended interpretation of these two operations should already be clear: Identity gives the truth-value 1 if the to expressions flanking the equality sign extensionally coincide; and Application expresses *FA*, so that $\ulcorner\alpha(\beta)\urcorner$ denotes the result of applying the function denoted by α to β's extension.

The interpretation of the third and final operation of Ty1 is less obvious. We will motivate it by an example. We have seen that, according to one plausible analysis of restrictive relative clauses, they denote sets of individuals. Thus, e.g., (1) could be analyzed as denoting the set M of those individuals that love Mary:

(1) ***who loves Mary***

How can we compositionally assign M to (1)? The most straightforward way is this: we analyze the relative clause exactly in the same way as we would the corresponding sentence, except that we think of the relative pronoun as a variable (ranging over individuals). So (1) would have to be treated just like:

(1') x ***loves Mary***

Given the indirect approach to interpretation, such an analysis should not be too difficult: we would just have to *translate* ***who*** by the variable x of type *e*. We could then use the same mechanisms of interpretation in (1) as we would use in analyzing

(2) ***John loves Mary***,

the only difference between the two being that (1) has a variable where (2) has a constant of the same type. However, this cannot be the whole story about relative clauses. For the result of analyzing (1) in the same way as (2) would give us an object of type *t*, i.e. a truth-value. And, what is worse, this truth-value would depend on the denotation of the free variable x. This is certainly not what we want as the denotation of (1). What we do want is a *set* of individuals, viz. the set of those x that satisfy (1'). In other words, we do not want to read (1') – or whatever formula will correspond to it – as a statement about some particular individual x, but rather as a *condition* on *arbitrary* x. We would thus have to change the type of (1') from *t* to *et* and at the same time make sure that the x gets bound. Here is the notation:

(1") [λx x ***loves Mary***]

Thus 'λ' is an operator binding the variable x of type *e* and turning the type-*t*-expression (1') into the set-denoting expression (1"). How do we read (1")? The 'λx' turns free and fixed x into an arbitrary argument. So (1") is the *et*-function that yields (1') when applied to an arbitrary x in its domain. This formulation already shows how to read the λ-operator in the general case. To get some more feeling about what it does, let us look at a few more examples. One of the most revealing ones is from high-school mathematics, where poor little kids spend a long time discussing certain functions from real numbers to real numbers, like the one taking every number x to x^2. It is common practise to refer to this function by one of the following quasi-formal notations:

(3) $f(x) = x^2$

(3') $y = x^2$

Note that both the above expressions are equations and thus, strictly speaking of category *t* (if we were to think of them as logical formulae). So none of them would actually denote a function. With the λ-operator we would, however, have a straightforward and unambiguous way of referring to the intended function:

(3") $[\lambda x\, x^2]$

Note that the x gets bound, because (3") is supposed to denote that function that takes *arbitrary* numbers x to x^2. What is the type of this function? If we assume that real numbers are of some type *r*, then (3") denotes a function of type (*rr*), because its arbitrary arguments x are of type *r* and so are its values x^2 (since x^2 is a real number as long as x is).

In natural language semantics, λ-operators are particularly useful in solving certain kinds of compositionality problems. We will illustrate the basic idea by a well-known example, the analysis of quantified noun-phrases. Concentrating solely on the subject position, the problem of finding suitable denotations for NP_{quant}-expressions α can be attacked by considering adequate translations of sentences in which the αs occur and observing that they *schematically vary* when we replace the VP, leaving α in subject position: if α = ***every student***, we always get something of the form (4), where ‘S’ symbolizes the extension of ***student***; similarly, the noun phrase α = ***some student*** will always lead to translations of the form (5), no matter which VP we take it to be the subject of. (To be sure, the ‘always’ in these statements has to be taken *cum grano salis*, because for some sentences these formalizations do not work, for one reason or another; but let us for the moment ignore these trouble-makers.)

(4) $(\forall x)\,[S(x) \rightarrow X(x)]$

(5) $(\exists x)\,[S(x)\ \&\ X(x)]$

It is the ‘X’ in these formulae that systematically changes with the VP: once we fix a particular verb phrase β, we may identify X with β’s extension. Note that, in case β is complex, we might not know how to express its extension by a formula, let alone how to obtain this formula in a systematic way. But we do (usually) have some intuitions about which set this extension is going to be. And in any given case, we can think of X as referring to this extension.

The undetermined character of X in (4) and (5) suggests that we may take it to be a *variable* ranging over (possible) predicate denotations; the type of this variable would thus be *et*. And the fact that (4) and (5) each systematically depend on X indicates that we may think of the corresponding noun phrases as each denoting a *function* taking VP denotations X to the truth-value of the complete formula:

(4') $[\lambda X\ (\forall x)\ [S(x) \rightarrow X(x)]\ ]$

(5') $[\lambda X\ (\exists x)\ [S(x)\ \&\ X(x)]\ ]$

(4') and (5') can thus be used as translations of the noun phrases ***every student*** and ***some student***, respectively. Clearly, the functions denoted by them are exactly the sets of type (*et*)*t* that we had been using as NP_{quant}-denotations all along; but it is interesting to see how they can be obtained from the intended translations of (certain) sentences by the use of Abstraction.

In order to see how general the idea underlying the use of Abstraction in the above example is, we apply it once more to obtain the translation of the determiners ***every*** and ***some*** from (4') and (5'). Again we have a systematic variation in the denotation of the NP_{quant}, this time depending on what the noun is: in general, noun phrases of the form ⌜***every*** N⌝ and ⌜***some*** N⌝ denote quantifiers that can be symbolized by (4") and (5") respectively:

(4") $[\lambda X\ (\forall x)\ [Y(x) \rightarrow X(x)]\ ]$

(5") $[\lambda X\ (\exists x)\ [Y(x)\ \&\ X(x)]\ ]$

(Y varies over (possible) denotations of nouns, i.e. it is a variable of type *et*.) And again we can see that systematic variation amounts to functional dependence, so that the determiners ***every*** and ***some*** can be translated as:

($4_\forall$) $[\lambda Y\ [\lambda X\ (\forall x)\ [Y(x) \rightarrow X(x)]\ ]\ ]$

($5_\exists$) $[\lambda Y\ [\lambda X\ (\exists x)\ [Y(x)\ \&\ X(x)]\ ]\ ]$

The general pattern in these examples is that of *semantic decomposition by Abstraction*: under the assumption that we can systematically predict the extensions $G\,([\![\alpha]\!],[\![\beta]\!])$ of complex expressions $F\,(\alpha,\beta)$ from the (assumed) extensions of the βs, we may assign to α the function *f* yielding G ($[\![\alpha]\!]$,x) when applied to any x (of the same type as the βs). Now if $G([\![\alpha]\!]$,x) can be expressed by a logical formula φ, it is easily seen that $\ulcorner[\lambda x\ \varphi]\urcorner$ denotes f. Note that this method only works if the βs' extensions are somehow (assumed to be) previously given and, more importantly, that extensionally equivalent β and β' can be substituted for one another without changing the extension of the result. It should also be clear that, even if this method is applicable, we are by no means forced to apply it or even accept its conclusion that α's denotation should be f. But it frequently helps to determine denotations this way – especially if we have no clue to what else they could be.

We are now in a position to give a precise formulation of the syntax and semantics of Ty1. First the main syntactic definitions:

<u>*Syntax of Ty1*</u>:

(a) For each type $a \in$ T, let Var$_a$ (= { x^n_a | $n \in \omega$, $a \in$ T}) be the set of variables of type a and let Con$_a$ be a suitable set of constants of type a.

(b) The sets Ty1$_a$ of Ty1-expressions of category $a \in$ T are defined by the following recursion:

(Lex) (Var$_a \cup$ Con$_a$) $\subseteq$ Ty1$_a$;
(Id) if $\alpha \in$ Ty1$_a$ and $\beta \in$ Ty1$_a$, then $\ulcorner(\alpha = \beta)\urcorner \in$ Ty1$_t$;
(App) if $\alpha \in$ Ty1$_{ab}$ and $\beta \in$ Ty1$_a$, then $\ulcorner\alpha(\beta)\urcorner \in$ Ty1$_b$;
(Abs) if $x \in$ Var$_a$ and $\alpha \in$ Ty$_b$, then $\ulcorner[\lambda x \alpha]\urcorner \in$ Ty1$_{ab}$.

All other syntactic concepts are defined as their analogues in predicate logic: a *Ty1-sentence* is a Ty1-expression of category t, variables (i.e. their occurrences) within the scope of λ are *bound*, the others are *free*, expressions without free variables are called *closed*, etc. Note that Abstraction is the only variable-binding operation in Ty1.

The semantics of Ty1 consists in compositional assignments of extensions (of denotations) to the expressions of Ty1. These assignments depend on models that take care of the lexical expressions (= variables and constants) and that themselves depend on a previously defined ontology. Since we want the whole procedure to be compositional, we will have to interpret variables by variable assignments and the variable binding operation of Abstraction by an operation taking into account several variables assignments at the same time. The procedure is just as cumbersome as Tarski's satisfaction semantics of predicate logic and could just as well be replaced by a non-compositional treatment of Abstraction by substitution. We will, however, give the Tarskian version – be it only for the reason that it is the more standard one:

<u>*Semantics of Ty1*</u>:

(a) An *interpretation* (based on an ontology $(D_a)_{a\in T}$) is a function F taking each constant $c \in Con_a$ to an element of D_a: $F(c) \in D_a$, whenever $c \in Con_a$.

(b) A *variable assignment* (based on an ontology $(D_a)_{a\in T}$) is a function g taking each variable $x \in Var_a$ to an element of D_a: $g(x) \in D_a$, whenever $x \in Var_a$.

(c) A *model* is a triple $((D_a)_{a\in T}, F, g)$, where $(D_a)_{a\in T}$ is an ontology on which both the interpretation F and the variable assignment g are based.

(d) Given a model $M = ((D_a)_{a\in T}, F, g)$, the *extension* $[\![\alpha]\!]^{F,g}$ (= $[\![\alpha]\!]^{M}$)of any Ty1-expression α is determined by the following recursion:

(Lex) $[\![\alpha]\!]^{F,g} = F(\alpha)$, if α is a constant, and $[\![\alpha]\!]^{F,g} = g(\alpha)$, if α is a variable;

(Id) $[\![\alpha]\!]^{F,g} = 1$ if α is $\ulcorner(\beta = \gamma)\urcorner$ and $[\![\beta]\!]^{F,g} = [\![\gamma]\!]^{F,g}$; and $[\![\alpha]\!]^{F,g} = 0$
if α is $\ulcorner(\beta = \gamma)\urcorner$ but $[\![\beta]\!]^{F,g} \neq [\![\gamma]\!]^{F,g}$;

(App) if α is $\ulcorner\beta(\gamma)\urcorner$, then $[\![\alpha]\!]^{F,g} = [\![\beta]\!]^{F,g}([\![\gamma]\!]^{F,g})$, i.e. the result of applying the function $[\![\beta]\!]^{F,g}$ to the argument $[\![\gamma]\!]^{F,g}$;

(Abs) if $x \in Var_a$ and α is $\ulcorner[\lambda x\beta]\urcorner$, then $[\![\alpha]\!]^{F,g}$ is the function taking any $u \in D_a$ to $[\![\beta]\!]^{F,g[^x/_u]}$, where $g[^x/_u]$ is as g, except that $g(x) = u$, i.e. $g[^x/_u] = (g \setminus \{(x,g(x))\}) \cup \{(x,u\}$.

In case the clause (Abs) is a bit too much, here is a slightly more prosaic version: 'the extension of $[\lambda x\beta]$ is that function which maps arbitrary u on β's extension, provided that x is understood as referring to u'. Note that the above clauses guarantee – and, in the case of (App), even presuppose – that the extension of an expression of a category $a \in T_a$ is always an object of type a. One particular case of this clause deserves special attention both because it is somewhat neurotic and because it is the most common one to be encountered in indirect interpretation. We have seen how the λ-operator can be used to obtain a plausible logical translations of relative clauses. But reading

(6) $[\lambda x\ L(m,x)]$

as 'that function that maps arbitrary individuals u to the extension of L(m,x), where x is understood as referring to u' does not make too much intuitive sense because we usually do not think of a sentence (or formula of category t) as referring to a truth-value even though technically speaking it does. In order to read notations like (6), it is therefore better to switch from characteristic functions to the corresponding sets. Since the function denoted by (6) yields 1 for precisely those objects of which L(m,x) is true, it characterizes the set of individuals that *satisfy* this condition. (6) is thus the Ty1-analogue of the meta-linguistic notation (6') of *set-abstraction* that we have been using all the time:

(6') $\{u \in D_e | [\![L(m,x)]\!]^{F,g[^x/_u]} = 1\}$

Consequently, and quite generally, Ty1-formulae of the form $\ulcorner[\lambda x_a\ \varphi]\urcorner$ can be read 'the set of $x \in D_a$ such that φ', *provided that* φ *is of category* t. (The subscript on the first occurrence of a variable in a formula indicates its type.) This simple insight will make a lot of λ-terms much easier to read.

4.3 Expressive power

The formulae used in the above examples contained familiar logical constants from predicate logic, although Ty1 does not. In fact, it might appear that the expressive power of Ty1 is very restricted. However, everything predicate logic has can be *defined* in Ty1 in pretty much the same way as, e.g., all truth-functional connectives can be defined in terms of & and ¬. In order to prove this, it obviously suffices to show that universal quantification, negation, and conjunction are all definable in terms of Identity, Application, and Abstraction.

We start with ∀. In order to reduce it to a combination of Ty1-operations, we first observe that a universally quantified formula

(7) $(\forall x_a)\ \varphi$

is true just in case the set of objects satisfying φ coincides with the complete layer D_a:

(7') $\{u \in D_e \mid [\![\varphi]\!]^{F,g[^x/_u]} = 1\} = D_a$

Now, from our above observation about abstractions from type-*t*-formulae we see that the left side of (7) can be expressed in Ty1. Moreover, identity is a built-in operation of Ty1. So all we need in order to express (7) is a way of denoting D_a or, more precisely its characteristic function $\chi^a_{D_a}$. Clearly, $\chi^a_{D_a}$ is *that function that assigns to any u∈D_a the truth-value 1*, so if the truth-value 1 is definable in Ty1, we could get $\chi^a_{D_a}$ by abstraction. But the truth-value 1 is the common denotation of all Ty1-formulae of the form $\ulcorner(\alpha = \alpha)\urcorner$. We thus have a way of expressing (7') and, *a fortiori* (7), in Ty1:

(7") $(\ [\lambda x_a\ \varphi] = [\lambda x_a\ (x = x)]\ )$

Consequently, we can think of (7) as a mere abbreviation of the Ty1-formula (7"). (Note that the variable x_a bound by the universal quantifier in (7) must be the very same as that occurring in (7')!) In this sense Ty1 contains the means of expressing universal quantification.

There is a slightly different way of paraphrasing the universal quantifier in Ty1. For if we take ∀ to be a generalized quantifier (of type (*at*)*t*) meaning 'everything of type *a*', it is just the singleton $\{D_a\}$. Can we define it in Ty1? Yes, because singletons $\{u\}$ are always definable whenever their members *u* are: $\{u\} = \{v \mid v = u\}$, which can be obtained by set-abstraction from an identity statement. Our definition of ∀ as a generalized quantifier would thus be:

(8) $[\lambda P_{at}\ (P = [\lambda x_a\ (x = x)]\ )\ ]$

What is the relation between (7) and (8)? Since (7) can be read as 'everything in D_a satisfies φ', we could paraphrase it by applying the quantifier defined in (8), i.e. $\{D_a\}$ to the set of objects satisfying φ:

(9) $[\lambda P_{at} (P = [\lambda x_a (x = x)]\,)\,] ([\lambda x_a\ \varphi])$

This way of reducing (7) shows that we can think of the symbol '∀' as denoting a particular generalized quantifier (rather than a variable-binding operator) that is applied to a set defined by abstraction:

(9') $\forall\ ([\lambda x_a\ \varphi])$,

where '∀' is just an abbreviation of (8) or name of the quantifier it denotes. Since (7") and (9) are obviously equivalent – a fact to which we will return in due course – we do not have to decide which of them we are going to adopt as our official definition of (7). In any case it is clear that the following holds for arbitrary models $((D_a)_{a\in T}, F, g)$:

(∀) If $x \in Var_a$ and φ is $\ulcorner(\forall x)\ \psi\urcorner$, then $[\![\varphi]\!]^{F,g} = 1$ iff $[\![\psi]\!]^{F,g[^x/_u]} = 1$, for any $u \in D_a$.

Negation is a function of type *tt* and as such characterizes a certain set of truth values, viz. {0}. Since this set is a singleton, a definition of its member would immediately lead to a definition of the whole set. So we need a definition of the truth-value 0, i.e. a Ty1-expression of category *t* that always comes out as false. Such a formula would have to express something that we know to be false of every ontology like, e.g., that it does not contain any individual, i.e. that $D_e = \varnothing$. However, if we try to express this equation in Ty1, it seems that the right hand-side requires us to give a definition of Ø and hence of the truth-value 0, because Ø is the function taking everything to 0. So this strategy does not seem to lead anywhere. What else do we know to be false of every ontology? Since D_t is always {0,1}, know that, e.g., none of the following equations holds:

(10) (a) $D_t = \varnothing$
(b) $D_t = \{0\}$
(c) $D_t = \{1\}$

Now trying to express any of (10) (a) of (b) seems to lead to the circularity problem just encountered: it looks like we would have to find a definition of 0 before we can formulate these equations. (10c) is different. For it only requires a definition ⊤ of the truth-value 1, from which we could then obtain a definition of the singleton: [λx_t x = ⊤]. But we already got ⊤ and thus get the following reduction of negation:

(11) '¬' := '[λy_t (y = ([λx_t (x = x)] = [λx_t x = (x = x)]))]'

Of course, *negating* a formula φ∈ $\mathrm{Ty1}_t$ then amounts to combining '¬' and φ by Application: ⌜¬(φ)⌝. In order to be in accordance with usual notational conventions, we will in this case omit the brackets.

We finally have to reduce conjunction to the expressive means of Ty1. We have seen that conjunction can be thought of as a function of type *t*(*tt*) characterizing the binary relation {(1,1)} among truth values, i.e. that relation that holds among truth-values *u* and *v* iff (*u*,*v*) = (1,1). Again, it looks like we just have to define a singleton, but this time our old trick combining Abstraction, Identity, and a definition of the member won't work because the member (1,1) itself is not an element of the ontology: the latter only contains *sets* of ordered pairs but no ordered pairs by themselves. But there is a way around this difficulty. For instead of directly expressing the equation:

(12) (*u*,*v*) = (1,1)

we can reformulate it using a version of *Leibniz's Law* that says that *w* and *w*' are identical if and only if they share all their properties, i.e. if every set containing *w* also contains *w*'. This law is obviously true because one of the sets containing *w* is {*w*}, which only contains *w*' if *w*' happens to be identical to *w*. Applying Leibniz's Law to (12) results in the equivalent condition:

(12') For all M: (*u*,*v*) ∈ M iff (1,1) ∈ M.

Clearly, one can concentrate on such M that only contain pairs of truth-values; the (12') would still be equivalent to (12). But the M containing pairs of truth-values are precisely the binary relations among them, i.e. the objects of type *t*(*tt*). Since we already know how to universally quantify over arbitrary objects of the ontology and, moreover the material equivalence is just Identity applied to truth-values (!), (12') is indeed expressible in Ty1:

(13) $(\forall R_{t(tt)})$ (R(y)(x) = R(⊤)(⊤)),

where '⊤' is some trivial equation again. (13) still contains two free variables corresponding to the arbitrary truth-values we are going apply it to. It should be clear by now that these have to be bound by Abstraction if we want to obtain the general definition of conjunction as a function of type *t*(*tt*):

(13') '&' := '$[\lambda x_t\ [\lambda y_t(\forall R_{t(tt)})$ (R(y)(x) = R(⊤)(⊤))]]'

Conjoining two formulae of type *t* amounts to iterated uses of Application; we will thus take '[φ & ψ]' to be an alternative notation for '&(ψ)(φ)'. The above reasoning made clear that we get the usual truth-conditions:

(&) $[\![[\varphi \,\&\, \psi]]\!]^{F,g} = 1$ iff $[\![\varphi]\!]^{F,g} = [\![\psi]\!]^{F,g} = 1$.

The other connectives and the existential quantifier can now be defined in the usual way. We will from now on freely use them in our Ty1-formulae.

We have seen that, in spite of its low number of somewhat unfamiliar semantic operations, Ty1 does have at least the expressive power of predicate logic. In fact, since quantification is not restricted to individuals, Ty1 turns out to include second (third, ...) order logic as well. Thus has some possibly unwelcome side-effects on the computational complexity of the logical relations holding among Ty1-expressions. Before we can (briefly) discuss this, we better get familiar with some technical tools that will allow us to turn complicated λ-expressions into more readable formulae.

4.4 Logical reductions

Ty1-translations encountered in indirect interpretation procedures tend to be very long and messy. As the above discussion about the compositional translation of quantified noun phrases and determiners already revealed, even a simple sentence like

(14) ***Every student is asleep.***

will have to be translated by something like:

(14') $[\lambda Y_{et}\ [\lambda X_{et}\ (\forall x_e)\ [Y(x) \rightarrow X(x)]\ ]\ ]\ (S)\ (A)$.

And, as we will soon see, the slightly more complicated

(15) ***Every student reads some book.***

will come out as the almost unreadable:

(15') $[\lambda Y_{et}\ [\lambda X_{et}\ (\forall x_e)\ [Y(x) \rightarrow X(x)]\ ]\ ]\ (S)$
$(\ [\ \lambda x_e\ [\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]\ (B)\ (\ [\lambda y_e\ R(y)(x)]\ )\ ]\ )$.

Even at a second glance, it might not be clear that (15') is a Ty1-expression at all – let alone a sentence corresponding to (15). (The fact that it is a category *t*-expression will have to be proved in an exercise!) Yet (15') says no more and no less than the more familiar formula (15") of predicate logic:

(15") $(\forall x)\ [S(x) \rightarrow (\exists y)\ [B(x)\ \&\ R(x,y)]\ ]$

Similarly, (14') amounts to the first-order formula:

(14")$(\forall x)\ [S(x) \rightarrow A(x)]$

We have just seen that first-order formulae like (14") and (15") can be thought of as special cases of $\mathrm{Ty1}_t$-expressions, which puts us in a position to define the relationship that holds between (14') and (14") and between (15') and (15") as one of logical equivalence: any Ty1-expressions α and β of the same category are said to be *logically equivalent* if and only if $[\![\alpha]\!]^{F,g} = [\![\beta]\!]^{F,g}$ for all models $((D_a)_{a\in T}, F, g)$. Why should (14') and (14") be logically equivalent? One reason is, of course, that we construed (14') in a way that should guarantee this equivalence. In order to show that we have actually succeeded, we could argue as follows: the translation of the determiner ***every*** denotes a function $f_{\textbf{\textit{every}}}$ that assigns the extension of

(16) $[\lambda X_{et}\ (\forall x_e)\ [Y(x) \rightarrow X(x)]\ ]$

to arbitrary sets $U \in D_{et}$ that serve as a possible interpretations of Y. In

(14') we get a specific value for Y, viz. the denotation M of $S \in Ty1_{et}$, the translation of ***student***. Thus, in (14'), the argument U of $f_{\mathbf{every}}$ is not arbitrary, but coincides with the set M. So instead of pretending that the variable Y refers to M, we may as well choose an expression that we know to refer to M, viz. S. Instead of (16) we thus get:

(16') $[\lambda X_{et}\ (\forall x_e)\ [S(x) \rightarrow X(x)]\ ]$,

which we obtain from (16) by simply *replacing* Y by S. But (16) denoted the result of applying $f_{\mathbf{every}}$ to M, i.e. the extension of:

(16") $[\lambda Y_{et}\ [\lambda X_{et}\ (\forall x_e)\ [Y(x) \rightarrow X(x)]\ ]\ ]\ (S)$

Since (16') and (16") refer to the same function, we may replace the latter's occurrence in (14') by the former, thus obtaining the shorter:

(14^*) $[\lambda X_{et}\ (\forall x_e)\ [S(x) \rightarrow X(x)]\ ]\ (A)$

(Note that we cannot do the same trick with (16) instead of (16'), since we do not know whether the free occurrence of Y actually refers to M!) A similar reasoning then shows that (14^*) and, consequently, (14') have the same extension as the first-order formula (14"). Since we did not make any specific assumptions about the model with respect to which the extensions of these formulae are to be determined, the argument generalizes to all models and hence establishes the logical equivalence of (14') and (14^*).

The above reasoning is admittedly sketchy but it contains all ingredients of a precise proof into which it can be turned easily. Instead of indulging into this formal exercise, we will try to find the more general principles underlying this specific argument. In order to find them, it suffices to concentrate on the equivalence of (14') and (14^*), which was established in two steps: we *first* argued that the sub-expression (16") of (14') has the same extension as the shorter (16') and *then* used this observation to replace the former by the latter. The second step seems to reflect a very general principle to the effect that substitution of an expression α by a β with the same extension never changes the extension of the *host*, i.e. the γ in which α originally occurred. A little care must be taken, though, if one wants to give a precise account of this *Substitution Principle*. First of all, one must realize that, in case that α is a variable, only its *free* occurrences should be replaced by extensionally equivalent β:

(17) $(\forall x)\ P(x)$

does not necessarily have the same truth value as

(17') $(\forall x)\ P(y)$,

even if the variables x and y happen to denote the same individual *u*: (17') will already be true if P's extension is {*u*} but (17) will be false – unless {*u*} is the universe. (Of course, *renaming* the bound variable, i.e. replacing 'x' by 'y' in both the quantifier prefix and the formula *does* work in this case – but that's quite a different story to be told later.) The same problem arises if α *contains* a variable that is bound, not in α itself, but by an operator in the host: we should not replace the second occurrence 'f(x)' in (18) by 'y', even if the two terms denoted the same object:

(18) $[Q(f(x))\ \&\ (\forall x)\ P(f(x))\ ]$

But the first 'f(x)' is outside the scope of the quantifier, and there is nothing wrong with replacing it by 'y', thereby obtaining:

(18') $[Q(y)\ \&\ (\forall x)\ P(f(x))\ ]$

which has the same truth-value, if indeed 'y' and 'f(x)' are co-extensional. So the notion of substitution must be one of *substituting free occurrences*. It is possible (and not even too difficult) to define such a notion by recursion on the host expression but we will just assume that we know what we mean by 'substituting all free occurrences of α in γ by β'; and we will write 'γ $[^{\alpha}/_{\beta}]$' for the result. (This is the same notation that we used for modifying variable assignments; needless to say the two should not be confused.) Care must also be taken if β happens to be or only contain a free variable. For even if, say, 'f(x)(y)' and 'z' happen to denote the same object, the formula

(19) $(\forall x)\ (\exists y)\ (\ z = f(x)(y)\ )$

need not be true, but (19') certainly is, even though (19') = (19)$[^{z}/_{f(x)(y)}]$:

(19') $(\forall x)\,(\exists y)\,(\, f(x)(y) = f(x)(y)\,)$

What goes wrong is the fact that substitution has turned the free variables x and y into bound variables that no longer get their extension from the variable assignment. Substitution must thus be restricted to the case where the β to be inserted does not contain any free variable that would be bound if we replaced α by β. In other words, α must not occur in γ within the scope of an operator that binds a variable occurring freely in β. This condition is usually expressed as: α *is free for* β *in* γ. Again, this condition could be given a rigorous recursive definition but we do not need do this here. We are finally in a position to formulate the second principle underlying our reduction of (14') to (14"):

Substitution Principle

Let $((D_a)_{a\in T}, F, g)$ be a model and α, β, and γ be Ty1-expressions such that $[\![\alpha]\!]^{F,g} = [\![\beta]\!]^{F,g}$ and α is free for β in γ. Then $[\![\gamma]\!]^{F,g} = [\![\gamma\,[^{\alpha}/_{\beta}]]\!]^{F,g}$.

Given the appropriate definitions of the background concepts, the Substitution Principle can be proved by induction on γ's complexity. We will only sketch the proof. In order to get the induction off the ground, we must show that any lexical γ (i.e. variable or constant) satisfies the principle. There are two cases to be distinguished according to whether γ happens to be the same expression as α: if so, replacing α (which would be free) by β would result in β, i.e. $\gamma\,[^{\alpha}/_{\beta}] = \beta$, and consequently $[\![\alpha]\!]^{F,g} = [\![\beta]\!]^{F,g}$ trivially implies $[\![\gamma]\!]^{F,g} = [\![\alpha]\!]^{F,g} = [\![\beta]\!]^{F,g} = [\![\gamma\,[^{\alpha}/_{\beta}]]\!]^{F,g}$; if, on the other hand, α is distinct from γ, then it does not occur in it (because γ is lexical) and thus replacing it has no no effect on γ so that $\gamma = \gamma\,[^{\alpha}/_{\beta}]$, and thus $[\![\gamma]\!]^{F,g} = [\![\gamma\,[^{\alpha}/_{\beta}]]\!]^{F,g}$. In the inductive step we may assume that we know the Substitution Principle to hold for the parts of the complex expressions for which we want to prove it. Thus, if γ is of the form $\ulcorner\gamma_1(\gamma_2)\urcorner$ and (*) γ_1 and γ_2 satisfy the Substitution Principle, we must show that $[\![\gamma_1(\gamma_2)]\!]^{F,g} = [\![\gamma_1(\gamma_2)\,[^{\alpha}/_{\beta}]]\!]^{F,g}$. But this is easily established, since the latter is $[\![\gamma_1[^{\alpha}/_{\beta}]\,(\gamma_2[^{\alpha}/_{\beta}])]\!]^{F,g}$, i.e. $[\![\gamma_1[^{\alpha}/_{\beta}]]\!]^{F,g}\,([\![\gamma_2[^{\alpha}/_{\beta}]]\!]^{F,g})$ to which we can apply the inductive hypothesis (*) giving us: $[\![\gamma_1]\!]^{F,g}\,([\![\gamma_2]\!]^{F,g})$, i.e. $[\![\gamma_1(\gamma_2)]\!]^{F,g}$. The case where γ is $\ulcorner(\gamma_1 = \gamma_2)\urcorner$ is similar. Abstraction is a little harder. If γ is of the form $\ulcorner[\lambda x\, \gamma_1]\urcorner$ and α happens to be the variable x itself nothing is to be proved, because α (i.e. x) does not occur freely in $\ulcorner[\lambda x\, \gamma_1]\urcorner$ and so replacing its free occurrences once more does not have any effect: $\ulcorner[\lambda x\, \gamma_1]\, [{}^{\alpha}/_{\beta}]\urcorner = \ulcorner[\lambda x\, \gamma_1]\, [{}^{x}/_{\beta}]\urcorner = \ulcorner[\lambda x\, \gamma_1]\urcorner$. And even if α is not the variable x, there is still a possibility that $\ulcorner[\lambda x\, \gamma_1]\, [{}^{\alpha}/_{\beta}]\urcorner = \ulcorner[\lambda x\, \gamma_1]\urcorner$; for α might not occur freely in γ_1 ('vacuous substitution'), and again nothing would have to be proved. But if α does occur freely in γ_1, then x cannot be free in either α or β: if x were free in β, it would get bound after replacing α by β, and thus α would not be free for β in γ; and x cannot have any free occurrences in α, because otherwise the latter would not occur freely in γ (= $[\lambda \underline{x}\, \gamma_1]$!). In particular, then, it does not matter to α or β what g assigns to x and our assumption $[\![\alpha]\!]^{F,g} = [\![\beta]\!]^{F,g}$ implies $[\![\beta]\!]^{F,g[{}^{x}/_{u}]} = [\![\beta]\!]^{F,g[{}^{x}/_{u}]}$, no matter which *u* (of the appropriate type) we take. (Or so we will assume; a proof of this elementary fact, known as the *Coincidence Lemma*, would actually require a separate induction!) But then $[\![\gamma_1]\!]^{F,g[{}^{x}/_{u}]} = [\![\gamma_1[{}^{\alpha}/_{\beta}]]\!]^{F,g[{}^{x}/_{u}]}$ (for any *u*) because, according to the induction, γ_1 satisfies the Substitution Principle. We thus have: $[\![[\lambda x\, \gamma_1]]\!]^{F,g}\, (u) = [\![[\lambda x\, \gamma_1[{}^{\alpha}/_{\beta}]]]\!]^{F,g}\, (u)$, for arbitrary *u*. Since α does not contain free x, $[\lambda x\, \gamma_1[{}^{\alpha}/_{\beta}]] = [\lambda x\, \gamma_1]\, [{}^{\alpha}/_{\beta}]$, and so we finally conclude that $[\![[\lambda x\, \gamma_1]]\!]^{F,g} = [\![[\lambda x\, \gamma_1][{}^{\alpha}/_{\beta}]]\!]^{F,g}$.

Two features about the above proof are worth emphasizing. One is that it implicitly contains a definition of the operation of substituting the free occurrences of one expression by another expression. Secondly, those who feel they get lost in it should be aware of the fact that it would be longer had we included more basic means of expressions in Ty1.

The Substitution Principle as we have formulated it above says something about substituting *denotationally equivalent* formulae. It is thus much stronger than the fact that substitution of logically equivalent expressions again leads to logically equivalent expressions: the latter follows directly from the compositional nature of the interpretation of Ty1.

The Substitution Principle will help us to establish the other principle we are interested in, the one responsible for the equivalence of (16") and

(16'). The idea behind the above argument is simple (and general) enough: an expression of the form $\ulcorner[\lambda x\ \alpha]\ (\beta)\urcorner$ denotes the result of applying a function f that takes any u to whatever α's extension would be if x denoted u; so applying f to a particular u that is actually denoted by β should give us the extension of $\alpha\ [^{x}/_{\beta}]$. Since the argument again involves substitution, we should not be too surprised that the same restrictions apply as in the above principle: only free occurrences of x are supposed to be replaced, and β should not contain any variables that would get bound by the substitution. (The matter will be further pursued in an exercise.) But given these restrictions the reduction principle is perfectly general:

<u>*The Principle of λ-Conversion*</u>
Let $\ulcorner[\lambda x\ \alpha]\ (\beta)\urcorner$ be a Ty1-expression such that x is free for β in α. Then $\ulcorner[\lambda x\ \alpha]\ (\beta)\urcorner$ is logically equivalent to $\alpha\ [^{x}/_{\beta}]$.

Again we will only sketch a proof. Given a model $((D_a)_{a\in T}, F, g)$, the semantic rules of Ty1 tell us that $\llbracket[\lambda x\ \alpha]\ (\beta)\rrbracket^{F,g} = \llbracket\alpha\rrbracket^{F,g[^{x}/_{u}]}$, where $u = \llbracket\beta\rrbracket^{F,g}$. So it remains to be shown that:

(!) $\llbracket\alpha\rrbracket^{F,g[^{x}/_{u}]} = \llbracket\alpha\ [^{x}/_{\beta}]\rrbracket^{F,g}$.

We cannot immediately apply the Substitution Principle, because all we have got is: $\llbracket x\rrbracket^{F,g[^{x}/_{u}]} = u = \llbracket\beta\rrbracket^{F,g}$, which is not enough because it involves two different models. (It would suffice if either $g(x) = u$ or β did not contain x; but λ-conversion is much more general.) However, a little trick (inspired by the substitutional interpretation of Abstraction!) will help: we will pick a 'neutral' name for u, i.e. one that is not affected by the difference between the two models. So let z be a variable (of the same type as x) that occurs neither in α nor in β. In particular, then, $\llbracket\beta\rrbracket^{F,g[^{z}/_{u}]} = \llbracket\beta\rrbracket^{F,g} = u$; and, clearly, $\llbracket z\rrbracket^{F,g[^{z}/_{u}]} = u$. We can thus apply the Substitution Principle, using $\alpha\ [^{x}/_{z}]$ as a host:

(*) $\llbracket\alpha\ [^{x}/_{z}]\rrbracket^{F,g[^{z}/_{u}]} = \llbracket\alpha\ [^{x}/_{z}]\ [^{z}/_{\beta}]\rrbracket^{F,g[^{z}/_{u}]}$

Of course, we must check that the Substitution Principle is actually applicable, but there is no problem about that: the places where z occurs in $\alpha\,[^{x}/_{z}]$ are those where x occurs in α (because z was 'new') and they are safe places for β's free variables, because x is free for β in α. The left hand-side of (*) is just $[\![\alpha]\!]^{F,g[^{x}/_{u}]}$: it makes no difference whether we use x or z as a name for *u*. (Again, this can be seen as a consequence of a version of the Coincidence Lemma.) But performing the substitutions on the Ty1-expression mentioned on the right side of (*) leads to $[\![\alpha\,[^{x}/_{\beta}]]\!]^{F,g[^{z}/_{u}]}$, i.e. $[\![\alpha\,[^{x}/_{\beta}]]\!]^{F,g}$, because z does not occur in it. We have thus established (*) and with it the famous Principle of λ-Conversion.

The equivalence between our alternative definitions (7") and (9) of the universal quantifier can now be seen to be an instance of λ-conversion:

(7") $(\,[\lambda x_{a}\ \varphi] = [\lambda x_{a}\ (x = x)]\,)$

(9) $[\lambda P_{at}\ (P = [\lambda x_{a}\ (x = x)]\,)\,]\ ([\lambda x_{a}\ \varphi])$

(9) is of the form $\ulcorner[\lambda P_{at}\ \alpha\,]\ (\beta)\urcorner$ and (7") is $\alpha\,[^{P}/_{\beta}]$. Moreover, P is free for β (= $\ulcorner[\lambda x_{a}\ \varphi]\urcorner$) in α (= $\ulcorner(P = [\lambda x_{a}\ (x = x)]\,)\urcorner$), because it does not occur within the scope of any λ-operator. Note that the fact that P is itself bound within the functor $\ulcorner[\lambda P_{at}\ \alpha\,]\urcorner$ is quite immaterial for λ-conversion.

Our formulation of the Principle of λ-Conversion might create the impression that it is not always applicable and that consequently not every complex Ty1-expression involving application of a λ-term can be reduced by it. That this impression is at least misleading will become apparent once we have seen two other principles of reduction the first of which is motivated by the so-called *extensionality* of functions, i.e. the (set-theoretic) fact that a function *f* can be exhaustively described by its 'course of values', i.e. by stating, for each argument *u* of *f*, which value *f* assigns to *u*. If we think of an expression $\ulcorner[\lambda x\ \alpha(x)]\urcorner$ as such a description of a function in terms of what ('α(x)') it assigns to any argument ('x'), extensionality amounts to stating that $\ulcorner[\lambda x\ \alpha(x)]\urcorner$ denotes the same function as α itself. So we expect another general principle to hold:

The Principle of η-Conversion

Let $\alpha \in Ty_{ab}$ be a Ty1-expression without free occurrences of $x \in Var_{a}$. Then $\ulcorner[\lambda x\ \alpha(x)]\urcorner$ is logically equivalent to α.

('η' is for 'extensionality'.) The principle is easily established. Given an arbitrary model $((D_a)_{a\in T}$, F, g) and some $u\in D_a$, we must show that $[\![[\lambda x\ \alpha(x)]]\!]^{F,g}(u) = [\![\alpha]\!]^{F,g}(u)$. (That this suffices is actually due to our extensional, set-theoretic notion of a function!) But the semantic rules of Ty1 tell us that:

$$
\begin{aligned}
& [\![[\lambda x\ \alpha(x)]]\!]^{F,g}(u) \\
= \quad & [\![\alpha(x)]\!]^{F,g[^x/_u]} \\
= \quad & [\![\alpha]\!]^{F,g[^x/_u]}([\![x]\!]^{F,g[^x/_u]}) \\
= \quad & [\![\alpha]\!]^{F,g[^x/_u]}(u).
\end{aligned}
$$

But since x does not occur freely in α, $[\![\alpha]\!]^{F,g[^x/_u]} = [\![\alpha]\!]^{F,g}$, and we're home.

It is worth pointing out that the condition on x's non-occurrence in α is essential in that it blocks applications to formulae like [λx [λy (y = x)] (x)] where it would lead to undesired results (as should have become clear from one of the homework exercises).

The principles of λ-conversion and η-conversion can be combined to prove a very fundamental fact about renaming bound variables, one that should already be familiar from predicate logic:

The Principle of Alphabetic Variants

Let α be a Ty1-expression such that (i) $y\in Var_a$ does not occur freely in α and (ii) x is free for y in α. Then $\ulcorner[\lambda x\ \alpha]\urcorner$ and $\ulcorner[\lambda y\ \alpha\ [^x/_y]]\urcorner$ are logically equivalent.

(Note that (i) and (ii) are both met if y does not occur in α at all, whether free, bound, or in a λ-prefix!) Two formulae that are equivalent due to the above principle are called *alphabetic variants* of each other. The content and truth of the principle should be intuitively obvious: the precise choice of a bound variable is immaterial. And, with what we have already got, the proof is quite simple: given a model $((D_a)_{a\in T}$, F, g), we find that $[\![[\lambda x\ \alpha]]\!]^{F,g} = [\![[\lambda y\ [\lambda x\ \alpha]\ (y)\ ]]\!]^{F,g}$, by η-conversion, and then λ-conversion gives us the desired result.

Since the quantifiers were all defined in terms of Abstraction, the above principle also carries over to the variables they bound. Thus, e.g., '$(\forall x_e)\ P_{et}(x)$' is equivalent to '$(\forall y_e)\ P_{et}(y)$', etc. But this should be clear anyhow. The reason why we are interested in the renaming of bound variables is that, in a sense, it enables us to apply λ-conversion whenever we have a formula of the form $\ulcorner[\lambda x\ \alpha]\ (\beta)\urcorner$, whether x is free for α or not. For if not, some λ in α (with an occurrence of x in its scope) would bind a variable y that is free in β. But if we rename this bound variable with some z that occurs nowhere else in α and is not free in β, the λ would not do any harm to β's free ys anymore. And if there is more that one such binding of originally free variables, we can go through the procedure again, until the resulting formula meets the criteria of λ-conversion. This strategy of renaming bound variables before applying λ-conversion will be called *critical λ-conversion*. Instead of defining this complicated procedure in detail, we will look at it in one specific case and then understand how it works in general. The example will be a formula we have already mentioned, viz. the translation (15') of ***every student reads some book***, here repeated as:

(20) $[\lambda Y_{et}\ [\lambda X_{et}\ (\forall x_e)\ [Y(x) \rightarrow X(x)]\ ]\ ]\ (S)$
$(\ [\ \lambda x_e\ [\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]\ (B)\ (\ [\lambda y_e\ R(y)(x)]\ )\ ]\ )$

The overall structure of this expression is $\ulcorner[\lambda Y\ \alpha]\ (\beta)\ (\gamma)\urcorner$, where β is the constant S of type *et*. Since S does not contain any variable, $\ulcorner[\lambda Y\ \alpha]\ (\beta)\urcorner$ is logically equivalent to $\alpha[^{Y}/_{\beta}]$, so that (20) becomes:

(20') $[\lambda X_{et}\ (\forall x_e)\ [S(x) \rightarrow X(x)]\ ]$
$(\ [\ \lambda x_e\ [\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]\ (B)\ (\ [\lambda y_e\ R(y)(x)]\ )\ ]\ )$

The argument does not contain any free variables, so X is free for it and λ-conversion turns (20') into (20"), where the original argument has been underlined for better readability:

(20") $(\forall x_e)\ [S(x) \rightarrow$
$\underline{[\ \lambda x_e\ [\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]\ (B)\ (\ [\lambda y_e\ R(y)(x)]\ )\ ]}\,(x)]$

Since the underlined functor is of the form $\ulcorner[\lambda x\ \delta]\urcorner$, we would like to apply λ-conversion – which is no problem because its argument only contains x as a free variable and x certainly cannot be bound if we 'substitute' it for x wherever x is *free* in δ! (Note that this is quite generally so; hence any expression of the form $\ulcorner[\lambda x\ \delta]\ (x)\urcorner$ is logically equivalent to δ itself.) So we get:

(20''') $(\forall x_e)\ [S(x) \rightarrow$
$[\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]\ (B)\ (\ [\lambda y_e\ R(y)(x)]\ )\ ]$

Once more we can perform an unproblematic λ-conversion because B is a constant not containing any endangered variables:

(21) $(\forall x_e)\ [S(x) \rightarrow [\lambda X_{et}\ (\exists x_e)\ [B(x)\ \&\ X(x)]\ ]\ (\ [\lambda y_e\ R(y)(x)]\ )\ ]$

Now we would like to further reduce (21) by applying λ-conversion to its sub-expression:

(22) $[\lambda X_{et}\ (\exists x_e)\ [B(x)\ \&\ \underline{X}(x)]\ ]\ (\ [\lambda y_e\ R(y)(x)]\ ),$

which, of course, we are not supposed to do: the argument contains a free x and replacing it for the underlined occurrence of X in (22) would get it into the scope of the existential quantifier. (Note that the fact that x is bound *in (21)* does not help: it is the 'local' property of being free *in the argument* that counts.) So we must apply critical λ-conversion and first rename the bound variable causing trouble. We may pick some 'new' variable z, but note that y would also do because it is bound in β and does not occur in α. We would then get:

(21') $(\forall x_e)\ [S(x) \rightarrow [\lambda X_{et}\ (\exists y_e)\ [B(y)\ \&\ X(y)]\ ]\ (\ [\lambda y_e\ R(y)(x)]\ )\ ]$

to which we can apply λ-conversion, thus obtaining:

(22') $(\forall x_e)\ [S(x) \rightarrow (\exists y_e)\ [B(y)\ \&\ [\lambda y_e\ R(y)(x)]\ (y)]\ ],$

which can be turned into the desired first-order formula by one more trivial λ-conversion:

(15'') $(\forall x)\ [S(x) \rightarrow (\exists y)\ [B(y)\ \&\ R(x,y)]\ ]$

(We regard 'R(y)(x)' and 'R(x,y)' as notational alternatives.) Apart from illustrating the strategy of critical λ-conversion, the above example also shows how important the reduction of Ty1-formulae by λ-conversion in general is: in order to turn (20) into the shorter and more readable (15"), we had to perform no less than six λ-conversions, one of which was critical. We will see in the examples further below that this situation is by no means exceptional.

Our reduction of (20) to (15") can be used to illustrate an important feature of λ-conversion, the so-called *diamond-property*. Since λ-conversion turns one Ty1-formula into a logical equivalent one, it may be used to replace arbitrary sub-expressions $\ulcorner[\lambda\ x\ \alpha]\ (\beta)\urcorner$ of larger formulae by the corresponding $\alpha\ [^x/_y]$, just as we have done, e.g., in getting from (20") to. (20"') above where we did not apply λ-conversion to the whole formula but only to one of its parts. We could even have applied such an 'inner' λ-conversion to our starting point (20) itself: instead of trading in the top-level 'λY' for the argument 'S' in (20), we might as well have reduced the 'λY-term' (corresponding to the noun phrase ***some book***) by inserting 'B' for 'Y' (and dropping the λ-prefix). The Principle of λ-Conversion tells us that the result would have been equivalent to(2). Similarly, we could have eliminated this second 'λY' immediately after the first one; and we would have had the corresponding choice of either getting rid of the other 'λY' or of the second 'λS' if we had chosen to reduce (20) by inserting'B' for the second 'λY', in the way just indicated. Some of these reductions might not seem very orderly but they are certainly licensed by λ-conversion. And, as one easily sees, there are lots of such alternative ways of ultimately reducing (20) until λ-conversion cannot be applied anymore. (Actually this point might not have been reached yet with (15"); this depends on our precise 'implementation' of the connectives and quantifiers. We will ignore this complication.) An interesting fact about all these alternative reductions is that they all lead to essentially the same result, as one could verify by trying them out. ('essentially', because we may get alphabetic variants of (15").) And this is no coincidence but reflects a general feature of reduction by λ-conversion. In order to formulate it properly we will say that a Ty1-formula α *λ-reduces to* β iff successive application of (possibly critical) λ-conversion eventually turns it into β to which no more λ-conversions can be applied, i.e. β does not contain any part of the form $\ulcorner[\lambda\ x\ \gamma]\ (\delta)\urcorner$. We can now state a following result:

Strong Normalization Theorem (for Ty1)

λ-reduction in Ty1 has the diamond property, i.e. if any Ty1-formula α λ-reduces to both β and β', then β and β' are alphabetic variants of each other.

Our main interest in the above theorem (the proof of which is far beyond the present course) lies in the fact that it allows us to perform λ-reductions in whatever way we like: the ultimate result will always be the same. In particular, then, we may apply λ-conversions as early as possible, i.e. to sub-expressions whose translations we have determined without already knowing the translation of the whole sentence. So if we know that a determiner *D* translates into $\ulcorner[\lambda X\ \alpha]\urcorner$ and that the noun *N* it is combined with translates as β, then we may safely apply (critical) λ-conversion to $\ulcorner[\lambda X\ \alpha]\urcorner$ and β, regarding the result as the translation of the noun phrase $\ulcorner D\ N\urcorner$: it is logically equivalent to the 'official' translation $\ulcorner\alpha(\beta)\urcorner$ (by λ-conversion) and the fact that we already performed the conversion will not interfere with or preclude any λ-reductions we might want to perform on the translation of the sentence in which the NP occurs. (We also need to know something about the entire translation procedure: it must be *compositional* , in a sense to be explored in section 4.7.) This strategy of early reduction will help us keep our translations as short and easily readable as possible.

Before we will take a brief look at other ways of reducing Ty1-formulae, it should be mentioned that, strictly speaking, λ-reduction does not necessarily lead to *shorter* formulae. For it may well happen that the argument to be inserted for the λ-bound variable x is rather long and that the λ binds more than one occurrence of x. In such a case, λ-conversion is bound to lead to a longer formula than the original one. If moreover the result of this conversion cannot be λ-reduced any further, we have 'reduced' one formula to a more complex one. The reduction of (23) to (23') is such a case:

(23) $[\lambda x_t\ (\ (x = x) = (x = x)\ )]\ (\ ([\lambda y_e\ y] = [\lambda y_e\ y])\ )$

(23') $(\ (([\lambda y_e\ y] = [\lambda y_e\ y]) = ([\lambda y_e\ y] = [\lambda y_e\ y])) =$
$(([\lambda y_e\ y] = [\lambda y_e\ y]) = ([\lambda y_e\ y] = [\lambda y_e\ y]))\ )$

Fortunately, this kind of situation is not typical for the process of indirect interpretation: the translations of natural language expressions tend to become less complex with every λ-conversion, one reason being that, for the most part, every λ-operator binds exactly one variable, a remarkable fact to which we will return soon.

Since Ty1 contains all of predicate logic as a 'sub-language', one would expect their logical laws to be valid here, too. This is indeed the case. In particular, any two formulae of predicate logic (with identity) are logically equivalent in Ty1 iff each logically implies the other in the sense of predicate logic. So all kinds of ordinary logical equivalences can be applied to Ty1-formulae and hence to the translations of natural language sentences. Thus $\ulcorner\neg\neg\varphi\urcorner$ always reduces to (i.e. is logically equivalent with) φ itself, $\ulcorner\neg[\neg\varphi \vee \neg\psi]\urcorner$ amounts to $\ulcorner[\varphi \ \& \ \psi]\urcorner$, $\ulcorner[(\forall x)\varphi \ \& \ (\forall x)\psi]\urcorner$ becomes $\ulcorner(\forall x) \ [\varphi \ \& \ \psi]\urcorner$ and $\ulcorner(\forall x) \ [P(x) \rightarrow (\exists y) \ [Q(y) \ \& \ (x = y)] \ ]\urcorner$ is the same as $\ulcorner(\forall x) \ [P(x) \rightarrow Q(x) \ ]\urcorner$, etc. We will freely use these kinds of equivalences in the examples to be discussed below.

It would certainly be nice if we had some procedure for reducing a Ty1-formula as much as possible, i.e. to the shortest formula it is equivalent to. Unfortunately, such a procedure does not exist and never will. (This does not mean that the shortest reduced form does not exist; in a sense, it always does.) For if applied to any to any first-order formula φ it would have to give us something like '$x = x$' whenever φ is valid, thus being a *decision procedure* for (the notion of validity in) first-order logic. Such a thing is, however, impossible as was shown by Alonzo Church in 1936. So whichever reductions we use, they may be very helpful for a better understanding of long formulae and they may very often lead to the shortest equivalent formulae possible – but they are always far away from being perfect. A similar remark applies to all attempts to axiomatize all possible reductions. Although the fact that Ty1 contains first-order logic does not contradict this possibility, the fact that it also includes second-order logic does: such an axiomatization would amount to an enumeration of all valid second-order formulae, which again can be shown to be impossible.

4.5 A fragment

We will now apply some of the above ideas to the construction of a simplified Montagovian algorithm for compositionally translating the syntactic structures of a trivial fragment of English into formulae of Ty1.

The algorithm consists of two parts, corresponding to the two parts of a compositional interpretation: a *lexical* part providing the translations of the lexical expressions and an *operational* part that tells us how to translate complex expressions. It has already been mentioned that most lexical expressions will be translated by Ty1-constants of the corresponding type. For the presentation of the translation algorithm, we will thus rely on the type assignment (2) of part 3, extending it so as to include some more categories: if not otherwise stated, the translation of a lexical expression of a category K will be an element of Con_a, where a is the type corresponding to K; moreover we will assume that different lexical expressions are translated by different constants. In order to refer to these constants, we will use the kind of mnemonic notation familiar from predicate logic formalization and the above discussion: 'M' ($\in \mathrm{Con}_{et}$) translates ***man***, 'S' might stand for the translation of ***sell*** ($\in \mathrm{Con}_{e(et)}$) or for the translation of ***student*** ($\in \mathrm{Con}_{et}$) and context will help determine which one. Sometimes we also refer to the translation of a lexical expression by priming. We may thus say that ***student***' $\in \mathrm{Con}_{et}$, but the same is not true of ***every***'. Indeed we have:

(24) ***every***' $:= [\lambda Y_{et}\, [\lambda X_{et}\, (\forall x_e)\, [Y(x) \rightarrow X(x)]\,]\,]$

(25) ***some***' $:= [\lambda Y_{et}\, [\lambda X_{et}\, (\exists x_e)\, [Y(x)\ \&\ X(x)]\,]\,]$

These two equations are instances of *lexical decomposition*, which applies whenever the translation of a lexical item is itself a complex expression. In (24) and (25) the two determiners in question get completely analyzed in terms of logical material. This is not the typical or classical case of decomposition, that gives the meaning of one expression in terms of the meanings of other expressions, as in:

(26) ***girl***' $:= [\lambda x_e\, [F(x)\ \&\ P(x)\ \&\ C(x)]\,]$

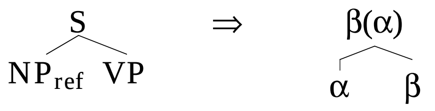
(Note that we have conjoined more than one formula by '&', because that way the formula is more readable, and bracketing does not disambiguate in this case.) The three predicates in (26) are supposed to be constants of type *et*: 'F' stand for 'female', 'P' for 'person', and 'C' for 'child' (in the non-relational sense). So the three constants may themselves be thought of translations of the lexical items ***female***, ***person***, and ***child***, respectively. (On the other hand, they may as well correspond to some 'pure' concepts that are close to, but not identical with, the true meanings of those three expressions.) Since ***female*** is an adjective, we have thus implicitly extended our type assignment to the category Adj by letting it correspond to the same type as VP and N, i.e. the type *et* of sets of individuals. This is in accord with ordinary logic textbook analysis.

The operational part of the translation algorithm gives the translations of complex expressions (or their structures) in terms of the syntactic operations they have been built by. How this is done can be seen from a simple example; the general framework will be discussed in section 4.7. We assume we have a construction *F* (like the corresponding context-free rule) that combines a referential (subject) noun phrases and VPs into sentences. We then have the following *translation rule*:

(T_F) If A is a referential noun phrase that translates as α ($\in \mathrm{Ty}_e$) and B is a verb phrase that translates as β ($\in \mathrm{Ty}_{et}$), then $F(\alpha,\beta)$ translates as $\ulcorner\beta(\alpha)\urcorner$.

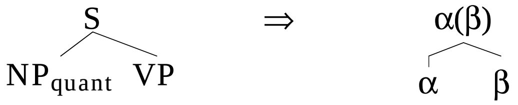
Instead of this clumsy formulation we will use the more suggestive notation:

(27)

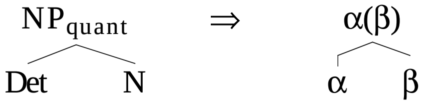


Here are some more rules in the same format:

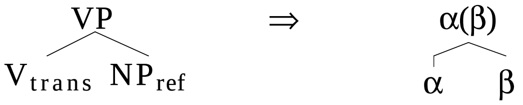
(28)



(29)



(30)



We have already seen that all these constructions only involve functional application. On the other hand, combining a transitive verb with a quantified object noun phrase requires some more effort. In an indirect interpretation framework, the relevant rule is most easily found if we consider sentences like (31) and their intended translations (31'):

(31) ***Everyone knows someone.***

(31') $(\forall x)\ (\exists y)\ K(x,y)$

(We have deliberately ignored the implicit relativization of the two quantified NPs to persons; it only obscures the point.) We have seen that the notation '$(Qx)\ \varphi$' can be thought of as a variant of '$Q([\lambda x\ \varphi])$', whenever Q is a quantifier ($\in Ty1_{(et)t}$), φ is a sentence ($\in Ty1_t$) and $x \in Var_e$. So (31') actually is:

(31") $\forall(\ [\ \lambda x\ \exists(\ [\ \lambda y\ K(x,y)\ ]\ )\ ])$

(Recall that '$K(x,y)$' is the same as '$K(y)(x)$'.) Clearly, '$\forall$' is the translation of the subject which, by (28), is applied to the translation of the predicate ***knows someone***. So the latter can be translated by:

(32) $[\ \lambda x\ \exists(\ [\ \lambda y\ K(x,y)\ ]\ )\ ]$

or something equivalent to it. But there is no question how (32) can be obtained from the translations 'K' of ***know*** and '$\exists$' ***someone***. The only important thing is to notice that this combination also works for other quantifiers, whether relativized or not. Take ***every stranger***, which translates as:

(33) $[\lambda X_{et}\ (\forall x)\ [S(x) \rightarrow X(x)]\ ]$

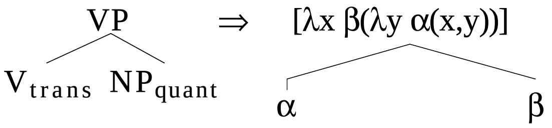
Replacing the existential quantifier in (32) by this relativized universal quantifier results in (34), which is equivalent to (34'), by two λ-conversions (one of them critical).

(34) $[\ \lambda x\ [\lambda X_{et}\ (\forall x)\ [S(x) \rightarrow X(x)]\ ]\ (\ [\ \lambda y\ K(x,y)\ ]\ )\ ]$

(34') $[\ \lambda x\ (\forall y)\ [S(y) \rightarrow K(x,y)\ ]\ ]$

But one can easily verify that (34') gives the intended extension for the VP ***know every stranger***. There is thus hope for a generalization of (32):

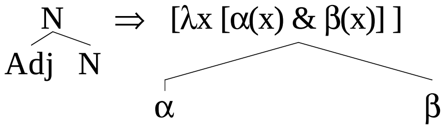
(35)



According to (35), the result of combining a transitive verb denoting a binary relation R with a quantified object denoting a set Q of sets of individuals denotes the set $\{x\in D_e \mid \{y\in D_e \mid xRy\} \in Q\}$, i.e. $XY(R,Q)$, as it was defined in part 3.

We have just seen that adjectives like ***female*** can be translated as constants of type *et*. How are they going to combine with other expressions? We will only look at one particular construction, viz. the attributive use, as in ***female person***. Since the latter obviously denotes the intersection of the extensions of ***female*** and ***person***, all we have to do is a way of defining that intersection in Ty1. But we have already seen how this is done, in our discussion of restrictive relative clauses: intersection amounts to a combination of conjunction and abstraction. So here is the general rule for attributive adjectives:

(36)



Note that (36) combines two sets of type *et* into another such set. It was remarked earlier that such combinations are outside the scope of Lambek's Calculus. So, in an obvious sense, the method of indirect interpretation gives us more than the latter. We will have to return to this point.

A similar rule could be used to combine nouns and restrictive relative clauses, but we will not give it here. Clearly, it would have to presuppose that we first construe the relative clause out of a sentence and interpret it by abstraction.

A celebrated piece of logical analysis is Montague's treatment of the copula in indirect interpretation, which actually goes back to Quine. Consider the following sentences:

(37) ***Every cow is a mammal.***
(38) ***Every cow is four-legged.***

In order to translate the first sentence, we need a translation of the indefinite article, ***a***. For our present purposes, the following will do:

(39) ***a***' := $[\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]$

(Note that this is just the same as the translation of ***some*** given in (25) above.) Now we must find a suitable extension for the *copula* ***is***. (37) obviously attributes the property of being a mammal to every cow; so the extension of the VP ***is a mammal*** must be the same as that of the noun ***mammal*** itself. Similarly, (38) attributes the property of being four-legged to every cow; so the extension of the VP ***is four-legged*** must be the same as that of the adjective ***four-legged***. The copula ***is*** thus does not seem to contribute to the meaning of the sentence. (We are deliberately ignoring tense.) This would suggest a semantically void rule of combining the latter with an adjective. We will not further discuss this possibility here. But we observe that the same strategy does not work in (37) where the copula is combined with the NP ***a mammal***: according to (39), the latter denotes a certain quantifier, and it is not at all obvious how this can be combined with anything (i.e. the desired extension of ***is***) in order to produce the set denoted by ***mammal***. This could be taken as one indication that (39) is incorrect.

Quite surprisingly, there exists a natural possibility of combining (39) with an equally natural extension of the copula. In order to motivate it, we will first look at what seems to be yet another use of ***is***, viz. in *identity statements*:

(40) ***Barbara Vine is Ruth Rendell.***

The reason why (40) could be of interest to is that (37) can be paraphrased by:

(37') ***Every cow is identical to a mammal.***

Note that (37') does not express that the quantifiers denoted by the two NPs in it are identical. Rather, it seems to be a quantified identity statement. In order to purse this idea, we will assume that the two proper names occurring in (40) are referential noun phrases to be translated by the Ty1-constants ‘b’ and ‘r’ of type *e*. A plausible translation of (40) then is:

(40') (b = r)

Since ***Barbara Vine*** is the (referential) subject of (40) that would have to be combined with the extension of the predicate by functional application, we expect the VP ***is Ruth Rendell*** to be translated by something equivalent to:

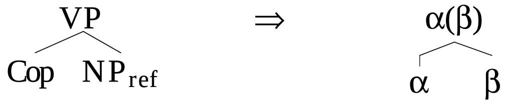
(41) [λx (x = r)]

How can we get (41) from the translation ‘r’ of ***Ruth Rendell***? The answer is all too obvious: by combining it with (42):

(42) [λy [λx (x = y)]]

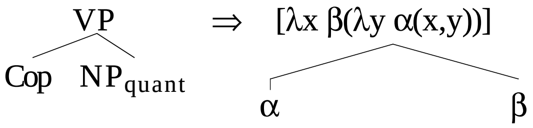
So (42) may be regarded as the translation of ***is***. Note that (42) denotes the relation of identity between individuals. So the assumption that ***is*** translates as (42) can be characterized as an *analysis of the copula as identity*. In order to actually obtain (41), we need a translation rule for the relevant construction:

(43)



Cop is, of course, the category of ***is***, and it corresponds to the type *e*(*et*) of binary relations; this information can be read off the translation (42). We now observe that (42) predicts the same combination as (30) and that the same types are involved, because *e*(*et*) is also the type of transitive verbs. So maybe the analogy carries over to the combinations with quantified instead of referential noun phrases. So how about combining the extension of ***is*** with a a quantifier by applying the operation *XY*? Here is the hypothetical translation rule:

(44)



If we now apply (44) to ***is a mammal***, we obtain:

(45) $[\lambda x\ [\lambda X_{et}\ (\exists x_e)\ [M(x)\ \&\ X(x)]\ ]\ ([\lambda y\ [\lambda y\ [\lambda x\ (x = y)]\ ]\ (y)\ (x)\ ]\ )\ ]$,

which λ-reduces to:

(45') $[\lambda x\ (\exists y_e)\ [M(y)\ \&\ (x = y)]\ ]$

But the first-order formula

(46) $(\exists y_e)\ [M(y)\ \&\ (x = y)]$

is logically equivalent to (46'), by usual laws of identity.

(46') $M(x)$

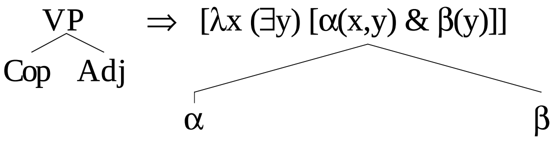
So (45') reduces to:

(46") $[\lambda x\ M(x)]$,

which is equivalent to ‘M’, due to η-conversion. So the idea of analyzing the predicative use of the copula in (37) as identity turned out to be sound. Needless to say that this does not mean that this is the only or the best analysis possible.

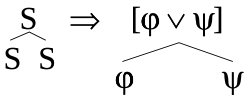
How about (38), then? Can we somehow make use of the translation (42) of ***is***, or are we forced to assume that in this case the copula does not carry any information (other than temporal information)? One possible analysis of this predicative construction is to think of the adjective ***four-legged*** as expressing something like ***a four-legged individual*** so that combining the copula with it would again result in the desired predication. We will not go into the details of this possibility but merely state it in form of a (conceivable) translation rule the correctness of which will be shown in an exercise:

(47)



We will now briefly turn to some other easily translatable constructions most of which have already been mentioned. The first of these is disjunction, i.e. the inclusive reading of ***or***. If it is used as a sentence connective, it receives a straightforward treatment:

(48)



(Recall that (48) is only a sloppy notation for a more precise formulation like (T_F) that would make clear that the construction to be translated is one of ***or***-coordination.) It appears that ***or*** can also occur among expressions of categories other than S:

(49) ***Caroline loves Alain or Tom.***

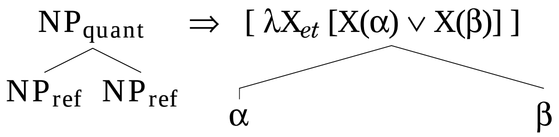
(50) ***Caroline loves every boy or some girl.***

(51) ***Caroline hugs Alain or kisses Tom.***

(52) ***Caroline hugs or kisses Tom.***

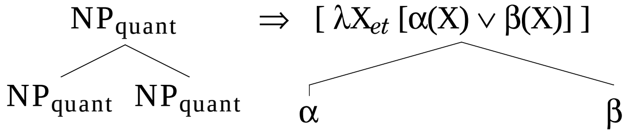
One way of analyzing (49) - (52) is by *conjunction reduction*, i.e. by assigning it a structure involving disjunctions of sentences as in (48). An alternative treatment would regard all the above ***or***s as *constituent coordinations*. We have already seen how such coordinations could be interpreted. Thus, e.g., the coordination of the two referential NPs ***Alain*** and ***Tom*** produces a quantified noun phrase which can be combined with a VP thus producing the corresponding disjunction. A tentative translation rule would thus be:

(53)



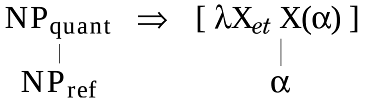
Similarly, (50) requires straightforward rule of quantifier disjunction:

(53')



However, this would still not cover all cases of NP-conjunction; for a quantified noun phrase might also be coordinated with a referential one, as in ***Alain or some boy***. It would thus appear that we need two more rules to cope with this difficulty. However, there is a more economic way of dealing with these cases, viz. Montague Lifting of referential NPs to quantified ones and then coordinating NP_{quant}s only. So we replace (53) by:

(53")

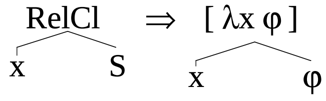


Yet another possibility would be to eliminate the category NP_{ref} altogether and translate by *(et)t*-expressions of the form [λX X(α)]. This is actually strategy adopted by Montague (a typical instance of 'generalizing to the worst case'), and it has become more and more out of fashion: nowadays semanticists prefer to assign the lowest (smallest) types possible to any expressions and only start lifting and shifting when syntactic constructions seem to force them to. In general, the modern strategy leads to simpler and more canonical translations – provided that type changes can be predicted (e.g. by something like Lambek's Calculus).

We will now leave the area of coordination; disjunctions of verb constituents will be treated in an exercise, and coordination with ***and*** immediately leads to problems in the semantics of plurals that we want to avoid.

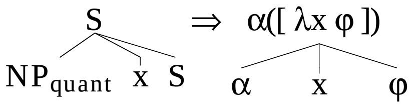
Finally, there are those constructions the translations of which involve binding of hitherto free variable. We are going to briefly discuss two of them. The first one, the relative clause, was already mentioned as a motivation for having Abstraction. The idea was to translate relative clauses as sentences with a free variable that gets bound by a λ-operator; and the syntactic input tells us which variable is going to be bound. We thus have something like:

(54)



Note that this rule requires logical forms to contain variables. The same assumption will have to be made for the other variable binding construction, viz. *quantifier raising* (or *quantifying in*), in which a quantified NP is combined with a sentence (containing a free variable); the result is a again a sentence, with the NP taking scope over the rest. In order to translate the construction, one should again recall that the usual variable binding notation '(Qx) φ' type-logically amounts to applying a (generalized) quantifier to a set obtained by Abstraction. Moreover, which variable must be bound, is a matter of syntactic information. We thus have:

(55)

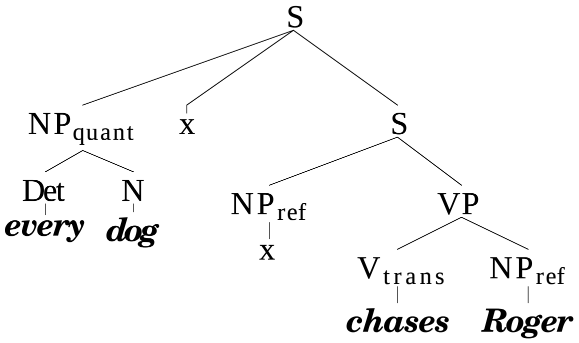


Let us apply (55) to a simple example in order to see how it works:

(56) ***Every dog chases Roger.***

If we take

(57)

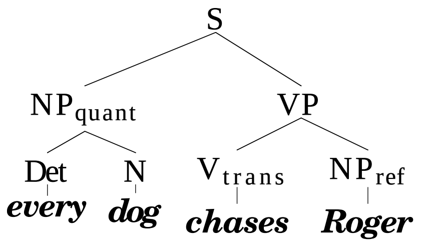


to be a first approximation of the structure of (56), it is easily shown (guess where) that the sentence will be translated into:

(58) $(\forall x)\ [D(x) \rightarrow C(x,r)]$,

which is also the translation of the *in situ* structure (57'):

(57')



Variable-binding rules make sense only if there are any variables to be bound, i.e. if the translations of at least some expressions contain free variables. This may be the case for certain indexed noun phrases, traces, and other elements of syntactic structures. These free variables will then be passed on to the translations of larger expression until they finally get bound by rules like (54) or (55). This means that the rules applying in the meantime must not change the status of these variables, i.e. they must not bind them. Care must thus be taken in their formulation, because otherwise some unexpected binding will take place. In fact, some of the rules above could produce such effects, because we had formulated them without thinking of free variables. The translation (36) of the noun/adjective combination is an example. For if a noun translated as a formula $\ulcorner[\lambda y\ R(x,y)]\urcorner$, where x is the very variable used in (36), the result of applying the latter would no longer denote the intersection of the extension of the noun with that of the adjective (which we may assume to be translated by the constant A):

(59) $[\lambda x\ [\ A(x)\ \&\ [\lambda y\ R(x,y)]\ (x)\ ]\ ]$

Since x does not get bound in the translation of the noun, λ-reduction turns (59) into:

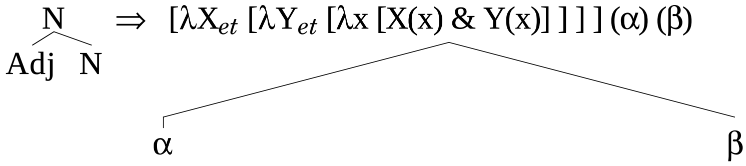
(59') $[\lambda x\ [\ A(x)\ \&\ R(x,x)\ ]\ ]$

which is *not* equivalent to the intended intersection; the latter could be denoted by:

(59'') $[\lambda y\ [\ A(y)\ \&\ R(x,y)\ ]\ ]$,

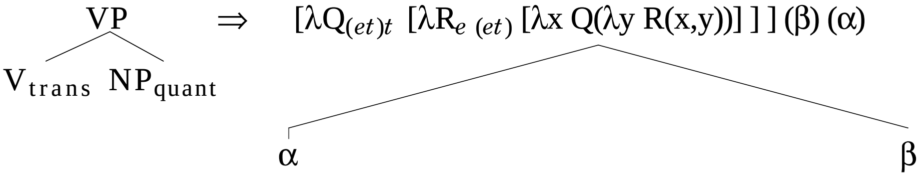
and this formula still contains free x, as required: as long as the operation does not bind anything, free variables are passed on. There are several ways of solving this problem about inadvertent binding. One could, e.g., argue (or even prove) that the translation of nouns never contain free variables. While this may or may not be so, there are more systematic ways of attacking the problem. We may have required the variable x occurring in (36) to have no free occurrences in either α or β; and while there is nothing wrong with this requirement, we will not adopt it, because it would force us to leave the general framework of indirect interpretation to be discussed in the following section. A third, and less radical, solution is to avoid it by a reformulation of (36). The following version will do:

(36')



The trick is just to get the constituent translations α and β out of the scope of the λ-operator. The result clearly denotes the intersection of α's and β's extensions. Similar changes may have to be made for some of the other rules discussed above. In particular the *XY*-rule (35) would have to become:

(35')



4.6 Compositional translation

Although the examples discussed so far should already give an idea of what compositional translation algorithms in general look like, we will still take a closer look at the general theory of indirect interpretation. Apart from their intrinsic interest, these considerations will help us to motivate and better understand a certain approach to restricting semantic operations to be discussed thereafter.

The translation procedure sketched in the previous section consisted of two components, reflecting the division of natural language expressions into lexical and complex ones. We take it that this is so for any translation algorithm used in indirect interpretation. The translation of the lexicon is a function assigning to every basic natural language structure a logical expression. We have seen that the translation of a basic expression may or may not be basic itself; so the general notion of a translation algorithm should not say anything about the complexity of lexical translations. On the other hand, we do have to say something about the categories of the resulting expression. For the translations of the expressions of one category should all be of the same logical language category; otherwise they could not be combined by the same translation rules. We will thus assume that the translation algorithm is *based on* a *category correspondence* between the categories of the two languages, i.e. a function assigning to every natural language category a category of the logical language. Note that if the latter is Ty1 (which we will not require), such a correspondence amounts to a type assignment.

The second component of the translation algorithm takes care of complex expressions by reducing their translations to those of their parts. More specifically, to each mode F of construing a complex structure $F\,(\Delta_1,\ldots,\Delta_n)$ out of some given structures $\Delta_1,\ldots,\Delta_n$ there corresponds a translation rule combining the translations $\alpha_1,\ldots,\alpha_n$ of $\Delta_1,\ldots,\Delta_n$ into a (usually complex) expression ρ of the logical language. Now from the examples discussed above it is clear that ρ itself is essentially a formula of the logical language containing the constituent translations $\alpha_1,\ldots,\alpha_n$. Strictly speaking, however, something like $\ulcorner\beta(\alpha)\urcorner$, i.e. the ρ from the translation rule (27), is not a formula of the logical language Ty1, because our *meta-variables* α and β aren't. But $\ulcorner\beta(\alpha)\urcorner$ is a 'schematic formula' or a (*syntactic*) *polynomial* of the language Ty1: it is built up exactly like a formula, but the positions of some sub-expressions have been filled by meta-variables. The same is true of all other schematic translations ρ occurring in the translation rules discussed above. And we will assume that they reflect the general form of translation rules. We are now in a position to give the main definition of compositional translation:

Definition: Given a (natural) language L_N and a (logical) language L_L as well as a category correspondence between them, a *translation algorithm from L_N to L_L* consists of two components:

- a *lexical translation* assigning to any basic expression of L_N

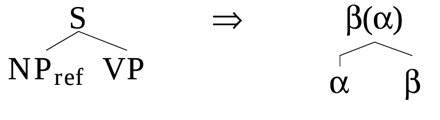
an expression of L_L of the corresponding category;

- an *operational translation* assigning to every (n-place) syntactic operation F of L_N a syntactic polynomial ρ (in $\alpha_1,\ldots\alpha_n$) of L_L with the following property: all occurrences of $\alpha_1,\ldots\alpha_n$ in ρ replace expressions of categories corresponding to the input categories of F.

(A polynomial *in* the meta-variables $\alpha_1,\ldots\alpha_n$ is simply a polynomial containing no meta-variables other than $\alpha_1,\ldots\alpha_n$.) The condition on the resulting polynomial ρ looks pretty complicated, but it only blocks silly translations like $\ulcorner(\alpha_1 = \alpha_2)\urcorner$ of a rule combining a quantified noun phrase and a verb phrase: the two categories correspond to different types and hence translations of NPs and VPs cannot flank the same identity sign.

It is tempting to think of the meta-variables occurring in syntactic polynomials as notational variants of object-language variables. However, there are important differences that become clear when one attempts to describe the semantic effect of translation rules. In part 1 we saw that, according to compositional interpretation, to every syntactic combination there corresponds a semantic operation, i.e. a combination of meanings. Given a compositionally interpreted logical language (like Ty1) and a translation algorithm (like the one described in the previous section), one may now wonder which semantic combinations the translation procedure ascribes to the syntactic operations of the natural language. Let us look at a simple example:

(27)



Our type assignment told us that the two categories involved, NP$_{\text{ref}}$ and VP, correspond to e and et, respectively. The resulting polynomial $\ulcorner\beta(\alpha)\urcorner$ thus meets the standards of the above definition (if we ignore the purely notational differences between 'α_1' and 'α' and between 'α_2' and 'β'). Moreover, (27) tells us to combine the translation of a referential subject and that of a VP by applying the Ty1-operation of Application, in order to obtain the translation of the whole sentence. According to the semantic rules of Ty1, this combination denotes the result of applying the extension of the functor (i.e. the translation of the predicate) to that of the argument (translating the subject). If we now define the extensions of the English expressions to be the extensions of their translation – and this is, of course, the idea behind the whole approach – we see that the construction translated in (27) is interpreted as functional application. Using Schönfinkel's trick we find that the very same operation is denoted by the Ty1-expression:

(60) $[\lambda X_{et}\ [\lambda x_e\ X(x)\ ]\ ]$

Note that in (60) the Ty1-variable X plays the role of the meta-variable β in (27), and $x \in Var_e$ corresponds to α. The λ-operators only indicated that the whole formula is to be interpreted as a function (operation); this corresponds to the occurrences of the meta-variables α and β in the daughter nodes of the right tree in (27). So the part of the polynomial in (27) could also have been played by the Ty1-formula (60). Similarly, one could argue that the more complicated polynomials (61) and (62) (occurring in (35') and (36')) could be replaced by (61') and (62'), respectively:

(61) $[\lambda Q_{(et)t}\ [\lambda R_{e\,(et)}\ [\lambda x\ Q(\lambda y\ R(x,y))]\ ]\ ]\ (\beta)\ (\alpha)$

(61') $[\lambda Q_{(et)t}\ [\lambda R_{e\,(et)}\ [\lambda Q_{(et)t}\ [\lambda R_{e\,(et)}\ [\lambda x\ Q(\lambda y\ R(x,y))]\ ]\ ]\ (Q)\ (R)\ ]\ ]$

(62) $[\lambda X_{et}\ [\lambda Y_{et}\ [\lambda x\ [X(x)\ \&\ Y(x)]\ ]\ ]\ ]\ (\alpha)\ (\beta)$

(62') $[\lambda X_{et}\ [\lambda Y_{et}\ [\lambda X_{et}\ [\lambda Y_{et}\ [\lambda x\ [X(x)\ \&\ Y(x)]\ ]\ ]\ ]\ (X)\ (Y)]\ ]$

(61') and (62') are obtained form the corresponding polynomials (61) and (62) in the same way as (60) was obtained from the polynomial $\ulcorner\beta(\alpha)\urcorner$, i.e. by replacing the meta-variables by Ty1-variables (of the correct type) and then abstracting from these variables. Applying η-conversion to (61') and (62'), we see that they are equivalent to:

(61") $[\lambda Q_{(et)t}\ [\lambda R_{e\,(et)}\ [\lambda x\ Q(\lambda y\ R(x,y))]\ ]\ ]$

(62") $[\lambda X_{et}\ [\lambda Y_{et}\ [\lambda x\ [X(x)\ \&\ Y(x)]\ ]\ ]\ ]$

Clearly, in any model (61") denotes the operation *XY* and (62") denotes intersection (of sets of individuals). So the operations denoted by these Ty1-formulae turn out to be the semantic operations expressed by the polynomials occurring in the translation rules.

Unfortunately, this trick of turning polynomials into Ty1-formulae does not work for variable-binding rules like relative-clause formation (54) or quantifier raising (55). Let us just consider the former because it is simpler. Its resulting polynomial is:

(63) $[\lambda x\ \varphi]$

The way (63) is used in (55) makes it perfectly clear that it is a polynomial in x and φ: both meta-variables occur as daughters in the translation tree. ('x' *is* a meta-variable that *refers* to Ty1-variables!) If we now want to replace the polynomial (63) by a corresponding Ty1-formula, we're in trouble; for the old trick of replacing meta-variables by object variables won't work anymore: once the meta-variable 'φ' gets replaced by a Ty1-variable (of category *t*, in this case), the λ-operator becomes vacuous: the formula

(64) $[\lambda y_e[\lambda p_t[\lambda y_e\, p]\,]\,]$

will always return a constant function of type *et*, no matter what it is applied to. Let us check this with a concrete example. The translation of the relative clause ***who is asleep*** (before applying the formation rule) might be:

(65) A(x),

where x is the variable to be bound by the relative pronoun. Clearly, rule (55) would turn this into:

(66) [λx A(x)],

which is equivalent to the constant 'A' itself, again by η-conversion. In particular, then, the relative clause ***who is asleep*** denotes whatever the model wants 'A' to denote. Now we watch what happens if we apply the function defined in (64) to the translation of the variable x (i.e. x itself) and that of the clause, (i.e. (65)):

(67) $[\lambda y_e[\lambda p_t[\lambda y_e\, p]\,]\,]\,(x)\,(A(x))$

If we apply one λ-conversion, the first argument simply gets swallowed, because y has no free occurrences in the functor. Now for the second one, i.e. (65). Under the assumption that 'x' and 'y' denote different Ty1-

variables, λ-conversion is o.k. and we get:

(67') $[\lambda y_e\ A(x)]$

(If 'x' and 'y' were the same variable, we would have had to rename its bound occurrences in the functor, thus again obtaining (67') or an alphabetic variant of it.) Since (67') contains a free variable, its extension depends on the variable assignment g: either g(x) is in the set denoted by 'A' or it isn't; in the first case (67') denotes the universe D_e, otherwise its extension will be the empty set. This is clearly an unwelcome result, because the extension of 'A' (and hence the intended extension of the relative clause) may be any subset of D_e.

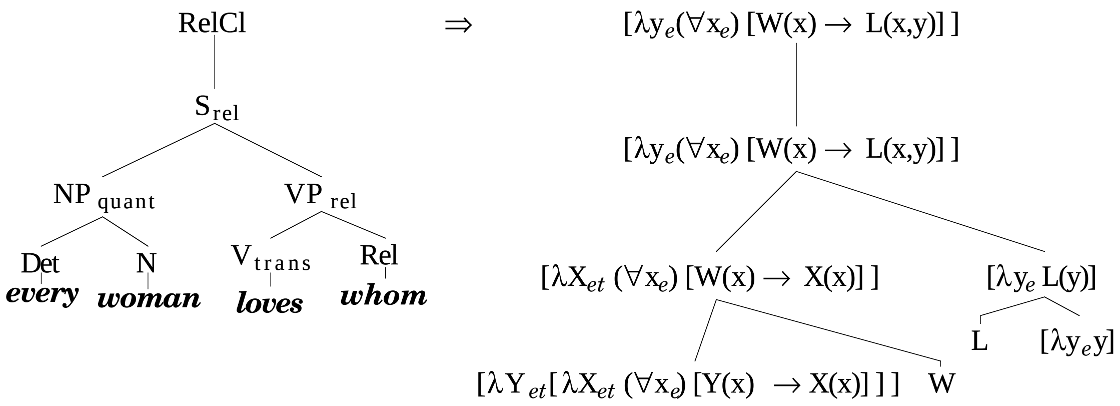
The reason why not all polynomials correspond to formulae has to do with the fact that, in the presence of variable-binding operations, compositionality cannot be based on meanings as extensions. We have already seen this in part 1, where it was argued that the substitutional interpretation of quantification is non-compositional, because it rests on the assumption that meanings are extensions. In order to obtain a compositional treatment of quantification, we used Tarski's idea of identifying meanings of predicate logic expressions with functions from variable assignments to extensions. The same trick works in more complicated languages like Ty1: if we define the (model-dependent) *meaning* of a formula $\alpha \in Ty_a$ as that function that maps every variable-assignment g (based on the model's ontology) to α's extension $[\![\alpha]\!]^{F,g}$ under that assignment, we could (but won't) show that the interpretation of Ty1 given in section 4.2 is indeed compositional. The proof is completely parallel to the corresponding one for predicate logic; the only critical case is Abstraction. In fact, without the latter operation, one would not need the complicated Tarskian meanings for Ty1-expressions, but could do with extensions instead. More precisely, the other two operations, Application and Identity, are *extensional* in the sense that they can be *reduced to* certain operations on extensions: if, e.g., b_1 is a functor meaning (mapping variable assignments to functions of type D_{ab}) and b_2 is a corresponding argument meaning (mapping variable assignments to elements of D_a), then the result of combining b_1 and b_2 by the semantic operation corresponding to Application is a meaning b_3 whose value for a given variable assignment is completely determined by the values of b_1 and b_2 on that assignment: $b_3(g) = b_1(g)\ (b_2(g)) = FA\,(b_1(g), b_2(g))$. In that sense, Application reduces to the combination

FA of extensions. Similarly, the interpretation of Identity as an operation on Tarskian meanings reduces to a combination *Id* on extensions, as is clear from the semantics of Ty1.

The distinction between extensional and non-extensional operations is very general and can be applied to arbitrary combinations of (Tarskian) meanings. In particular, it can also be applied to those operations that correspond to or are *defined* by syntactic polynomials, i.e. those that combine any given meanings $b_1,\ldots b_n$ just in the way the interpretation of a polynomial in $\alpha_1,\ldots\alpha_n$ combines whatever the meanings of $\alpha_1,\ldots\alpha_n$ could be. It is not hard to see that variable-binding polynomials always define extensional operations. (The converse is not quite true, but in most cases it does hold.) In order to fully understand the scope of this observation, one should recall that a variable-binding polynomial is one that contains a λ-operator with a meta-variable in its scope. Thus, e.g., the resulting polynomial of the 'Adj+N' rule (36') is not variable-binding, even though it does contain three λ-operators. On the other hand, both relative clause formation (54) and quantifier raising (55) have been translated by variable-binding polynomials, but they were the only rules in the fragment with this property. Now, we have already observed that the semantic operations defined by our non-variable-binding resulting polynomials ρ could also have been defined by formulae ρ* containing Ty1-variables in place of the meta-variables. Strictly speaking, however, the latter do not define semantic operations at all, but rather denote operations on extensions. On the other hand, it can be shown that any non-variable-binding polynomial ρ defines an (extensional) operation that reduces to the function denoted by the corresponding formula ρ*. We will not prove this general proposition here, because its content should already be clear from the examples discussed so far.

It should be pointed out that there are various ways of avoiding variable-binding operations and thus guaranteeing that all polynomials can be defined by formulae of the logical language. Given such a strategy, one could thus avoid the use of syntactic polynomials in indirect translation altogether. Although we will not discuss any of these strategies here, we will give a rough idea how one could proceed. One way of doing it can be illustrated by a reformulation of the compositional translation of relative clauses. Earlier on we have motivated the use of Abstraction by showing that it allows us to reduce this problem to the problem of translating sentences containing free variables. However, we may be prepared to give up this reduction and prefer the following kind of translation instead:

(68)


RelCl
⇒
[λy$_e$(∀x$_e$) [W(x) → L(x,y)]]
S$_{rel}$
[λy$_e$(∀x$_e$) [W(x) → L(x,y)]]
NP$_{quant}$
VP$_{rel}$
Det
N
V$_{trans}$
Rel
every
woman
loves
whom
[λX$_{et}$ (∀x$_e$) [W(x) → X(x)]]
[λy$_e$ L(y)]
L
[λy$_e$y]
[λY$_{et}$[λX$_{et}$ (∀x$_e$) [Y(x) → X(x)]]]
W

(The final recategorization only indicates the movement of the relative pronoun, if necessary; it has no semantic effect.) The (categorial) idea is to treat an expression containing a relative pronoun as an incomplete expression with a slot (or *gap*) for a referential noun phrase. In particular, the pronoun itself is a referential NP with a gap for a referential NP, corresponding to type *ee*! Semantically, leaving a gap amounts to not specifying the meaning of the expression to fill this gap, i.e. to treat it as arbitrary. Thus, wherever the gap appears we find a *bound* variable in the translation. Apart from passing on this bound variable to all higher nodes, the semantic combinations are the same as in the corresponding constructions without gaps.

The above example hopefully illustrates one general strategy of avoiding variable-binding polynomials; we will not say more about it. Another, somewhat related, but more radical strategy consists in eliminating all variable-binding operations from the logical language. Note that this has not been done in (68), for even the translation of the relative pronoun contains a λ-operator. But it is possible to construe alternative languages (so-called *combinatory logics*) that avoid all variable-binding. For these languages, definability by polynomials amounts to definability by formulae. In fact, such languages can be given a compositional interpretation with extensions as meanings. Unfortunately, the formulae of combinatory logics tend to be pretty complicated to read, especially if the expressive power of such a language is as strong as that of Ty1 (which is possible). Nevertheless, some semanticists (e.g. Anna Szabolczi) have suggested to use such languages without variable-binding for reasons having to do with the topic of the next section.

We have referred to the kind of translation algorithm exemplified in the previous section as *compositional*, because of obvious similarities with compositional meaning assignment: in either case, something (i.e. a meaning of a formula) is assigned to natural language expressions by combining whatever their immediate parts get assigned in a fashion corresponding to the construction of the expressions. And as we have already indicated in section 4.1, compositional translation algorithms not only provide meanings for the natural language expressions, but they even do so in a compositional way: it can be proved that the meaning assignment that consists in giving to each natural language expression the meaning of its logical language translation is a compositional meaning assignment, provided that the logical language itself is interpreted in a compositional way.

One way of seeing what compositionality does to translations is to look at counter-examples, i.e. rules (or whole algorithms) that do not meet this requirement. Let us look at a simple example, viz. a non-solution to the NP+S analysis problem of restrictive relative clauses. We first observe that, according to our (standard) rules, the translations of quantified noun phrases all reduce to Ty1-formulae of the form:

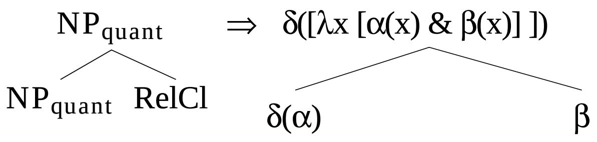
(69) $\delta(\alpha)$,

where α is the (possibly complex) translation of a noun and δ translates a determiner. Moreover, it is easily seen that restricting the noun by a relative clause results in a translation that is equivalent to:

(70) $\delta([\lambda x_e\,[\alpha(x)\ \&\ \beta(x)]])$,

where β is the translation of the relative clause. (For simplicity we assume that neither the translation of α nor that of β contains free x; nothing but the shortness of the formulae depends on that.) Comparing (69) and (70), may suggest the following translation rule:

(71)



Under the (sound) assumption that (69) is indeed the general form of NP$_{\text{quant}}$-translations, (71) is a well-defined translation rule. But it is not compositional, in a very strong sense. First of all, it does not meet the standards of compositional translation algorithms as they were defined above, because the resulting formula contains more meta-variables than the (assumed) syntactic operation takes arguments: the latter only combines two expressions, but the former combines three logical formulae. Maybe we could live with that. But secondly, and that's the strong part, assigning the meanings of their Ty1-translations to natural language expressions would lead to a non-compositional meaning assignment: this follows directly from the fact that there is no compositional way of interpreting the NP+S analysis of relative clauses that yields the same results as the N+S analysis and at the same time treats quantified NPs as generalized quantifiers. But the above rule gives such an analysis. More specifically, applying (71) to the example discussed in exercise 2 would give the correct result, but it would distinguish between synonymous expressions, just because they happen to get different, though equivalent translations. So changing the format of the translation rules may result in non-compositional interpretation. Moreover, in the presence of rules like (71), we may no longer safely reduce translation to logically equivalent formulae: ***every president*** and ***some president*** will get equivalent translations but neither must be replaced by the other one, because (71) treats them differently. So if we want to continue using logical reductions, we better stick to the compositional translation algorithms defined above.

4.7 Semantic Operations

If one logical language suffices as a universal medium suited for the indirect interpretation of all natural languages, the whole approach naturally suggests a restriction on all possible semantic operations, viz. that they be polynomially definable in that logical language. Although we will see that Ty1 a suitable universal medium, it is still instructive to investigate this *Definability Hypothesis* for Ty1: the languages that have actually been proposed for such a purpose all bear a certain resemblance to it and a better understanding of definability in Ty1 certainly contributes to a better understanding of definability in a more realistic medium. In particular, the tools for relating Ty1-definability to the other approaches to restricting semantic operations (Lambek's Calculus and logicality) have proved to be quite generally applicable. And it is these tools that we will be primarily interested in. Our restricted perspective also motivates a further simplification that we will from now on make: we will concentrate on extensional operations in the sense defined in the previous section. It must be emphasized, though, that the methods discussed in this section also carry over to variable-binding operations; but everything would get more complicated and messy. Given what we have observed about extensionality, the Definability Hypothesis can be formulated as follows:

Definability Hypothesis (for Ty1)
Any possible extensional semantic operation G taking extensions of types $a_1,\ldots,a_n$ to extensions of type b can be defined by a Ty1-expression α_G of type $a_n(\ldots(a_1\ b)\ldots)$: $G\,(\mathrm{b}_1,\ldots,\mathrm{b}_n) = [\![\alpha_G]\!]^{F,g}\,(\mathrm{b}_n)\ldots(\mathrm{b}_1)$.

(For ease of presentation, we have identified extensional operations with the operations that they reduce to.) This formulation of the hypothesis contains a certain vagueness that we will have to eliminate. For it depends on a model without specifying which one. One may suspect that *some* model would do. Unfortunately, the hypothesis would trivialize that way. For if G is any (extensional) semantic operation we may pick a constant of the corresponding type and *define* the extension of the constant to be the function corresponding to G, via Schönfinkel's trick. So this constant would be the desired Ty1 definition of G.

Something is obviously going wrong here. For a constant is no definition, because it can be interpreted in any way we like; and the constant itself reveals nothing about its intended interpretation. What we would like to have, then, is some notion of a *Ty1-definition* that only applies to expressions whose interpretation does not depend on the particular model. Since a model's contribution to the interpretation of an expression consists in providing the extensions of the constants and free variables occurring in it, the natural candidate for the notion of a Ty1-definition is this:

Definition
A *definition* is a Ty1-formula that does not contain any constants or free variables.

If we now replace 'Ty1-expression' in the Definability Hypothesis by 'definition', the model used for determining α_G's extension becomes irrelevant. And that's exactly how we will read the hypothesis.

The Definability Hypothesis certainly works for our above fragment and instead of looking into its empirical consequences, we will rather study its formal content by relating it to the other two hypotheses discussed in part 3, viz. derivability in Lambek's Calculus and logicality. But we will first have to turn Lambek derivability into a constraint on semantic operations: so far we have only seen that it restricts the types of extensions that can be combined by semantic operations. In order to be more specific as to which of these operations are admissible, we will *interpret* the proof trees used to establish Lambek derivability. The interpretation will be indirect, by way of translating proofs into Ty1 definitions. The general strategy can already be seen by looking at the simplest proofs, viz. those that merely instantiate one of the axiom schemata of Lambek's Calculus. Let us start with (L0):

(L0) $a \rightarrow a$

We have already seen that one operation that combines extensions of type *a* with extensions of the same type is the identity operation (of type *aa*) that can be defined by:

(72) $\underline{[\lambda x_a}\ x]$

And (72) is the definition we will associate with any one-line proof consisting of an instance of the axiom (L0). Similarly, all instances of the axiom schema (L1) can be interpreted by functional application which can be defined by one of the Ty1-formulae in (73):

(L1) $(ab), a \rightarrow b\ ;\ a, (ab) \rightarrow b$

(73) $\underline{[\lambda x_a [\lambda f_{ab}}\ f(x)]];\ \underline{[\lambda f_{ab} [\lambda x_a}\ f(x)]]$

(The reason why the definitions in (73) present their arguments in the reverse order from the corresponding axiom only reflects the fact that they must define the Schönfinkel counterparts of the operations to be associated with the Lambek derivations; and, as we have seen, Schönfinkel turns everything around.) We now observe certain structural similarities between the axioms of Lambek's Calculus and the definitions we have associated with them: types occurring on the left hand-side of the arrows become variables of those types in the (underlined) *prefix* of the corresponding definition; and the type to the right of the arrow is the category of the expression following the prefix. This similarity is actually pretty natural, given the fact that the λ-terms say how arbitrary objects of the types of the variables occurring in their prefix can be combined into an object of the type following the prefix: $[\lambda x_a\, [\lambda y_b\, [\lambda z_c\, \alpha]]]$ can be read as: 'combine given x, y, and z of types *a*, *b*, and *c* into α (of whatever type α's category is).' But note that for the whole expression to be a definition, what follows the prefix must not contain any constants or free variables other than the one bound by the prefix λs.

Before we start worrying about longer proofs in Lambek's Calculus, we will introduce a shorthand-notation for our translations. Although in the above cases this notation hardly saves any space, it will turn out to be valuable when we translate the rules. In this new notation, the translations of (L0) and (L1) are given by:

(L0')

$$\begin{array}{ll} a \to & a \\ x_a & x \end{array}$$

(L1')

$$\begin{array}{lll} (ab), & a \to & b \\ f_{ab}, & x_a & f(x) \end{array}$$

(L1")

$$\begin{array}{lll} a, & (ab) \to & b \\ x_a, & f_{ab} & f(x) \end{array}$$

In the firm hope that the notation is self-explanatory, we now turn to the task of translating complex Lambek proofs into Ty1 definitions. The general procedure can be seen by looking at the simplest example, viz. the rule (L4). The premise of the rule is that we have already derived something of the form:

(74) $b,\ a_1, \ldots a_n \to c$

We may thus assume that this derivation corresponds to a definition. In our notation:

(74') $$\begin{array}{lllll} b, & a_1, & \ldots & a_n & \to c \\ y_b\,, & x_{a_1}, & \ldots & x_{a_n} & \gamma \end{array},$$

where γ is an expression of category *c* containing (at most) the prefix variables as parameters. Applying (L4) then turns this proof into a proof of:

(75) $a_1, \quad \ldots \quad a_n \quad \rightarrow \quad bc$

We now want to associate with this proof a combination of $x_{a_1},\ldots,x_{a_n}$ into an expression of the type *bc*. Now whatever operation *O* is defined by (74'), it must combine arbitrary *n*+1 arguments (of the correct types) into something of type *c*; and what we want is an operation *O*' combining arbitrary *n* arguments (of the same types without *b*) into some *f* taking the *n*+1st argument (of type *b*) to something of type *c*. So it is only natural to let *O*' be that function that assigns to any $u_1,\ldots,u_n$ the *f* taking any v of type *b* to whatever *O* would have taken v plus $u_1,\ldots,u_n$: $f(v) = O(v,u_1,\ldots,u_n)$. And this *f* is definable by Abstraction. We thus get the following recursive translation rule:

(L4') If: then:

$$\frac{b,\ a_1,\ \ldots\ a_n\ \rightarrow c}{y_b,\ x_{a_1},\ \ldots\ x_{a_n}\quad \gamma}\ ,\qquad \frac{a_1,\ \ldots a_n\ \rightarrow bc}{x_{a_1},\ \ldots x_{a_n}\quad [\lambda y_b\,\gamma]}$$

Of course, we also need a similar rule for that variant of (L4) that takes the rightmost type on the left side across the arrow. It is clear that this rule is essentially of the same form, so that we will not have to state it here.

Note that the definition associated with applying (the first variant of) (L4) is the very same formula as the one associated with its premise. The difference lies only in the status of the ‘λy’-binding: in the translation of the premise it is part of the prefix and would thus correspond to the translation of a constituent to be combined by the corresponding syntactic rule. But after applying (L4) we get an operation with less arguments, and the ‘λy’ is part of the outcome of that operation.

We now turn to the remaining two rules of Lambek’s Calculus. They differ from what we have translated so far in that they involve combinations of types into more than one type. As in the original motivation for these rules, it is again unclear how we would want to interpret such *multiple* combinations. Here is a suggestion. Whenever we have a multiple combination like:

(76) $a_1, \ldots, a_n \rightarrow b_1, \ldots, b_m$,

we will associate with it m combinations of extensions, each of them combining $u_1,\ldots,u_n$ (of types $a_1,\ldots,a_n$) into one of the bs. In doing so, we can use our notation for these translations:

(76')
$$\begin{array}{llll} a_1, & \ldots a_n & \rightarrow b_1, & \ldots b_m \\ x_{a_1}, & \ldots x_{a_n} & \quad \beta_1, & \ldots \beta_n \end{array}$$

So (76') encodes the definitions $[\lambda x_{a_n} \ldots [\lambda x_{a_1} \beta_1]\ldots]$, $[\lambda x_{a_n} \ldots [\lambda x_{a_1} \beta_2]\ldots]$,... all the way up to $[\lambda x_{a_n} \ldots [\lambda x_{a_1} \beta_n]\ldots]$. It is here where our shorthand notation proves to be of value. Even though it looks a bit messy, the translation of (L2) is pretty straightforward:

(L2') If:

$$\begin{array}{llll} a_1, & \ldots a_n & \rightarrow b_1, & \ldots b_m \\ x_{a_1}, & \ldots x_{a_n} & \quad \beta_1, & \ldots \beta_m \end{array}$$

then:

$$\begin{array}{llllll} c_1, & \ldots c_k, & a_1, & \ldots a_n, & d_1, & \ldots d_l \rightarrow \\ y_{a_1}, & \ldots y_{a_k}, & x_{a_1}, & \ldots x_{a_n}, & z_{d_1}, & \ldots z_{d_l} \end{array}$$

$$\begin{array}{llllll} c_1, & \ldots c_k, & b_1, & \ldots b_m, & d_1, & \ldots d_l \\ y_{a_1}, & \ldots y_{a_k}, & \beta_1, & \ldots \beta_m, & z_{d_1}, & \ldots z_{d_l} \end{array}$$

So the rule only introduces the prefix variables on the right hand-side. Among the definitions associated with an application of (L4) will then be the terms $[\Lambda\ y_{a_1}]$, $[\Lambda\ \beta_m]$, and $[\Lambda\ z_{d_l}]$, where Λ is the long λ-prefix from 'λz_{d_l}' to '$\lambda\ y_{a_1}$'.

Finally we must deal with the transitivity rule (L3). Before giving the general procedure, let us look at an example. (22) of section 3.5 started like this:

$$\frac{\text{L1: } e, e(et) \rightarrow et}{\text{L2: } e, e(et), (et)t \rightarrow et, (et)t\ ;} \qquad \text{L1: } et, (et)t \rightarrow t$$

The rules introduced so far give us:

$$\text{L1:}\quad \begin{array}{lll} e, & e(et) & \to et \\ \mathrm{y}_e\,, & \mathrm{R}_{e(et)} & \mathrm{R(y)} \end{array}$$

____

$$\text{L2:}\quad \begin{array}{llll} e,\ e(et),\ (et)t & \to et, & (et)t \\ \mathrm{y}_e,\ \mathrm{R}_{e(et)},\ P_{(et)t} & \mathrm{R(y)}, & P \end{array} \qquad \text{L1:}\quad \begin{array}{lll} et, & (et)t & \to et \\ \mathrm{X}_{et}, & Q_{(et)t} & Q(\mathrm{X}) \end{array}$$

In order to interpret the application of (L3), we must now find a Ty1-definition of the form [λ*P* [λR [λy δ]]], where δ is of category *et*. Now, our interpretation of (L2) tells us how to combine three arguments (indicated by *P*, R, and y) into something of type *et* or (*et*)*t*; and (L1) combines arbitrary objects of the latter types into an object of the desired type *et*. So we can take the definition we get from (L1), i.e. [λ*Q* [λX *Q*(X)]], and apply it to the results of applying (L2) to our arbitrary arguments:

____

$$\text{L3:}\quad \begin{array}{ll} e,\ e(et),\ (et)t & \to et \\ \mathrm{y}_e,\ \mathrm{R}_{e(et)},\ P_{(et)t} & [\lambda Q_{(et)t}\ [\lambda \mathrm{X}_{et}\ Q(\mathrm{X})]]\ (P)\ (\mathrm{R(y)}) \end{array}$$

Note that the result in (L3) can be reduced by two λ-conversions, so that the definition associated with this proof of '$e,\ e(et),\ (et)t \to et$' turns out to be logically equivalent to:

(77) $[\lambda P_{(et)t}\ [\lambda \mathrm{R}_{e(et)}\ [\lambda \mathrm{x}_e\ P(\mathrm{R(y)})\ ]]]$

These reductions can always be performed whenever (L3) has been applied, so that we might as well give the translation of (L3) in terms of substitution rather than application of λ-expressions. However, we take the latter option because it is slightly more straightforward. So here is the general rule:

(L3') <u>If</u>: <u>and</u>:

$$\begin{array}{llll} a_1, & \ldots a_n & \to b_1, & \ldots b_m \\ \mathrm{x}_{a_1}, & \ldots \mathrm{x}_{a_n} & \beta_1, & \ldots \beta_m \end{array} \qquad \begin{array}{llll} b_1, & \ldots b_m & \to c_1, & \ldots c_k \\ \mathrm{y}_{a_1}, & \ldots \mathrm{y}_{a_m} & \gamma_1, & \ldots \gamma_k \end{array},$$

<u>then</u>:

$$\begin{array}{lll} a_1, & \ldots a_n & \to \\ \mathrm{x}_{a_1}, & \ldots \mathrm{x}_{a_n} & \end{array}$$

$$\begin{array}{ll} c_1, & \ldots c_k \\ [\lambda y_{a_m} \ldots [\lambda y_{a_1}\, \gamma_1\,]\ldots]\, (\beta_m)\ldots(\beta_1), & \ldots [\lambda y_{a_m} \ldots [\lambda y_{a_1}\, \gamma_k\,]\ldots]\, (\beta_m)\ldots(\beta_1) \end{array}$$

The interpretation of Lambek's Calculus as a device for restricting possible semantic operations is now complete. Let us try it out on one example, viz. the derivation (23) of section 3.5:

(78)

$$\text{L1:}\quad \begin{array}{lll} e(et), & e, & \rightarrow et \\ R_{e(et)}, & y_e & R(y) \end{array}$$

$$\text{L2:}\quad \begin{array}{llll} e, & e(et), & e & \rightarrow e,\ et \\ x_e, & R_{e(et)}, & y_e & x,\ R(y) \end{array} \qquad \text{L1:}\quad \begin{array}{lll} e, & e(et) & \rightarrow t \\ z_e, & S_{e(et)} & S(z) \end{array}$$

$$\text{L3:}\quad \begin{array}{lll} e(et), & e & \rightarrow t \\ R_{e(et)}, & y_e & [\lambda S_{e(et)}\, [\lambda z_e\ S(z)\,]\,]\, (R(y))\, (x)\quad (\approx \underline{R(x,y)\,!}\ ) \end{array}$$

$$\text{L4:}\quad \begin{array}{lll} e, & e(et) & \rightarrow et \\ x_e, & R_{e(et)} & [\lambda y_e\ \ R(x,y)\,] \end{array}$$

$$\text{L2:}\quad \begin{array}{lll} e,\ e(et),\ (et)t & \rightarrow et, & (et)t \\ x_e, R_{e(et)}, P_{(et)t} & [\lambda y_e\ R(x,y)\,], & P_{(et)t} \end{array} \qquad \text{L1:}\quad \begin{array}{lll} et, & (et)t & \rightarrow t \\ X_{et}, & Q_{(et)t} & Q(X) \end{array}$$

$$\text{L3:}\quad \begin{array}{ll} e,\ e(et),\ (et)t & \rightarrow t \\ x_e, R_{e(et)}, P_{(et)t} & [\lambda Q_{(et)t}\, [\lambda X_{et}\ Q(X)\,]\,]\, (P)\, ([\lambda y_e\ R(x,y)]) \\ & (\approx \underline{P([\lambda y_e\ R(x,y)])\,!}\ ) \end{array}$$

$$\text{L4:}\quad \begin{array}{ll} e(et),\ (et)t & \rightarrow et \\ R_{e(et)}, P_{(et)t} & [\lambda x_e\ P([\lambda y_e\ \ R(x,y)])] \end{array}$$

As expected, the function associated with this proof corresponds to the operation *XY* combining the extensions of transitive verbs and their quantified objects. In particular, then, the latter is *Lambek definable* because it is denoted by a Ty1-definition that can be obtained by translating a derivation in Lambek's Calculus.

The above procedure guarantees that every Lambek definable semantic operation meets the Definability Hypothesis. Is the converse also true? I.e. is every Ty1-definition equivalent to a translation of some Lambek derivation? This is not to be expected. For we have already mentioned (without proof) that the combination:

(79) $et, et \to et$

cannot be derived in Lambek's Calculus even though we do have, e.g., the definition:

(80) $[\lambda X_{et}\ [\lambda Y_{et}\ [\lambda x_e\ [X(x)\ \&\ Y(x)\ ]\ ]\ ]\ ]$

But maybe, then, Ty1-definability implies Lambek definability whenever the type combination is admissible according to Lambek's Calculus? Again, the answer is negative. For although (81) is just an instance of (L1) and translates into the definition (81'), the definition (81") cannot be obtained from any Lambek derivation:

(81) $e, ee \to e$
(81') $[\lambda f_{ee}\ [\lambda x_e\ f(x)]]$
(81") $[\lambda f_{ee}\ [\lambda x_e\ f(f(x))]]$

We will not prove this fact here. Intuitively, the reason why (81") is out is because it contains a bound variable ('f') that occurs more than once in the scope of its operator; however, the definitions to be obtained from translating Lambek derivations always have *exactly one* occurrence of a variable 'x' within the scope of 'λx'. (So (80) is excluded for a similar reason.) In particular, then, vacuous binding is not permitted, although it may occur in Ty1-expressions as we defined them:

(82) $[\lambda f_{ee}\ [\lambda x_e\ f]]$

These few examples indicate that Lambek definability – even though it is clearly too strong a restriction because it rules out (79) – is not such a bad hypothesis after all: clearly, there is something artificial about definitions like (81") and (82), and it would be very surprising indeed if any of them turned out to be the interpretation of a syntactic construction!

So is there a more liberal version of Lambek's Calculus that includes some of the combinations hitherto blocked without opening the door to every Ty1-definition? There are quite a few possible candidates indeed. For, as we have remarked earlier on, Lambek's Calculus can be seen as a heavily restricted version of Gentzen's calculus of propositional logic. Now, if we add more and more of Gentzen's original rules, the Calculus becomes stronger and stronger until it finally reaches the power of *Intuitionistic Logic*, a system containing many (but not all) truth-functional tautologies. (The reason why we do not get all of them has to do with the fact that types contain no other connectives than material implications: the proofs of some tautologies require detours via formulae involving negation; a classical tautology that is not intuitionistically valid is *Peirce's Law*, corresponding to the awkward type (((*et*)*e*)*e*).) Like Lambek's Calculus, Intuitionistic Logic can be given a natural interpretation in terms of Ty1-definitions: it turns out that intuitionistic definability is the same as definability by Ty1-formulae without identity. And with some minor changes of Gentzen's rules one even gets a full correspondence between Ty1-definability and (interpreted) derivability. Given such a correspondence, all kinds of weaker notions of definability suggest themselves: between Lambek's Calculus and (enriched) intuitionism, there is a whole hierarchy of systems of type combinability. Each such system comes with its own notion of definability, and they are all waiting to be explored for the purpose of restricting the notion of a possible semantic operation.

There are more direct ways of restricting definability than the ones just sketched. One may, e.g., restrict the term 'definition' to a certain sub-class of Ty1-formulae. A natural restriction of that kind may concern the number of variables that a λ-operator is allowed to bind. In fact, if this number is always 1, the resulting notion of definability will be a variant of Lambek-definability. (It will coincide with the latter if, moreover, identity is not allowed to occur in definitions.) But other restrictions of definitions to 'fragments' of Ty1 are also conceivable: one may, e.g., allow certain, but not all logical constants (or their paraphrases) to enter the definitions of semantic operations. Finally, it has also been suggested (by Mark Steedman) to replace Ty1 by some other medium (of *combinatory logic*) and then find natural restrictions on definability in that medium. Up to now, it seems that none of these possibilities has been fully investigated; a lot of work remains to be done.

We thus see that there is a tight connection between the definability approach to semantic operations on the one hand and Lambek's Calculus and related systems of type combinability on the other. One might now wonder whether there is a similar relation between definability and logicality. There is, but we will only scratch the surface of it. It can be shown that definability, though weaker than Lambek definability, is a stronger notion than logicality: (*) whatever is denoted by a Ty1-definition is at the same time logical, but there are logical objects that cannot be defined in Ty1. The first half of (*) (which we are not going to prove here) can be used as a *logicality test*: in order to show that a given operation is logical (in the sense defined in part 3), it suffices to define it in Ty1. Given this test, we can conclude that all semantic operations discussed above are indeed logical. The second half of (*) reveals that failure of definability does not necessarily imply non-logicality. (A reason for this was already mentioned in part 3: in large ontologies, certain types contain more logical objects than there are natural numbers and, consequently, than there are Ty1-definitions!) Does that also mean that non-definable objects are somehow inaccessible and obscure? Not at all. If a definition fails in Ty1, we may still be able to create a new logical language (possibly extending Ty1) in which the objects of our desire become definable. This strategy may remind of the 'definition by constants' refuted a few pages ago. But creating a new *logical* language and *defining* in it some object is no arbitrary naming of that object. If the language meets any intuitively valid standards of logic, only logical objects should be definable in it. Of course, this would have to be argued for, but then we promised not to go into these matters too deeply.

-

Exercises

10. Let B and S be constants of category *et*, and let R be a constant of category *e*(*et*). Show that (15'), i.e.:

$$[\lambda Y_{et}\ [\lambda X_{et}\ (\forall x_e)\ [Y(x) \rightarrow X(x)]\ ]\ ]\ (S)$$
$$(\ [\ \lambda x_e\ [\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]\ (B)\ (\ [\lambda y_e\ R(y)(x)]\ )\ ]\ )$$

is a Ty1-expression of category *t*. [Hint: First draw a tree indicating the structure of (15') and then use the syntactic rules of Ty1 to recursively determine the categories of its sub-expressions.]

11. In this exercise, you will have to show that the two restrictions on the Substitution Principle also apply to λ-Conversion:

(a) Find a model $((D_a)_{a\in T}, F, g)$ and an expression $\ulcorner[\lambda x\ \alpha]\ (\beta)\urcorner$ such that $[\![[\lambda x\ \alpha]\ (\beta)]\!]^{F,g} \neq [\![\alpha']\!]^{F,g}$, where α' is the result of replacing *all* occurrences (bound or free) of x in α by β. Hint: You can adapt the example used in connection with the Substitution Principle but you would still have to present a concrete model.

(b) Find a model in which

$$[\lambda x_t\ [\lambda y_t\ (x = y)\ ]\ ]\ (y)$$

and

$$[\lambda y_t\ (y = y)\ ]$$

have different extensions.

12. Assume that ***individual*** translates into the Ty1-expression:

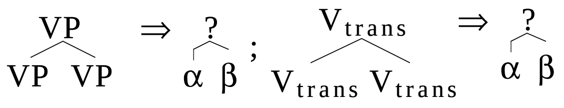
$$\ulcorner[\lambda x_e\ (x = x)]\urcorner.$$

Now use the rules given in 4.5 to translate (38) and (38') into Ty1 and show that they are equivalent to the same formula of predicate logic *without identity*.

(38) ***Every cow is four-legged.***
(38') ***Every cow is a four-legged individual.***

You can make use of all reduction principles discussed in 4.4, (including laws of predicate-logic), but you must make every reduction (including renaming of bound variables) explicit.

13. Give translation rules for VP and V_{trans} disjunction:

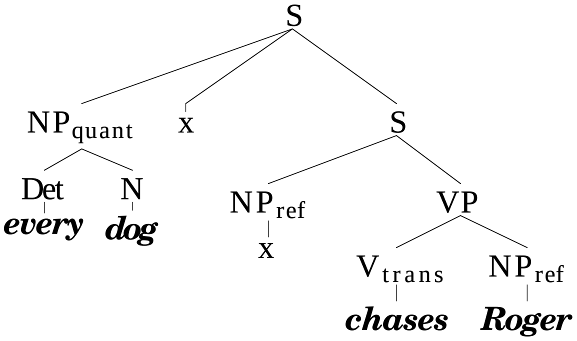


Show the correctness of your translations by applying them to (51) and (52):

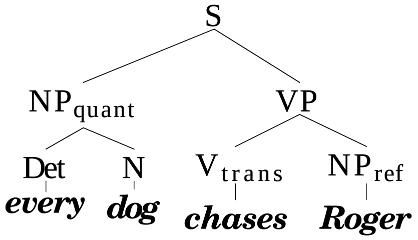
(51) ***Caroline hugs Alain or kisses Tom.***
(52) ***Caroline hugs or kisses Tom.***

14. Translate (57) and (57') and show that each is equivalent to (58):

(57)



(57')



(58) $(\forall x)\,[D(x) \rightarrow C(x,r)]$

15. Interpret the Lambek proof (!) from exercise 6 (part 3) by associating a Ty1-definition with it:

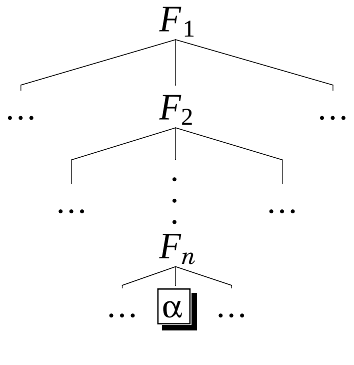
(!)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{L1: } e, e(et) \to et}{\text{L2: } e, e, e(et) \to e, et} \; ; \qquad \text{L1: } e, et \to t}{\text{L3: } e, e, e(et) \to t}}{\text{L4: } e, e(et) \to et}}{\text{L2: } e, e(et), (et)t \to et, (et)t} \; ; \qquad \text{L1: } et, (et)t \to t}{\text{L3: } e, e(et), (et)t \to t}}{\text{L4: } e(et), (et)t \to et}$$

5. Intensionality

5.1 Intensional Constructions

If all syntactic operations were extensional, the contribution that an expression α makes to the extension of a larger expression γ in which it occurs would just be its extension. In particular, if α happens to be *coextensional* with (have the same extension as) some β of the same category, then β could make the same contribution as α and could therefore play α's role in determining the extension of γ. This is a direct consequence of the principle of compositionality and the hypothesis that all operations are extensional: if, e.g., γ is $F\,(F'(\alpha,\delta),\delta')$ and F and F' happen to be extensional constructions corresponding to the combinations G and G' of extensions, then the extension of γ would be: $G\,(G'(a,d),d')$, where a, d, and d' are the extensions of α, δ, and δ', respectively; but if β's extension is a, then the extension of $F\,(F'(\beta,\delta),\delta')$ is also $G\,(G'(a,d),d')$. In fact, it suffices to assume that the operations used to construe γ are all extensional; a generalization of this kind of reasoning then shows that any α occurring in γ could be replaced by coextensional β without thereby changing γ's extension. We can even go one step further: let us call an occurrence of α in γ *transparent* if it is not within the 'scope' of any non-extensional operation, i.e. if γ has the form:



where $F_1,\ldots,F_n$ are extensional and the occurrence of α in question has been marked. We can then replace this occurrence of α by any coextensional β, thus obtaining a coextensional γ':

Transparent Substitution Principle

The result of replacing transparent occurrences of α in arbitrary γ by (an occurrence of) some β of α's category and with the same extension as α is coextensional to γ.

Instead of proving this rather obvious principle, we apply it to a simple case:

(1) ***The sun is shining and we are happy.***

If (1) happens to be true, then its extension is the truth-value 1. Given the meaning of ***and***, we know that in this case (2) and (3) must also be true:

(2) ***The sun is shining.***
(3) ***We are happy.***

Since (2) and (3) are coextensional, the Transparent Substitution Principle would give us:

(4) ***We are happy and we are happy.***

Now, although (4) may sound odd and redundant, it would certainly be true under the circumstances described and so its extension would be 1, as predicted by the principle. Similarly, we would get a correct prediction if, say, (2) were false: its extension would then be the truth-value 0, so that we could replace (2) by any other false sentence:

(5) ***2 + 2 = 7 and we are happy.***

Again the result sounds odd (incoherent), but it is clearly a false sentence just like (1), under the assumed circumstances. So the Principle of Extensional Substitution does make correct predictions, as long as we stick to extensional operations.

We have already seen that variable-binding operations are not extensional. On the other hand, there might be a way around variable binding, so that it appears that the Transparent Substitution Principle could be universally applicable. However, there exists a large class of so-called *intensional* constructions that we have so far carefully avoided and to which the above principle does not apply; and intensional operations do not seem to involve variable-binding in any obvious sense. Here is a case in point:

(6) ***Tom knows that the sun is shining.***
(7) ***The sun is shining.***
(8) ***There are exactly eight people in this room.***
(9) ***Tom knows that there are exactly eight people in this room.***

Let us assume (6) and (8) are true; then so must be (7), for what is known must be the case. So the truth-value 1 is the common extension of (7) and (8). Thus, if the Transparent Substitution Principle applied, the result (9) of replacing (7) in (6) by (8), would have to have the same extension as (6) itself, i.e. the truth-value 1. However, Tom might well be aware of the weather without thereby being informed about the number of persons occupying this room.

The failure (or, rather, non-applicability) of the Transparent Substitution Principle must have to do with the fact that (6) includes a ***that***-clause: either ***that***-clause formation or ***that***-clause embedding is an *intensional construction*, i.e. a non-extensional operation not involving variable binding in any obvious sense. Since it is of no importance which of the two operations is the intensional one, we will, for definiteness, assume that it is ***that***-clause formation. Here is another example of intensionality:

(10) ***A unicorn appears to be approaching.***

Under the (not entirely unreasonable) assumption that there are no unicorns, the extension of the noun ***unicorn*** would be the empty set and hence identical to the extension of ***married Catholic priest***. Now, for all we know, the 'Det + N' construction is extensional; so the noun phrase ***a unicorn*** must be coextensional with ***a married Catholic priest***. But replacing the former's occurrence in (10) by the latter does not necessarily result in an extensionally equivalent sentence:

(11) ***A married Catholic priest appears to be approaching.***

One could imagine all sorts of weird but realistic situations in which one of the two sentences would be true without the other one also being true. In view of the Transparent Substitution Principle, then, one of the constructions used to build (10) must be intensional. Given the fact that ***appear*** appears to be a raising verb, it is not unlikely that the intensionality has to do with lowering a quantified noun phrase. In fact, an explicit paraphrase of the raising construction reveals that it is closely related to ***that***-clause embedding:

(12) ***It appears that a unicorn is approaching.***

Many, but not all intensional constructions can be systematically related ***that***-clause embeddings in a similar way. Here is one that defies any such obvious paraphrasing:

(13) ***Alain is a good swimmer.***
(14) ***All swimmers are pianists and all pianists are swimmers.***

Let us, for the sake of the argument, imagine that both (13) and (14) were true. The truth of (14) implies that the extension of the noun ***swimmer***, i.e. the set of swimmers, coincides with that of the noun ***pianist***, the set of pianists. If the occurrence of ***swimmer*** in (13) were transparent, we would be able to conclude that replacing it by ***pianist*** would result in a true sentence:

(15) ***Alain is a good pianist.***

But (15) could well be false even if (14) was true. (In fact, (14) is quite compatible with the assumption that all good swimmers are second-rate pianists or worse, as long as they are pianists at all.) This time the intensionality effect must be due to the 'Adj + N' construction. It can neither be interpreted as intersection (as it was in the fragment in 4.6) nor as any other combination of extensions: otherwise we would be able to infer (15) from (13) and (14).

The fact that modifier constructions like the one just discussed cannot be directly related to ***that***-clause embeddings has sometimes been used to show that they are not intensional after all (but instead reveal a complicated argument structure of nouns, verbs, or whatever is modified). Nevertheless, a classical analysis treats sentences like (13) and (15) as involving an intensional combination of adjective and noun. And even though this relation cannot be directly reduced to ***that***-clause formation, the technique developed for analyzing the latter naturally carries over to modifier constructions. It is this general technique that we will now look at.

5.2 Information and Intension

We have seen that, even if we ignore variable-binding operations, we cannot do with extensions alone as the semantic values in a compositional interpretation: certain *intensional* operations like ***that***-

clause formation require different entities. Let us call them *intensions*. It thus seems that instead of a:

Naive Principle of Compositionality
The extension of a complex expression is uniquely determined by the extensions of its (immediate) parts and their mode of combination.

we need something more sophisticated like:

Frege's Original Principle
The extension of a complex expression is uniquely determined by the extensions or the intensions of its (immediate) parts and their mode of combination.

(We will, for the rest of our discussion, simply ignore variable-binding operations.) As long as we do not know what intensions are, this modification of the Naive Principle of Compositionality is, of course, not very helpful. What we need in addition is a definition of the notion of intension. From the preceding section we know that we may concentrate on ***that***-clauses as the model case of an intensional construction. We will therefore only be concerned with intensions of *sentences*. In order to get a first idea about what they could be, let us briefly reconsider the substitution argument (6) - (9) concerning Tom's knowledge. What is it that distinguishes (7) from (8) and thereby precludes the substitution of the former by the latter? To see this, we may compare the example to a case in which a similar argument does go through:

(16) ***Alain knows that a little bird with short wings and a long beak made that noise.***
(17) ***A little bird with short wings and a long beak made that noise.***
(18) ***The noise was made by a little bird with a long beak and short wings.***
(19) ***Alain knows that the noise was made by a little bird with a long beak and short wings.***

It is hard if not impossible to imagine that (16) - (18) are true without (19) also being the case. (In fact, (17) and (18) are not even needed as explicit premises: (17)'s truth follows immediately from (16); and (18) must be true if (17) is.) It thus seems safe to replace (17) by (18): unlike (7) and (8), (17) and (18) must be *intensionally equivalent,* i.e. they must have the same intension. An obvious difference between the two pairs of sentences is that the latter two sentences are about the same subject matter, whereas (7) concerns the weather and (8) the population of a certain room. (17) and (18), on the other hand, not only agree on their subject matter, viz. the source of a certain noise; they also say the very same thing about it, only in (slightly) different words: whether we used (17) or (18) to locate this source does not make any difference; the two sentences contain the same information. We are thus led to the following hypothesis:

Frege's Criterion
Two sentences are intensionally equivalent iff they contain the same information.

If, as we will assume, the criterion is correct, we may even identify the intension of a sentence with the information it contains. So what is the information contained in a sentence? Indeed, what is information in general? Though it may be of some intrinsic interest, Frege's Criterion still does not tell us what precisely intensions are. Instead we are faced with a new problem, viz. that of defining the notion of information. However, the criterion does change our perspective: in order to look for intensions in intensional contexts, we may look at extensionally construed sentences and the information they contain. Here is a list of examples that will set us on the right track:

(20) ***Exactly four coins were tossed.***
(21) ***At least one of the four coins tossed was heads.***
(22) ***At least one of the four coins tossed was tails.***
(23) ***Exactly two of the four coins tossed were heads.***
(24) ***Exactly two of the four coins tossed were tails.***

In order to see what the information contained in each sentence is, it helps to realize that, in a certain sense, some of the above sentences differ in the *amount* of information they contain, i.e. in their *informativeness*. Thus, e.g., (20) is obviously less informative than any other of the sentences, (21) is more informative, but less so than (23), which is itself as informative as (24), etc. So information is something that can be compared or even measured. There are, in fact, two different ways of comparing information, a *quantitative* and a *qualitative* one. The difference can be seen by looking at (21) and (22). In a (quantitative) sense, the amount of information contained in each of them is the same: (21) reveals as much (or little) about a given situation as (22) does. But surely, the two sentences reveal different things, so that it does not make much (qualitative) sense to say that one is more informative than the other. Similarly, (23) is quantitatively more informative than (22), but there seems to be no straightforward way of comparing their information qualitatively. On the other hand, (21) is more informative than (20), in both a qualitative and a quantitative sense; similarly, (23) is as informative as (24), in either sense.

There is a natural (and presumably familiar) way of modelling the notion of quantitative informativeness by counting ('favourable') *cases*: (20) is less informative than (21), because there are more cases in which four coins are tossed than there are cases in which one out of four tossed coins is heads; and since there are as many cases in which at least one coin is heads as there are cases in which at least one is tails, (21) is as informative as (22), again in the quantitative sense. So the cardinality $|\Sigma_\varphi|$ of the set Σ_φ of all cases covered by a sentence φ is a measure for φ's quantitative informativeness: the larger $|\Sigma_\varphi|$ is, the less informative is φ.

What are *cases*? A natural guess is that a case is something like a *situation* described by the sentence in question. If so, it is clear that these would have to be hypothetical, rather than real, situations: the fact that (20) is less informative than (22) does not depend on how many tossings of four coins with one or more tails have *actually* been performed. But what, then, are *hypothetical situations*? They would, e.g., have to be specific enough to allow for a distinction between different situations with, say, exactly two coins being tails; if we did not take this difference into account, (24) would be true in as many situations, and consequently as informative as:

(25) ***Exactly one of the four coins tossed was tails.***

But for any choice of four coins, there are only four ways of satisfying (25), whereas (24) could come out six different ways. So $\Sigma_{(24)}$ should be larger than $\Sigma_{(25)}$, which means that every hypothetical situation should be specific as to which coin has been tossed with which result. This kind of specificity is needed for the analysis of these particular examples. It is not hard to imagine that, in the analysis of other examples, other kinds of specificity of arbitrarily fine degrees would be needed. Hypothetical situations, then, should be as specific as possible – just like real situations, but unlike descriptions of (real or hypothetical) situations that always have to leave out some details. The hypothetical and maximally specific character of situations immediately leads to a problem for the determination of quantitative informativeness: there may be infinitely many hypothetical situations, in which case counting them can be difficult. But fortunately we are not concerned with the quantitative notion of informativeness here, because it is obviously the qualitative one that we are after.

The above account of quantitative informativeness in terms of numbers of favourable cases naturally leads to a characterization of the qualitative notion of informativeness. To see this, we merely have to look at (21) and (22) again. The reason why they appear to be equally informative in the quantitative sense is there are as many favourable cases for each of them: it seems reasonable to assume that $|\Sigma_{(21)}| = |\Sigma_{(22)}|$. But, clearly, the two sentences differ in qualitative informativeness, because they do not describe exactly the same cases: because $\Sigma_{(21)} \neq \Sigma_{(22)}$. In general, then, two sentences φ and ψ differ in qualitative informativeness if $\Sigma_\varphi \neq \Sigma_\psi$. If, on the other hand, φ and ψ are true in exactly the same situations, it is hard to imagine that could contain different information. We are thus led to:

Carnap's Idea
The information contained in a sentence is the set of hypothetical situations (specific states of affairs, possible worlds, indices…) that it correctly describes.

(The parenthetical list offers some traditional terminological alternatives.) According to this idea, the intension of a sentence can thus be identified with a set of situations. Since this set comprises whatever the sentence says, whatever information it contains, it will also be called the *proposition* expressed by the sentence. A more dramatic way of putting the above idea is thus: propositions are sets of possible worlds. We will now see how to make use of this idea.

5.3 Two-sorted type theory

The sentence

(26) ***Bill loves Coco.***

correctly describes all situations i in which Bill loves Coco. Its intension is thus the set:

(26') $\{i|$ Bill loves Coco in $i\}$,

where 'i' is used as a variable ranging over (hypothetical and real) situations or indices. If we are going to use this proposition in determining the extensions of complex expressions like

(27) ***Nobody doubts that Bill loves Coco.***

we should try and construe the intension (26') in some systematic, maybe even compositional way. There are several ways of doing so and we will only look at one of them, viz. the technique of indirect interpretation. We will thus see how a sentence like (26) can be (compositionally) translated into a logical formula denoting the proposition the sentence expresses. Now, the notation used in (26') is already pretty close to the formula we will eventually arrive at and we will get there by employing essentially the techniques discussed in part 4: the intension of (26) will be denoted by a formula starting with '$[\lambda i\ldots$' – like the set abstraction '$\{i|\ \ldots$' used in (26'); the referential NPs will be translated as constants of type e; and they will be combined with the translation of the verb by successive functional application. However, before we can go into the details of this procedure, we must decide what to do with the qualification 'in i' occurring in (26'). It is obvious that we cannot just forget about it; for without it, the set defined in (26') would either be empty or consist of all situations whatsoever, depending on whether or not Bill does indeed love Coco. But (26') suggests that whether the latter relation obtains is itself contingent upon the particular situation: maybe he loves her here and now, but maybe no longer so in two years or if, in some hopefully unreal situation, he turns into a frog. In other words, the set of pairs (a,b) such that a loves b varies from one situation to another. We have conceived of this set as the *extension* of the verb ***loves***, and we will continue to do so; but we will, from now on, no longer ignore the fact that this extension depends on various circumstances. In terms of our indirect interpretation this means that ***loves*** should no longer be translated by a constant of type $e(et)$ but rather by something more complex like the formula:

(28) L_i

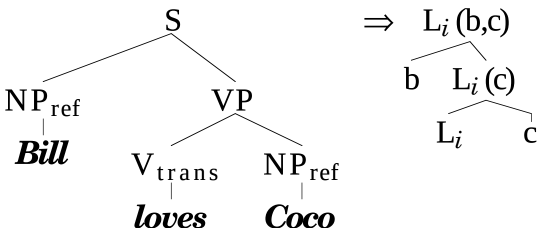
that is supposed to denote the set of pairs (a,b) such that a loves b in the situation i (or, more accurately, in the situation denoted by the variable i), i.e. the extension of ***loves*** in that situation.

Now, what kind of formula is (28)? Since it is supposed to denote a set of ordered pairs, it should come out as being of type $e(et)$, but what is its internal structure? Well, the loving relation is that which depends on the situation (denoted by '*i*'). So, mathematically speaking, love is a function from situations to sets of ordered pairs of individuals. (Surely, this shows that the mathematical perspective on life misses some of its excitement.) The 'L' in (28) may thus be understood as a constant of type $s(e(et))$, where the 's' stands for the *type of situations*, which we will assume to be an additional basic type s, on top of e and t. Consequently, '*i*' is a variable of type s, and (28) is but a notational variant of:

(28') $L(i)$,

i.e. the result of combining the expressions 'L' of category $s(e(et))$ and '*i*' of type s by functional application; clearly, this result is of category $e(et)$, as required. If we now take (28) or (28') to be the translation of ***loves*** and continue to assume that proper names like ***Bill*** and ***Coco*** translate into constants of type e, we can use our old compositional rules of indirect interpretation, i.e. rules (27) and (30) of part 4, to obtain the following compositional translation of (26):

(29)



Being of type t, the resulting formula does not refer to the intension of (26), i.e. the set (26'), but rather denotes a truth-value – which one depends on i (or, again, what '*i*' refers to). But clearly the desired *st*-expression can be obtained from (29) by abstracting from the variable i:

(30) $[\lambda i\ L_i(b,c)]$

(30) obviously denotes the set of situations in which Bill loves Coco and resembles the meta-linguistic definition (26') of that proposition. Note that the relation between the translation of (26), as it appears in (29), and the formula (30) denoting the corresponding intension is analogous to the one between 'L_i' and 'L': in either case, the first formula denotes the *extension* in some fixed situation (denoted by '*i*'), whereas the second formula denotes that function f assigning to each situation the extension in that situation. Since, in the case of (30), this function is the *intension* of the sentence, we may generalize this notion to the other case and say that the constant 'L' denotes the intension of the transitive verb ***love***,

whereas the complex expression (28') denotes its extension (in the situation denoted by i). We will, indeed, generalize the distinction to arbitrary natural language expressions Δ by defining the *intension* of Δ to be that function f that takes every situation to Δ's extension in that situation; and if Δ translates into the logical formula α denoting α's extension in the situation indicated by the type-s-variable i, then $\ulcorner[\lambda i\ \alpha]\urcorner$ will denote α's intension. Applying this general observation to the translation (28') of ***love***, we find that its intension is denoted by the formula

(31) $[\lambda i\ \mathrm{L}(i)]$,

which happens to be logically equivalent to the constant 'L', by the law of η-conversion. It thus suffices to take as the translation of a natural language expression a formula denoting its extension and indicating, by the presence of a variable i of type s, its situational dependence. In intensional constructions this variable will then be bound by a λ-operator, thereby obtaining a formula denoting the intension, as required by the construction. We will now give the precise definitions of this procedure.

Since we have a new basic type, we also have a new set of types altogether. We call them *two-sorted types* and they are defined by adding to the original definition of an extensional type the additional phrase structure rule '$S \rightarrow s$'. Clearly, every intensional type is a two-sorted type, and those two-sorted types – like s, $s(e(et))$, and st – that are not extensional will be called *intensional types*. Similarly, we let a *two-sorted ontology* to be a family of sets D_a, for any two-sorted type a, and define it just like in section 3.2, with the additional clause

$$D_s = S,$$

where S is an arbitrary, non-empty set. The situations thus play a role analogous to that of individuals. In particular, situations are not be further analyzed into their components, parts, or whatever and, unlike truth-values, what they are and how many of them there are, depends on the ontology. (The situations thus form a second *sort* of non-logical, basic objects.) These assumptions are mainly made for simplicity.

Now for the language *Ty2* of *two-sorted type theory*. Its basic expressions are the variables Var_a and constants Con_a of arbitrary two-sorted types a; about them we make the same assumptions as in

Ty1. The syntactic operations of Ty2 are the same as those of Ty1, i.e., Abstraction, Application, and Identity; needless to say that now they extend to arbitrary two-sorted types. Moreover, the notions of a *two-sorted variable assignment* and that of a *tow-sorted interpretation* will be defined in the obvious way, i.e. as functions from all two-sorted variables and constants to denotations of the appropriate types. And finally, the syntactic operations are interpreted exactly as in Ty1. Using these definitions, it is clear that, e.g.,

(32) $[\lambda j_s\, (i_s = j)]$

is a Ty2-formula of category *st* and that it denotes (the characteristic function of) the singleton set $\{g(i)\}$ in any two-sorted model M.

Given the basic syntactic and semantic definitions of two-sorted type theory, it is not hard to see that all important logical laws and equivalences discussed in part 4 carry over from Ty1 to Ty2. In particular, the Principles of λ-Conversion, η-Conversion, Substitution, and Alphabetic Variants as well as the Strong Normalization Theorem all hold in two-sorted type theory; and whatever is valid in predicate logic (of whatever order) or in one-sorted type theory remains valid in Ty2. We will not prove any of these assertions here but simply trust that the difference between one-sorted and two-sorted type theory is not dramatic enough to cause any significant deviance in logical behaviour.

We are now in a position to sketch a modification of the translation algorithm presented in part 4. Apart from the addition of some intensional construction, the main difference between the old and the new translations consists in the sensitivity of extensions to situations. We have already seen how this can be done in the case of such lexical expressions as the transitive verb ***love***. And this procedure will work for almost any lexical expression: unless otherwise stated, we assume that the translation of a lexical expression δ is a Ty2-formula of the form

(33) $c(i)$,

whenever δ's category corresponds to a (two-sorted) type a and $\ulcorner c \urcorner$ is a constant of type sa. As before, i is a variable of type s, but we actually have to say which one; for we want to make sure that the index-variable introduced by a lexical expression is the same as the one bound by the λ-prefix in the translation of an intensional construction. The easiest way to guarantee this is to once and for all fix one variable of type *s* and agree that it is going to be the only one that will ever occur in the translation of an expression. This is essentially what we will do, although we will sometimes rename the variable for better readability. So let us assume that the meta-variable '*i*' denotes the variable x_s^0, i.e. the 0th variable of type *s*. (33) then gives the general form of the translation of a lexical expression as consisting of a constant (denoting the intension of that expression) applied and this variable x_s^0.

There are certain exceptions to the scheme (33) of lexical translation most of which have already been discussed in part 4. In particular, words that can be lexically decomposed will be translated by formulae more complex than (33). Of course, we now have to add all situational dependencies, so that the translation (34) of ***girl*** (briefly discussed in section 4.5) becomes the slightly more complicated (34'):

(34) $[\lambda x_e\ [F(x)\ \&\ P(x)\ \&\ C(x)]\ ]$

(34') $[\lambda x_e\ [F_i(x)\ \&\ P_i(x)\ \&\ C_i(x)]\ ]$

(We have again made use of the notational convention of writing an argument as a subscript.) The difference between the constants in (34) and their primed counterparts in (34') is, of course, their type: the former are ordinary (first-order) predicates of type *et*, and the latter, being of category *s*(*et*), denote functions from situations to (characteristic functions of) sets of individuals; such functions that make the extension of a predicate depend on a situation will from now on be called *properties* (of individuals).

Bearing the difference between (34) and (34') in mind, it is fairly easy to translate a given extensional lexical decomposition into a two-sorted one. In the case of the translations of determiners, no change will be necessary, because their extensions never depend on the situation: ***every*** always denotes the subset-relation, ***no*** is disjointness, etc. So the very same Ty1-formulae that we have used to translate determiners can also be used as their Ty2-translations.

Proper names are a similar exception to the scheme (33). We have already seen in (29) that, in order to receive the desired translation of ***Bill loves Coco***, ***Bill*** and ***Coco*** should be translated by constants of type *e* – rather than by constants of type *se*, as the scheme (33) would suggest: whichever situation we are talking about, the name ***Bill*** always refers to the same person, Bill. (This is not to say that ***Bill*** may not be used to refer to another person; however, this would constitute a different usage of the name. We will return to this in an exercise.) So, just like determiners, proper names maintain their old Ty1-translations.

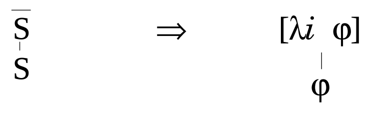
What proper names and determiners have in common is the situational independence of their extensions: they are (referentially) *rigid*. Other examples of rigid expressions include the connectives ***or*** and ***and*** (if they are interpreted as lexical items always denoting the same truth-tables) and possibly so-called *natural kind terms* as ***tiger*** or ***birch***. However, the whole issue of which lexical expressions are rigid is rather complex and we will not go into this debate here. It should only be mentioned that a rather popular way of modelling referential rigidity consists in translating the expression in question according to the scheme (33) and then assume a *meaning postulate* to the effect that ‘c’ is interpreted as a constant function:

(35) $(\forall i_s)\ (\forall j_s)\ (\ \mathrm{c}(i) = \mathrm{c}(j)\ )$

(Note that (35) contains more than one world variable; this does not matter, because it is not itself the translation of a natural language expression.) Clearly, (35) implies that the extension of the lexical item translated by (33) does not depend on any particular situation. In that sense (35) guarantees rigidity. We will, however, usually get the same effect by immediately translating the rigid expression into a constant of the same category as the entire formula (33), i.e. the category of ‘c’ minus the initial *s*. That way we avoid a meaning postulate – at the cost of admitting a certain number of exceptions to (33) as a general scheme of translating lexical items.

Now that we know how lexical expressions are translated, we can go on to the translation of syntactic rules. Here matters are even simpler. As the example (29) already suggested, there is no need to change our original translation rules from part 4. All we have to do is to add some intensional constructions, translating them by abstraction from the situation. Here is one way of adding ***that***-clauses:

(36)

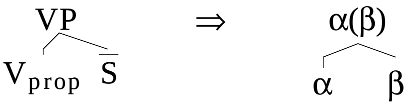


The $\overline{\mathrm{S}}$ in (36) is a ***that***-clause. The idea is that, whereas the sentence itself denotes a truth-value, the ***that***-clause denotes the intension of this sentence. Since ***that***-clauses usually appear in intensional constructions, the effect of this rule is that the ***that***-clause contributes a proposition to the extension of any larger expression – just like Frege's Original Principle would have it. Clearly, the '*i*' is the 0th variable of type *s* and the type corresponding to $\overline{\mathrm{S}}$ is *st*.

5.4 Propositional attitudes

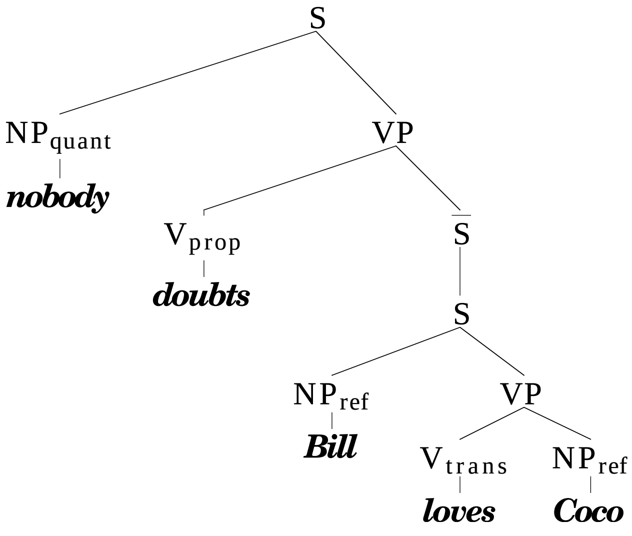
In order to be able to apply (36), we need a ***that***-clause embedding construction. In the present context, the most straightforward one is that combining (*propositional*) *attitude verbs* like ***believe***, ***know***, or ***doubt*** with a ***that***-clause. What these verbs have in common (and what makes them attitude verbs in our terminology) is that they need a subject and a ***that***-clause to complete a sentence. The most obvious categorization of these verbs is to regard them as expressing binary relations between individuals (denoted by their referential subjects) and propositions. We thus assume that the type corresponding to the category $\mathrm{V}_{\mathrm{prop}}$ of attitude verbs is (*st*)(*et*). We then add the following rule:

(37)



Let us apply these new rules to an example mentioned above:

(38)



The translation of the embedded sentence has already been given. It is obtained by translating ***loves*** according to scheme (33), treating proper names as rigid exceptions and combining the constituent translations by the old rules (27) and (30) of part 4. The result is:

(39) $L_i(b,c)$

Now (36) gives us the translation of the ***that***-clause, i.e. the formula (30) denoting the intension of the sentence ***Bill loves Coco***; we repeat it solely for the readers' convenience:

(30) $[\lambda i\ L_i(b,c)]$

Before we can apply (37), we must find a translation for the verb ***doubt***. Since it is of category V_{prop}, the translation should be of type (*st*)(*et*). The remaining question is whether it should be subsumed under the scheme (33). If not, we must either (i) give some decomposition for it or (ii) argue that it is rigid. While (i) may be possible, we will not attempt it here; one suggestion will be discussed in an exercise. On the other hand, (ii) is certainly impossible: who doubts what, i.e. the extension of the verb ***doubt*** does depend on the situation. We therefore conclude that (33) applies yielding (40) as a translation of ***doubt***:

(40) $D(i)$,

where 'D' is a constant of category *s*((*st*)(*et*)) and *i* is as usual. Now rule (36) can apply to combine (30) and (40) into:

(41) $D_i([\lambda i\ L_i(b,c)])$

(Note that we have again switched to the subscript notation.) As one can easily check, this formula is of category *et*, as is appropriate for the translation of a VP. It should be noted that the two occurrences of the variable '*i*' in argument position are quite independent of each other in the sense that the first one (as a subscript to 'D') is free in (41), whereas the second one (the subscript to 'L') is bound by a λ-operator and can therefore be renamed without changing the content (but certainly the readability) of the formula:

(41') $D_i([\lambda j\ L_j(b,c)])$

Now, although this resulting formula contains more than one type-*s*-

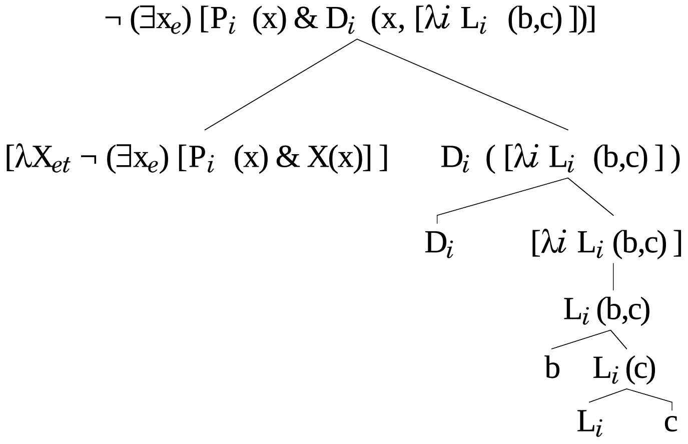
variable, this does not matter because it is not the ‘official’ translation of any expression, i.e. not the result of applying the algorithm sketched above, but merely a formula that happens to be equivalent to such a translation. And these equivalent formulae do sometimes contain more than one variable of type *s*, as this very example shows!

In order to finish the example, we must still give a translation for the quantified subject. Here is an *ad hoc* suggestion:

(42) ***nobody***' := $[\lambda X_{et} \neg (\exists x_e) [P_i(x) \,\&\, X(x)]]$,

where ‘P’ is a constant of category *s*(*et*) denoting the property of personhood. Using (42), we can now combine the subject and predicate of (38) by the familiar rule (28) from part 4. The following tree illustrates the whole translation process; logical reductions have been made as early as possible:

(42)


¬ (∃xe) [Pi (x) & Di (x, [λi Li (b,c)])]
[λXet ¬ (∃xe) [Pi (x) & X(x)]]
Di ([λi Li (b,c)])
Di
[λi Li (b,c)]
Li (b,c)
b
Li (c)
Li
c

Mixing object-language and meta-talk, we can say that the topmost formula correctly describes (is true of) all situations *i* in which there is no individual that is both a person and stands in the relation of doubt to the proposition that is true of all situations in which Bill loves Coco or, for short, to the proposition that Bill loves Coco. To say that the sentence is *true* or denotes the truth-value 1 then amounts to saying that the momentary, actual situation is one that it accurately describes. In this sense, free occurrences of *i* can be understood as referring to the situation that the sentence is about. Under normal circumstances and as long as nothing else has been said this will simply be the situation of utterance.

Does the translation (42) give a correct account of the meaning of the attitude sentence (38)? This is a tricky question and we will not give a definite answer to it. But we will point out at least some of the problems that have been raised against the so-called *possible worlds approach* to propositional attitudes and intensional constructions in general. The most celebrated argument has to do with the Substitution Principle. We know that we may replace one formula by an extensionally equivalent one without changing the extension of the host expression – provided that this substitution does not change any variable's status of freedom or bondage. Let us first see why the proviso saves us from the kind of fallacy that motivated the introduction of intensions. We may, e.g., assume that the following sentences are all true:

(43) ***Alain is a boy.***
(44) ***Alain is asleep.***
(45) ***Tom thinks that Alain is a boy.***

Our translation rules yield the following three formulae:

(43') $B_i(a)$
(44') $A_i(a)$
(45') $T_i(\, t, [\lambda i\ B_i(a)]\,)$

(For reasons that will become clear in a second we have not renamed bound variables.) Now consider the result of substituting (43) in (45) by (44) together with our translation of it:

(46) ***Tom thinks that Alain is asleep.***
(46') $T_i(\, t, [\lambda i\ A_i(a)]\,)$

Now, doesn't our algorithm predict that we can conclude (46) from (43) - (45)? After all, we do get the former's translation by substitution of (43') by (44') and we have assumed the latter to be extensionally equivalent, both denoting the truth-value 1. It's the variable condition, of course: (43') contains a free occurrence of i which is bound by a λ-operator in (45'). The fact that Alain is both a boy and asleep *in the situation referred to by 'i'* does not guarantee that he is asleep in exactly those (real and hypothetical) situations in which he is a boy. However, the latter would have to be the case for the substitution to be correct: in order to get (46') as a valid inference from (45') via the Substitution Principle, one would have to replace '$[\lambda i\ B_i(a)]$' by '$[\lambda i\ A_i(a)]$'.

The above example clearly shows the merits of renaming bound occurrences of the variable '*i*'; for had we rewritten the translation (45') as (45"), we would never have been tempted to apply the Substitution Principle:

(45") $T_i(t, [\lambda j\ B_j(a)])$

Another conclusion to be drawn from this example is that there is only one kind of failure of the Substitution Principle, viz. the confusion of bound and free occurrences of variables: under the present view and contrary to our informal introduction of the phenomenon, intensionality is a special case of variable binding. In this sense the possible worlds approach involves a reductionist theory of intensional constructions. Intensionalists see a fundamental error in this denial of a special status of intensionality.

There are stronger points to be made against the above treatment of propositional attitudes. The fact that substitution works whenever two expressions extensionally coincide in all possible situations leads to highly implausible results. In the above case the assumption that Alain is asleep whenever he is a boy (and vice versa) may be rejected on material grounds, but in the following example there is no escape:

(47) ***Tom is asleep.***
(48) ***Every individual that is [identical to] Tom is asleep.***
(49) ***Alain thinks that Tom is asleep.***

We want to understand ***individual*** as expressing a 'vacuous' property; following exercise 12 of part 4, we therefore suggest the translation rule:

(50) ***individual'*** $:= [\lambda x_e\ (x = x)]$,

Our rules then give us the following translations:

(47') $A_i(t)$

(48') $(\forall x_e) [(x = t) \rightarrow A_i(x)]$

(49') $T_i(a, [\lambda j\ A_j(t)])$

(This time we did rename the bound occurrences of '*i*'.) Let us now assume that (47) - (48) are true and therefore extensionally equivalent. In fact, according to our translation, the truth of (47) immediately implies the truth of (48) and vice versa: the two formulae (47') and (48') are logically equivalent. How about substituting (48) for (47)? The result would be:

(51) ***Alain thinks that every individual that is [identical to] Tom is asleep.***

(51') $T_i(a, [\lambda j\ (\forall x_e)\ [\ (x = t) \rightarrow A_j(x)\ ]\ ]\ )$

Given the truth of (49), would (51) also have to be true? Maybe not. Maybe Tom's being asleep amounts to every-individual-who-is-Tom's being asleep without Alain realizing it? Given a bad case of underdeveloped logical skills, one might actually doubt the validity of such a conclusion. However, our formalization forces us to accept it: since (47') and (48') will get the same truth-value no matter what the extension of 'i' may be, the functions taking every possible extension of this variable to the corresponding extension of the formulae will also coincide. So the logical equivalence of (47') and (48') immediately implies the logical equivalence of the following two formulae, i.e. the translations of the ***that***-clauses:

(52) $[\lambda j\ A_j(t)]$

(53) $[\lambda j\ (\forall x_e)\ [\ (x = t) \rightarrow A_j(x)\ ]\ ]$

It is clear that this kind of reasoning does not depend on any specific features of this example: whenever two sentences are logically equivalent, they have the same intensions and are thus inter-substitutable, even in intensional contexts. This fact has been seen as the major defect of the above kind of treatment of propositional attitudes.

How disastrous is the substitutivity of logical equivalents? In the attempt to answer this question, advocates of possible worlds semantics have sometimes tried to explain away the phenomenon of logical ignorance by arguing that, strictly speaking, a sentence like (49) does imply (51), because the very fact that Alain believes Tom to be asleep shows that he believes everyone who is Tom to be asleep – even though he himself may not be aware of the fact that the *sentences* (47) and (48) express the same *proposition*. We will not go into the details of this argumentation, let alone evaluate it.

Here is another, somewhat more exotic argument against identifying the objects of propositional attitudes with sets of situations. To follow it, the reader is asked to pick an arbitrary proposition p together with his or her favourite propositional attitude A and imagine a (possible) situation s_p in which p is the only proposition that anybody (or anything) bears A to. Now, p might have been the proposition expressed by

(51) and A may be the relation that holds between a person x and a proposition q in a situation s iff, in s, x utters a sentence expressing q; the situation s_p would then have to contain one or more individuals uttering (51), or (49), or something co-intensional with it without anybody else uttering anything else. Or p could be a set consisting of just one situation s and A could be the relation that always holds among Tom and all singleton sets but among nothing else; then s_p could be any situation whatsoever. The point is that, no matter how p and A are determined, there seems to be no intuitive objection against imagining such situations s_p. However, given the identification of propositions and sets of worlds, one can show that, at least for some choices of p and A, no such s_p exist. The reason has to do with the relative cardinalities of the set of possible situations and its power set, the set of propositions; and the proof, which we will now sketch, is a variant of Cantor's diagonal argument.

Let us, for the sake of this argument, fix an attitude A and imagine that, for arbitrary propositions p we had found a (fixed) situation s_p such that the only proposition that is Aed in s_p is p. We first observe that, (!) for any propositions p and q, $s_p = s_q$ implies that $p = q$. Next we consider the 'diagonal' proposition d of all situations s_p such that $s_p \notin p$:

(54) $d := \{s \mid \text{for some proposition } p,\ s = s_p \text{ and } s \notin p\}$

Since d is itself a proposition (that we might have determined in another, more straightforward way), there is a situation s_d in which only d is Aed. We now wonder whether d contains s_d as an element. If so, s_d would have to satisfy the condition on arbitrary s in (54) and thus $s \notin p$ for some p such that $s_d = s_p$. However, by (!), this p must be d itself, so that $s \in p$. The assumption that s_d is in d therefore leads to a contradiction. Does that mean that $s_d \notin d$? In that case we would have found a proposition p (= d) such that $s_d = s_p$ and $s_d \notin p$. So s_d meets the defining condition in (54) and is therefore in d – again a contradiction. So our initial assumption about the existence of arbitrary s_p must be wrong. We have thus proved that the following (schematic) Ty2-formula is a contradiction:

(u) $(\forall p_{st})\,(\exists i)\,(\forall q_{st})\,[\,\alpha_i(\beta,q) \leftrightarrow (p = q)\,]$,

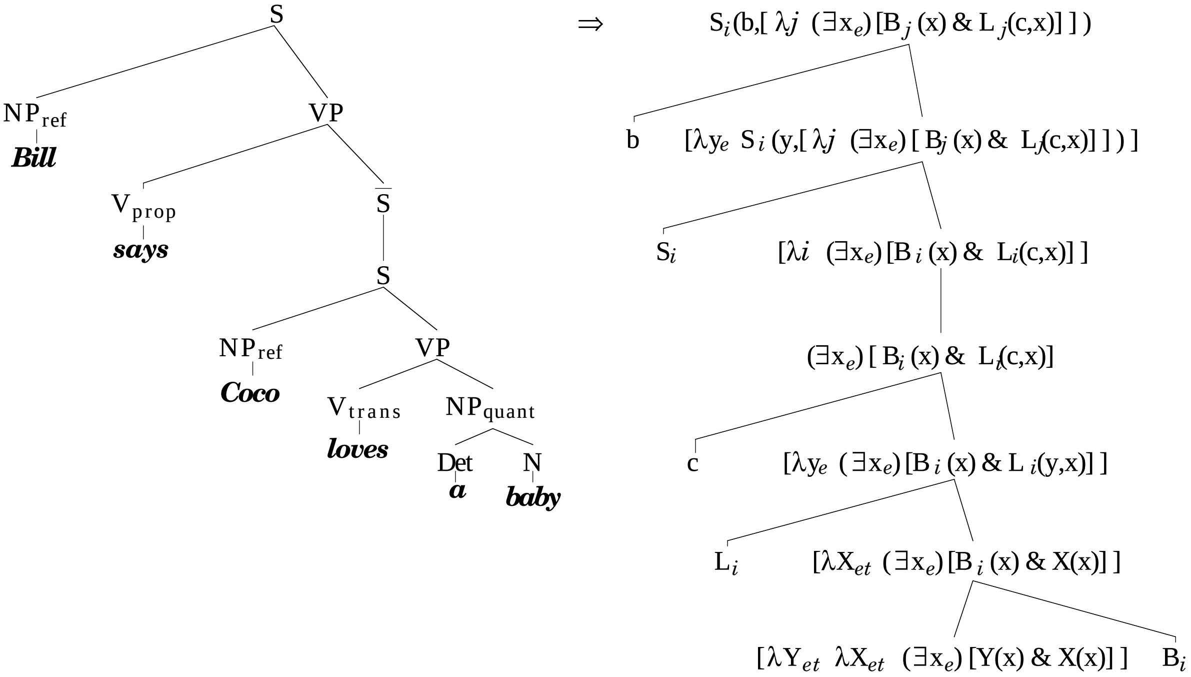
whenever α and β are expressions of categories $s((st)(et))$ and e (and no variable confusions arise). It has been argued that there is no intuitive reason to reject the possibility of situations s_p in which an arbitrary proposition p is the only thing to be believed, claimed, denoted, etc. Since the framework forces us to deny this universal possibility, something must be wrong with it. As a careful analysis would show, the above argument mainly rests on the assumption that there are more propositions than situations. Hence it is this assumption, which in turn follows from the identification of propositions with (arbitrary) sets of situations, that should be given up – or so the story goes. Again, we will not try to evaluate this argument against the very spirit of possible worlds semantics but merely point to its existence.

Before we can get to a third objection, we better look into some interactions of our treatment of propositional attitudes with certain other rules of grammar. More specifically, it is very instructive to consider ***that***-clauses containing one or more quantified noun phrases:

(55) ***Bill says that Coco loves a baby.***

The most straightforward analysis of (55) is certainly:

(56)


S
NP$_{ref}$
Bill
VP
V$_{prop}$
says
$\bar{S}$
S
NP$_{ref}$
Coco
VP
V$_{trans}$
loves
NP$_{quant}$
Det
a
N
baby
⇒
$S_i(b,[\lambda j\ (\exists x_e)[B_j(x) \& L_j(c,x)]])$
b
$[\lambda y_e\ S_i(y,[\lambda j\ (\exists x_e)[B_j(x) \& L_j(c,x)]])]$
S_i
$[\lambda i\ (\exists x_e)[B_i(x) \& L_i(c,x)]]$
$(\exists x_e)[B_i(x) \& L_i(c,x)]$
c
$[\lambda y_e\ (\exists x_e)[B_i(x) \& L_i(y,x)]]$
L_i
$[\lambda X_{et}\ (\exists x_e)[B_i(x) \& X(x)]]$
$[\lambda Y_{et}\ \lambda X_{et}\ (\exists x_e)[Y(x) \& X(x)]]$
B_i

As one would expect, according to this reading, the sentence expresses that the proposition said by Bill consists of all situations in which there is a baby that Coco loves. What does it mean for a person to say a *proposition*, in our technical sense of the term? A first guess is provided by the following meaning rule:

(57) The relation of saying (i.e. the extension of the verb ***say***) holds among an individual *x* and a proposition *p* in a situation s iff *x* utters a sentence (of some language known to *x* in s) whose intension is *p*.

Following this rule, Bill would have had to utter something like a sentence under (58) in order for (56) to be true:

(58) (e) ***Coco loves a baby.***
(f) ***Coco aime un bébé.***
(g) ***Coco liebt ein Baby.***
(l) ***Amat infantem Coco.***

Moreover, it seems that (57) is too strict in view of the following alternative utterances of Bill's that would certainly also verify (55):

(58') (e) ***Coco loves a fat baby.***
(f) ***Coco aime un gros bébé.***
(g) ***Coco liebt ein dickes Baby.***
(l) ***Amat infantem crassum Coco.***

The difference between (58) and (58') is one of (qualitative) informativeness. This observation suggests the following revision of (57):

(57') The relation of saying holds among *x* and *p* in s iff *x* utters a sentence whose intension is a (not necessarily proper) subset of *p*.

One feature of this analysis of (55) is of particular interest to us here: if we compare different situations in the proposition that Bill is said to bear the saying relation to, it turns out that they all contain babies that are loved by Coco, but that in different situations Coco may, of course, love different babies. (Recall that situations are possible situations.) In particular, according to analysis (56), (55) does not say that Bill made some specific claim like:

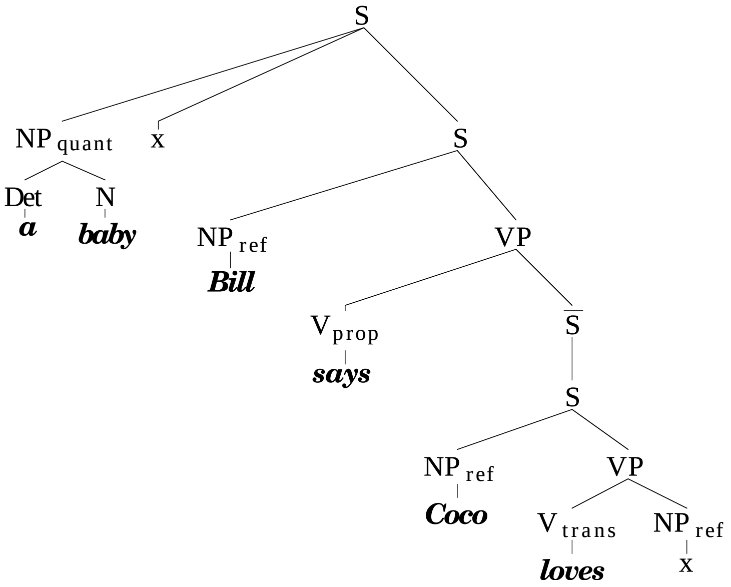
(59) ***Coco loves Zoë.***

(This time the translations are left to the reader.) In this sense, what Bill says according to (56) is *unspecific* with respect to the baby loved by Coco. On the other hand, it seems clear that Bill's utterance of (59) would also make (55) true, provided that the name ***Zoë*** refers to a baby.

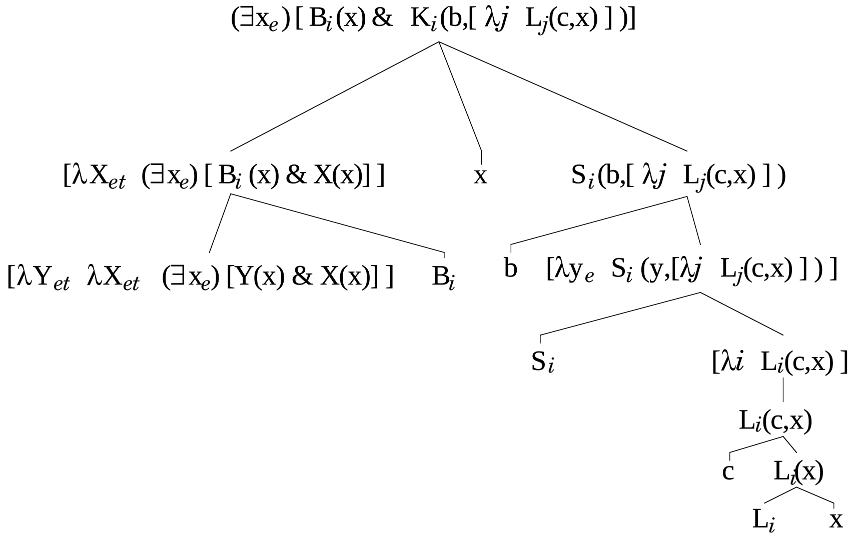
This fact is brought out by an alternative analysis of (55) to which we will now turn.

An exercise in part 4 strongly suggested that sentences with only one quantified NP do not show any scope ambiguities: the quantifying in rule (55) of part 4 does not produce any semantically distinct readings. However, this is only true as long as intensionality is not involved. For although our sentence (55) does not contain any quantified noun phrase other than ***a baby***, raising the latter does make a difference:

(60)



⇒


(∃x_e) [B_i (x) & K_i (b,[λj L_j(c,x)])]
[λX_et (∃x_e) [B_i (x) & X(x)]]
x
S_i (b,[λj L_j(c,x)])
[λY_et λX_et (∃x_e) [Y(x) & X(x)]]
B_i
b
[λy_e S_i (y,[λj L_j(c,x)])]
S_i
[λi L_i(c,x)]
L_i(c,x)
c
L_i(x)
L_i
x

According to this second reading, (55) is true of a situation s if there is a baby *x* (i.e. an individual that is a baby in s) such that, in s, Bill knows the proposition p_x consisting of all situations in which Coco loves *x*. So, contrary to what Bill says according to (56), p_x is a belief about a *specific* baby. It is this specificity that distinguishes the reading (60) from that in (56). (It should be noted that specificity does not imply uniqueness: due to the usual interpretation of the existential quantifier, there might be more than one baby x satisfying the relevant condition.) In order to grasp the difference between the two readings, one need only imagine situations s in which one is true and the other is not, i.e. situations accurately described by (56) but not by (60), or vice versa. For instance, if Bill (a native speaker of American English) utters (59) in s, where ***Zoë*** does refer to a baby, he thereby stands in the relation of saying to the proposition $p_{\text{Zoë}}$ that contains all situations s' in which Coco loves Zoë, the referent of the proper name ***Zoë***. In particular, then, there is some individual x (= Zoë) such that, in s, Bill stands in the saying relation to p_x. Hence, on the analysis (60), (55) is true. But is it also true under analysis (56)? For that to be the case, Bill would have to utter a sentence whose intension contains only situations in which Coco loves a baby, i.e. only such s' in which there exists an individual x who is a baby in s' and such that in s' Coco stands in the relation of love to x. Now, if we imagine that (59) is all that Bill says in s, (56) would only be true if the proposition $p_{\text{Zoë}}$ expressed by (59) contains only situations in which Coco loves a baby. But this is not the case: for (56) is also true of (possible) situations in which the only person that Coco loves is Zoë, who has meanwhile grown into a respectable young lady. Hence $p_{\text{Zoë}}$ is not a subset of p, i.e. the set of situations in which Coco loves some (unspecified) baby. It is therefore possible for (60) to be true without (56) being true. And, as an exercise will show, the converse also holds: the two readings ascribed to (55) are *logically independent*.

It is worthwhile to summarize the main aspects of the above analysis of sentences like (55), in which a quantified noun phrase occurs in the complement of a propositional attitude verb. The first claim is that such sentences are *systematically ambiguous* between a specific and a non-specific reading of the noun phrase in question. (Thus, if there is more than one NP_{quant} in the ***that***-clause, more ambiguities arise; likewise, we get more ambiguities by iterated embeddings of ***that***-clauses. We will, however, ignore such complications here.) This ambiguity claim is by no means trivial. For although it is undoubtedly true that (55) may be used to truthfully describe both Bill's utterances about specific babies loved by Coco and mere existence claims, this fact in itself does not imply that each use of (55) must be either specific or unspecific rather than, say, indeterminate. The idea that such sentences are indeed ambiguous between a so-called *de re* (= specific) and a *de dicto* reading has a long tradition but we will not try to justify or discredit it here.

The second ingredient of the above analysis is the reduction of the *de re/de dicto* ambiguity to a (quantifier) scope ambiguity: the specific reading is that in which the quantifier has wide scope over the attitude verb, whereas in the non-specific reading the quantifier is within the scope of the verb. Schematically, as along as one quantifier and one attitude is involved, the two readings have following form:

(61) (*dd*) $A_i(x,[\lambda i\ (Q_i y)\ P_i(y)])$

(*dr*) $(Q_j y)\ A_i(x,[\lambda i\ P_i(y)])$

(The reason why we have not renamed bound occurrences of the variable *i* will become clear in a second.) Even on this general level, two important differences between the two readings are apparent. The first is that the (61*dd*) reports that the relation expressed by the attitude verb (‘A_i’) holds between the subject (‘x’) and one proposition (‘$[\lambda i\ (Q_i y)\ P_i(y)]$’), whereas (61*dr*) says that the subject bears the relation to a certain quantity of propositions. Thus, in its *de dicto* sense, (62) reports one utterance of Bill’s, whereas in its *de re* sense it can be used to report as many utterances as there are babies:

(62) ***Bill says that Coco loves every baby.***

(*dd*) $S_i(b,[\lambda i\ (\forall y)\ [B_i(y) \rightarrow L_i(c,y)]\ ]\ )$

(*dr*) $(\forall y)\ [B_i(y) \rightarrow S_i(b,[\lambda i\ L_i(c,y)]\ )\ ]$

(It must be added that, according to our meaning rule (57'), even on the specific, *de re* reading, (62) may be used to report a single utterance; in that case the proposition expressed by that utterance would have to imply that Coco loves y, for every baby y.) The other difference between (61*dd*) and (61*dr*) concerns the status of the variable *i* as an argument to the quantifier. In (61*dd*) it is bound by the λ-operator introduced by the ***that***-clause and thus relativizes the extension of the quantified NP to whatever situations are relevant to the attitude expressed by ‘A’; but in (61*dr*) it is free and thus refers to whatever situation the translated sentence is about. The difference can best be seen from its effect in various examples. If we imagine that the unspecific reading (56) of (55) is used to (truthfully) describe a situation in which Bill uttered exactly one (English) sentence, then the intension of this sentence would have to be a subset of the proposition expressed by:

(63) ***Coco loves a baby.***

In particular, what Bill said would have to imply that there are babies: (63) can only be true of situations in which there exists at least one baby. But if (55) is used in its specific, *de re* sense (60) to report a particular utterance, then the sentence uttered by Bill may have been (59) which, unlike (63), does not imply the existence of any babies; in fact, Bill may have uttered (59) without even being aware of the fact that Zoë is a baby. On the other hand, the sentence (63) itself implies the existence of babies: the proposition said by Bill is about a certain individual y that is described as a baby. However, this is only true of the *de re* reading; the *de dicto* reading (55) may even be true in completely babyless worlds. Analogous remarks apply to the two readings of (62). In order for (62*dd*) to be true, Bill would not have to say anything about (or even know) any specific baby, whereas according to (62*dr*), the proposition(s) said by him would have to be about each baby y, whether or not Bill says (or even knows) that y is a baby.

One reason to carefully distinguish two aspects of the *de re*/*de dicto* ambiguity is that it has been argued that these are really two different phenomena that may but need not co-occur and should therefore be given an independent characterization. We cannot discuss this suggestion here but merely note that, according to the analysis presented above, the quantifier scope of the noun phrase and the evaluation of its extension (whether in the situation described or in the situations relevant to the attitude) cannot be separated: 'multiple reference [= wide scope] often necessitates transparency [= evaluation with respect to the situation described]', as Montague once put it. One strong point in favour of this correlation lies in an interesting application of seemingly unrelated examples involving certain transitive verbs. Before we show how this works, we have to keep our promise and briefly turn to a further argument against the identification of propositions and sets of worlds or situations.

It is a characteristic feature of *de re* readings that they report a subject to bear a certain attitude to one or more propositions *about* the object quantified over. Thus, on the *de re* reading of (64), Tom is said to believe something about some creature that is characterized as a monster:

(64) ***Tom believes that a monster is hiding under the bed.***

The sentence is thus understood in a *relational* sense that may also be expressed by:

(65) ***Tom believes of a monster that it is hiding under the bed.***

(Note that (65) lacks a *de dicto* reading.) It appears that (65) establishes the relation of belief to hold among three objects the first two of which are individuals (viz. Tom and Nessie, say), whereas the third is the property of hiding under the bed:

(65') $(\exists y)\ [M_i(y)\ \&\ B^*_i(t,y,[\lambda j\ \lambda z\ U_j(z)])\ ]$,

where 'M' ($\in$ $\mathrm{Con}_{s(et)}$) translates ***monster***, 'B*' ($\in$ $\mathrm{Con}_{s((st)(e(et)))}$) translates relational ***believe***, and 'U' ($\in$ $\mathrm{Con}_{s(et)}$) symbolizes the property of hiding under the bed. With 'B' ($\mathrm{Con}_{s((st)(et))}$) as the translation of ***believe***, the *de re* reading ascribed to (64) is:

(64') $(\exists y)\ [M_i(y)\ \&\ B_i(t,[\lambda j\ U_j(y)])\ ]$

What does it mean for someone to believe something *of* a monster (or of anything)? Whatever the exact conditions are, belief about something should at least imply *acquaintance* with that thing. We may thus assume the following general principle:

(66) $(\forall i)\ (\forall x)\ (\forall y)\ (\forall P_{s(et)})\ [B^*_i(x,y,P) \to A_i(x,y)]$,

where 'A' symbolizes the relation of being acquainted. Now the problem is that, if we understand the *de re* readings (like (64')) in a relational sense (as in (65')), they would imply that their subjects are acquainted with everybody. For if $u \in \mathrm{Var}_e$, (64') is obviously equivalent to:

(67) $(\exists y)\ [M_i(y)\ \&\ B_i(t,[\lambda j\ [U_j(y)\ \&\ (u = u)]\ ])\ ]$,

which in turn expands to:

(67') $(\exists y)\ [M_i(y)\ \&\ B_i(t,[\lambda j\ [\lambda k\ \lambda v\ [U_k(y)\ \&\ (u = v)]\ ]\ (j)\ (u)\ ])\ ]$,

by two backwards applications of λ-conversion. The relational reformulation of this is:

(67*) $(\exists y)\ [M_i(y)\ \&\ B^*_i(t,u,[\lambda k\ \lambda v\ [U_k(y)\ \&\ (u = v)]\ ])\ ]$,

to which we may apply the principle (66) yielding:

(68) $A_i(t,u)$

Since u was arbitrary, we have it that (64') implies:

(69) $(\forall u)\ A_i(t,u)$,

a highly undesirable result. One may wonder who or what is responsible for it. Again, we will not enter this discussion here, but only mention that it is hard to argue against either the relational reduction of *de re* belief or the acquaintance principle (66). But if both are right, the absurd consequence must have to do with the general framework, and particularly with the fact that, according to the possible worlds framework, any proposition *p* is about any individual x whatsoever, in the sense that there is a property *P* such that *p* is the set of situations in which x has this property: *P* can be defined as the function taking any situation s to $\{u \in D_e \mid u = x \text{ and } s \in p\}$. This fact was used in the initial step of the above argument (to get from (64) to (67')).

We have sketched three arguments against the identification of propositions (i.,e. intensions of sentences) and sets of situations or worlds, or indices), and they are certainly not the only ones. Moreover they are closely related to each other and can be seen as consequences of the one big problem of possible worlds semantics, viz. the fact that, being sets of *urelements*, propositions are essentially unstructured entities; in particular they do not reflect enough of the structure of the sentences or ***that***-clauses that can be used to express them. This lack of *fine-grainedness* is at the heart of all three arguments and a solution to any them has to face it. However, despite of some promising attempts within and outside possible worlds semantics, the problem of fine-grainedness still waits for a convincing and general solution.

5.5 Referential opacity in transitive verbs

It was already mentioned that ***that***-clause embeddings, though certainly not the only intensional constructions, are very central in that many other intensional phenomena can be reduced to them. In the present section, we will study one such phenomenon-plus-reduction in some detail. Here is the phenomenon:

(70) ***Coco hugs an abandoned baby.***

(71) ***Coco seeks an abandoned baby.***

There is an important semantic difference between (70) and (71): whereas (70) implies the existence of at least one abandoned baby – how else could Coco hug her? – (71) may even be true without there being any babies at all: Coco just has some ideal in her mind and hopes that reality fits it. Now the reason why (70) does have an existential entailment is brought out by its ordinary first-order analysis (70') which we can construe compositionally, in the usual manner:

(70') $(\exists x)\ [A_i(x)\ \&\ B_i(x)\ \&\ H_i(c,x)]$

(For simplicity we assume that ***abandoned*** is an intersecting adjective.) Applying the same kind of formalization technique to (71) yields (71'), which has the same entailment:

(71') $(\exists x)\ [A_i(x)\ \&\ B_i(x)\ \&\ S_i(c,x)]$

Now, it seems that (71) does have a reading like (71'), i.e. one according to which Coco is looking for some specific individual that happens to be a baby; we will return to this matter in due course. For the time being, however, we are concerned with the far more serious problem that a very prominent reading of (71) does not imply the existence of any babies whatsoever and that, consequently, (71') cannot be the only correct translation of (71).

Before attacking the interpretation of (71), we will first make sure that we are actually dealing with an intensional context. Let us therefore imagine that the only babies that have been abandoned happen to have no teeth and that all toothless babies have been abandoned. Given our interpretation of the indefinite article, we conclude that (72) and (72') are coextensional and can therefore be replaced for each other in any extensional context:

(72) ***an abandoned baby***
(72') ***a toothless baby***

(***a*** and ***an*** are, of course, superficial variants of the same word.) Performing the substitution (70) and (71) then shows that, unlike the one of ***hug***, the object position of ***seek*** is not extensional:

(73) ***Coco hugs a toothless baby.***
(73') ***Coco seeks a toothless baby.***

For given our assumptions, the truth of (70) guarantees that there will be some toothless baby that Coco hugs. But her search may still be directed towards those unfortunate babies that have been abandoned without being at the same time a hunt for babies without teeth: Coco may simply be ignorant about the coincidence of the two features. (However note that, if (71) is understood as a report about Coco's search of a particular baby, substitution does work.)

In order to motivate an alternative treatment of (71) that avoids the unwelcome existential implication, the following rough paraphrase proves to be helpful:

(74) ***Coco tries to find an abandoned baby.***

There are various observations to be made about (74) and its relation to (71). The first is that, however incomplete a paraphrase it may be, it is probably close enough to give an idea why one should not expect (71) to behave exactly like (70). That is, even though (74) might, for instance, indicate a more active search than (71) does, the reason why (74) does not imply the existence of babies is likely to be the same as for (71). Secondly, (74) seems to be prone to the same kind of ambiguity as was indicated in connection with (71): both sentences may be understood as describing Coco's search for a particular individual that happens to be an abandoned baby – in which case both do of course imply the existence of such babies. Finally, (74) comes pretty close to a reduction of (71) to a propositional attitude report: although ***try*** does not take a ***that***-clause, it is understood that the subject of its infinitival complement is 'controlled' by the matrix subject. We could thus decompose ***try*** into a logical variant of a propositional attitude $\mathrm{T} \in \mathrm{Con}_{s((st)(et))}$:

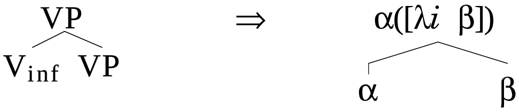
(75) ***try***' $:= [\lambda P_{s(et)}\ \lambda x_e\ \mathrm{T}_i(\mathrm{x}, [\lambda i\ \mathrm{P}_i(\mathrm{x})\ ]\ )\ ]$

An approximate paraphrase for the propositional attitude 'T_i' may be ***try to bring about***. According to (75), a sentence like (76) would get translated as (76'), if we assume the straightforward translation rule (77):

(76) ***Bill tries to sleep.***

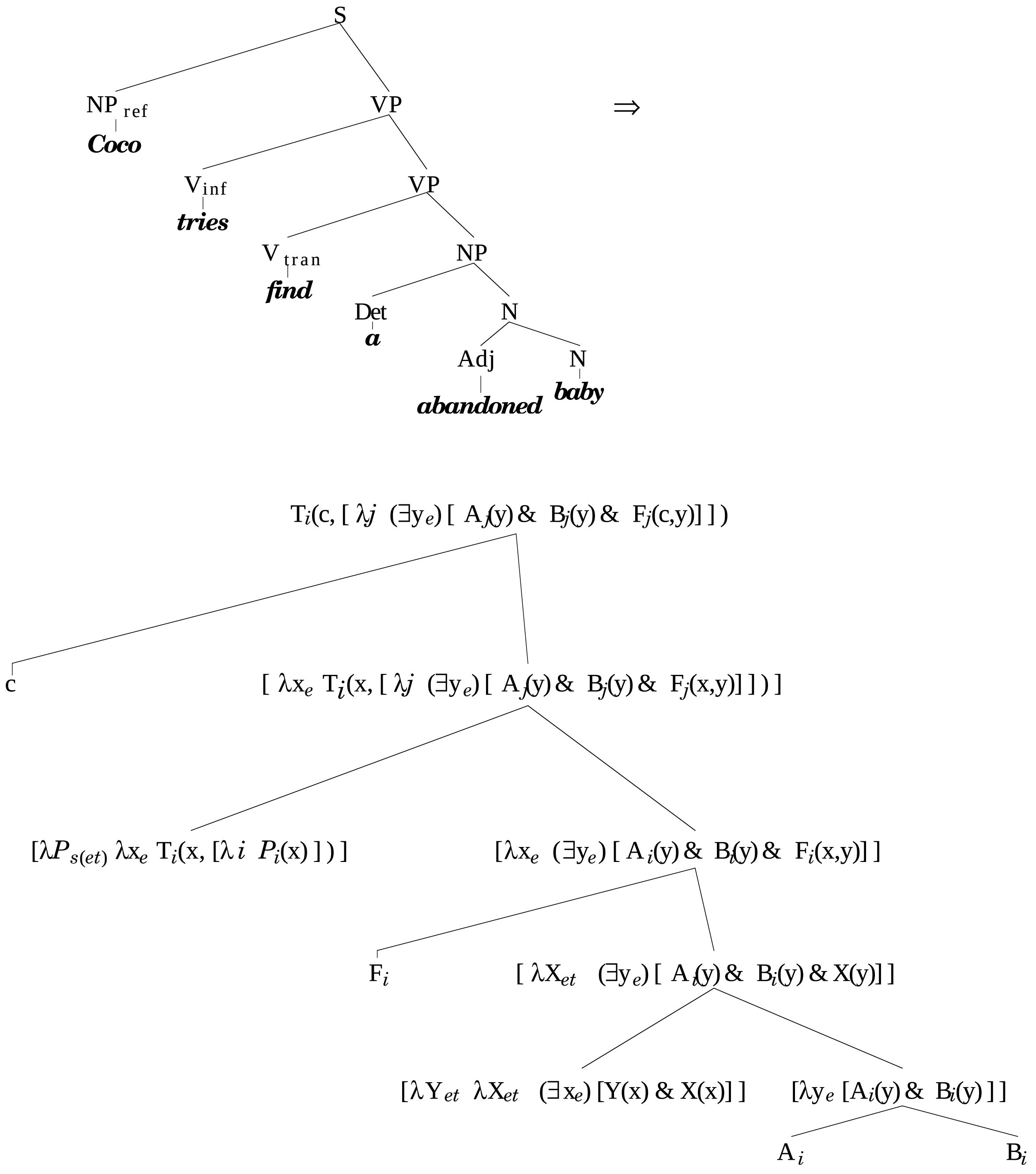
(76') $\mathrm{T}_i(\mathrm{b}, [\lambda j\ \mathrm{S}_j(\mathrm{b})\ ]\ )$

(77)



(We assume ***try*** to be of the category V_{inf} of infinitive embedding verbs; note that (77) is intensional because the variable i gets bound.) And this is what we get for (74):

(78)


S
NP_{ref}
Coco
VP
⇒
V_{inf}
tries
VP
V_{tran}
find
NP
Det
a
N
Adj
abandoned
N
baby
$T_i(c, [\lambda j\ (\exists y_e)[A_j(y) \& B_j(y) \& F_j(c,y)]])$
c
$[\lambda x_e\ T_i(x, [\lambda j\ (\exists y_e)[A_j(y) \& B_j(y) \& F_j(x,y)]])]$
$[\lambda P_{s(et)}\ \lambda x_e\ T_i(x, [\lambda i\ P_i(x)])]$
$[\lambda x_e\ (\exists y_e)[A_i(y) \& B_i(y) \& F_i(x,y)]]$
F_i
$[\lambda X_{et}\ (\exists y_e)[A_i(y) \& B_i(y) \& X(y)]]$
$[\lambda Y_{et}\ \lambda X_{et}\ (\exists x_e)[Y(x) \& X(x)]]$
$[\lambda y_e\ [A_i(y) \& B_i(y)]]$
A_i
B_i

The constants and their types are as usual; in particular, the transitive verb ***find*** translates as 'F_i', where $F \in Con_{e(et)}$. The analysis (78) makes it perfectly clear that we cannot replace ***an abandoned baby*** by ***a toothless baby*** without thereby possibly changing the truth-value of (74), even though the two noun phrases may extensionally coincide. For we would have:

(79) ***Coco tries to find a toothless baby.***

(79') $T_i(c, [\lambda j\ (\exists y_e)\ [T_j(y)\ \&\ B_j(y)\ \&\ F_j(x,y)]\ ]\ )$

And (79') does not necessarily have the same truth-value as (78) (i.e. the top formula in the translation tree). Even if, in the situation described, a baby is toothless iff she has been abandoned, there are certainly many possible situations in which the two sets do not coincide; consequently, the 'λi'-terms in (78) and (79') denote different sets of situations and Coco may try to bring about one but not the other. So the reason for the non-equivalence of (74) and (79) has to do with the fact that the substitution affects a position within a ***that***-clause, so that extensional equivalence does not suffice. Thus, if (71) and (73') are (rough) paraphrases of (74) and (79), respectively, we may say that their non-equivalence is due to a hidden ***that***-clause in which the objects of ***seek*** appear. This would also explain that these sentences seem to have additional readings, according to which they would be equivalent (given the coextensionality of the objects): as one can easily verify, the *de re* readings of their paraphrases (74) and (79) are:

(74) (*dr*) $(\exists x)\ [A_i(x)\ \&\ B_i(x)\ \&\ T_i(\ c, [\lambda j\ F_i(c,x)\ ]\ )$

(79) (*dr*) $(\exists x)\ [T_i(x)\ \&\ B_i(x)\ \&\ T_i(\ c, [\lambda j\ F_i(c,x)\ ]\ )$

It is obvious from these two formulae that the *de re* readings of (74) and (79) must have the same truth-values if only the two relative clauses are coextensional. We may thus speculate that the use of (71) and (73') as descriptions of acts of looking for specific babies is also due to quantifying in. Such an explanation would, of course, not be possible if we analyzed ***seek*** as a binary relation among individuals: as we have seen before, raising a quantified object of such a relation has no semantic effect if the subject is referential.

We have seen that the simple idea of paraphrasing ***seek*** as ***try to find*** has quite far-reaching consequences: it may be used to explain both the unusual logical behaviour of (certain readings of) sentences containing this verb and a characteristic *de re*/*de dicto* ambiguity in these sentences. The idea of employing the ***seek***-as-***try-to-find*** paraphrase in this way is due to Quine. We will now see how we can incorporate it into a type-theoretic framework.

It may be tempting to suggest the following simplistic lexical decomposition as a way to capture Quine's idea:

(80) ***seek***' = $[\ \lambda y_e\ \lambda x_e\ T_i(\ x, [\lambda i\ F_i(x,y)\ ]\ )\ ]$

However, this decomposition simply does not do the job we want it to do: it still gives us a binary relation among individuals. More specifically, an exercise will show that (80) would only allow us to analyze the *de re* readings of sentences containing ***seek***. But Quine's paraphrase was supposed to give us the *de dicto* readings, at least as long as no quantifying in is involved. Schematically, Quine's idea is to analyze sentences of the form (81) as (82):

(81) X ***seeks*** *N*.
(82) X ***tries to find*** *N*.

The important point is that, in the interesting cases anyway, *N* is a quantified noun phrase translating into an expression Q of category (*et*)*t*. (80) misses precisely this point; for it only provides a schema for referential objects, and we will see that the latter are of little interest to this analysis. Under the (simplifying but harmless) assumption that the subject X is referential, we get the following (*de dicto*) translation of the schema (82):

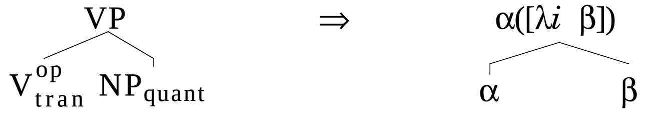
(82') $T_i(\ x, [\lambda j\ (Q_j y)\ F_j(x,y)\ ]\ )$,

where 'x' translates 'X' and 'Q' translates '*N*'. Now we are almost there. For if (82') is a correct translation of (82) and hence of (81), we may indeed decompose ***seek*** in precisely the way indicated in that translation:

(83) ***seek***' = $[\ \lambda Q_{s((et)t)}\ \lambda x_e\ T_i(\ x, [\lambda i\ (Q_i y)\ F_i(x,y)\ ]\ )\ ]$

(83) is Montague's version of Quine's analysis of ***seek***. Its most puzzling feature is the complicated type ascribed to ***seek***: (*s*((*et*)*t*))(*et*)! Clearly, there is no problem to combine this translation with a quantified object to obtain the desired paraphrase:

(84)



V^{op}_{tran} is the category of *(referentially) opaque* transitive verbs, i.e. those verbs that show the same kind of odd behaviour as ***seek***. Are there any other verbs in V^{op}_{tran}? Here is at least one fairly certain case, already discussed by the medieval logician Buridan:

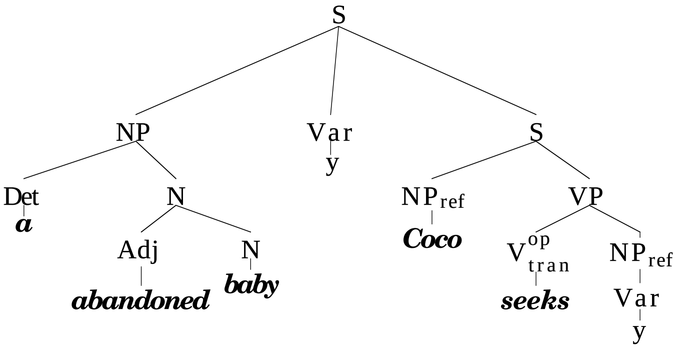
(85) ***Ernie owes Bert a horse-sized cookie.***

Under the convenient assumption that ***owe*** combines with its indirect object to form a complex transitive verb, this complex verb behaves pretty much like ***seek***. From the truth of (85) we cannot, e.g., deduce the existence of horse-sized cookies: maybe Ernie was so certain that he would win that he did not care what he bet. And, as in the case of ***seek***, there is a paraphrase relating the complex verb to a propositional attitude:

(85') ***Ernie is obliged to give Bert a horse-sized cookie.***

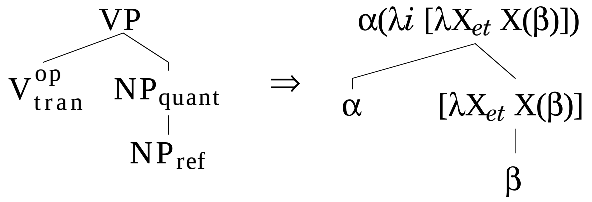
We leave the details of the analogy between ***seek*** and ***owe*** to an exercise and meanwhile turn to the *de re* readings of sentences with ***seek***. As was already mentioned, we could obtain these *de re* readings by treating ***seek*** as an *e*(*et*)-relation, as in the decomposition (80). However, this would mean that we would have to analyze ***seek*** as being ambiguous between an opaque and a *transparent* reading. As our analysis of the opaque reading rests on a reduction to a ***that***-clause embedding, it would certainly be preferable to obtain the transparent reading in the same way as the *de re* readings of sentences containing overt ***that***-clauses: we would then have a simple explanation for the ambiguity. So let us see whether we can analyze (71) by raising the object:

(86)



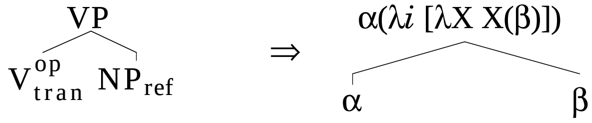
In order to translate this tree, we need a rule for the combination of an opaque transitive verb with a referential object. One way of attacking this problem is to recategorize the referential noun phrase as a quantifier, i.e. to apply Montague Lifting to it:

(87)



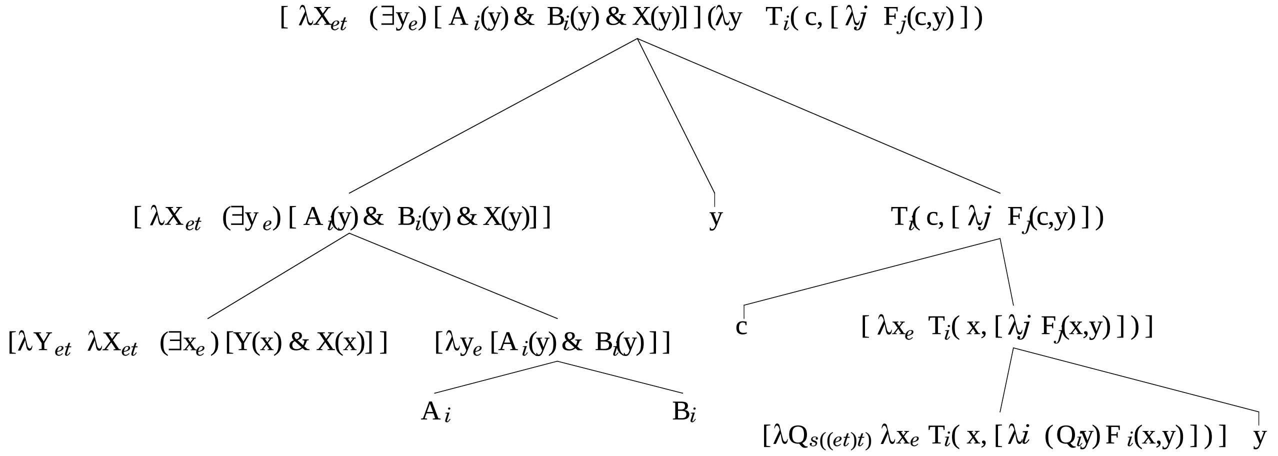
Bearing (87) in mind, we may then give a more direct or 'one-step' treatment of the combination ' V^{op}_{tran} + NP$_{ref}$', i.e. one that leads to the same result without Montague Lifting:

(87')



With (87'), (86) translates as:

(88)


$[\lambda X_{et}\ (\exists y_e)[A_i(y)\ \&\ B_i(y)\ \&\ X(y)]](\lambda y\ T_i(c, [\lambda j\ F_j(c,y)])$
$[\lambda X_{et}\ (\exists y_e)[A_i(y)\ \&\ B_i(y)\ \&\ X(y)]]$
y
$T_i(c, [\lambda j\ F_j(c,y)])$
$[\lambda Y_{et}\ \lambda X_{et}\ (\exists x_e)[Y(x)\ \&\ X(x)]]$
$[\lambda y_e\ [A_i(y)\ \&\ B_i(y)]]$
c
$[\lambda x_e\ T_i(x, [\lambda j\ F_j(x,y)])]$
A_i
B_i
$[\lambda Q_{s((et)t)}\ \lambda x_e\ T_i(x, [\lambda i\ (Q_i y)\ F_i(x,y)])]$
y

Two λ-conversions now turn the resulting formula into:

(88) $(\exists y)[A_i(y)\ \&\ B_i(y)\ \&\ T_i(c, [\lambda j\ F_j(c,y)])$,

which is an alphabetic variant of the *de re* reading (74*dr*) of the ***try-to-find*** paraphrase. So Quine's explanation of the ambiguity of (71) in terms of an ambiguity in the paraphrase carries over to Montague's decomposition of ***seek***.

Some features of this analysis of referentially opaque verbs should be pointed out to prevent serious misunderstandings that can unfortunately be found in a large part of the literature. The first point is that the main idea does not depend on the fact that ***seek*** is a transitive verb whose (direct) *object* position behaves in an unfamiliar way. Verbs with odd subject (or indirect object) behaviour could be analyzed along the same lines. Indeed, as we have seen, the subject position of a raising verb like ***appear*** is intensional and it is not hard to see that the relation between (10) and (12) is very similar to that between a ***seek***-sentence and its ***try-to-find*** paraphrase:

(10) ***A unicorn appears to be approaching.***
(12) ***It appears that a unicorn is approaching.***

We will not go into the details of such an analysis of raising but it should be clear that the the above treatment of ***seek*** can in principle be adapted to these cases, as Montague himself had already observed. (Whether it should be adapted is, of course, a different matter.) Let us instead turn to another important feature of the above analysis of referential opacity.

It should first be pointed out that the complex type assignment is essential to Montague's formulation of Quine's analysis of referential opacity: ***seek*** gets assigned the type $(s((et)t))(et)$ of relations between individuals and $\mathrm{NP}_{\mathrm{quant}}$-extensions, because it is, in effect, analyzed as a complex but defective verb phrase consisting of an attitude verb (***try***) and a ***that***-clause with one $\mathrm{NP}_{\mathrm{quant}}$ missing ('***that*** SUBJ ***finds*** ____'); the gap is then filled by the object of ***seek***. But only if we know the intension of the whole quantified NP, the proposition of the defective ***that***-clause can be determined; the complexity of the type assigned to ***seek*** is thus due to the fact that the VP's extension depends on how the object of ***find*** is quantified over; moreover, we need the intension, rather than the extension, of the object, because the quantification takes place within a ***that***-clause and thus yields different results for different situations.

There are two reasons why we have emphasized the importance of the complexity of the type assignment. The first, somewhat banal one is that it has often been misconstrued in the literature. More importantly, though, to understand the reason for this complexity is to understand the full generality of the analysis. For although we have as yet only applied it to indefinite noun phrases of the form $\ulcorner$***a*** + $N\urcorner$, there is no reason why we shouldn't try it on, say:

(89) ***Coco seeks every abandoned baby.***
(90) ***Coco seeks no abandoned baby.***

Our analysis yields the following readings:

(89) (*dr*) $(\forall y)\,[\,[A_i(y)\ \&\ B_i(y)] \rightarrow T_i(c,[\lambda j\ F_j(c,y)\,]\,)\,]$

(89) (*dd*) $T_i(c,[\lambda j\ (\forall y)\,[\,[A_i(y)\ \&\ B_i(y)] \rightarrow F_j(c,y)\,]\,]\,)$

(90) (*dr*) $\neg(\exists y)\,[\,A_i(y)\ \&\ B_i(y)\ \&\ T_i(c,[\lambda j\ F_j(c,y)\,]\,)\,]$

(90) (*dd*) $T_i(c,[\lambda j\ \neg(\exists y)\,[\,A_i(y)\ \&\ B_i(y)\ \&\ F_j(c,y)\,]\,]\,)$

Now, whereas (89*dr*) seems to be quite a straightforward reading, (89*dd*) is apparently much harder to get, if one gets it at all: there may be room for some dialect variation here; in any case, the *de re* reading is certainly the first that comes to mind. This fact is the more remarkable if we compare the situation not only with the indefinite case (where the *de dicto* reading might be said to be preferred due for plausibility reasons) but also to the paraphrase (89'), where the *de dicto* reading is much easier to obtain:

(89') ***Coco tries to find every abandoned baby.***

It seems that (89') may indeed express that Coco attempts to achieve the goal of finding *all* babies, which is precisely what the *de dicto* analysis predicts. It is not clear how to explain these data from the perspective of the above analysis.

Let us now turn to (90). If the sentence is acceptable at all, the *de re* reading seems to be o.k. again: (90) may indeed be used to convey that there is no abandoned baby that Coco tries to find. How about the *de dicto* reading, then? Can (90) have the reading (90*dd*)? Obvioulsy not. For this formula expresses that Coco tries to bring about a situation in which she does not find any abandoned baby; in other words, according to (90*dd*) Coco *avoids* finding abandoned babies. Such a reading is clearly impossible. On the other hand, (90*dr*) is not the only reading of (90). We also have:

(90!) $\neg T_i(c, [\lambda j (\exists y) [B_j(y) \& E_j(y) \& F_j(c,y)]])$

Formally, the only difference between (90!) and (90*dd*) lies in the scope of the negation. Still, the two formulae describe radically different situations. For (90!) is the negation of the *de dicto* reading of:

(71) ***Coco seeks an abandoned baby.***

It thus appears that the scope of the negation inherent in the word ***no*** is independent of the scope of the rest of the NP$_{\text{quant}}$ it determines. Clearly, this journey of the negation is not accounted for by the above analysis.

In order to avoid the impression that Quine's paraphrase only works for indefinite NPs of the form $\ulcorner \boldsymbol{a} + N \urcorner$, let us at least apply it to one case where it does predict the correct results:

(91) ***Coco seeks Bill***.

First note that this sentence is not ambiguous, although it contains ***seek***. And the above account would only predict one reading: quantifying in is not applicable, because the object is a referential NP. On the other hand, (91) certainly reports a search of one particular individual and has thus a *de re* flavour. But there is nothing to worry about. For the predicted *de dicto* reading is:

(91) $[\lambda Q_{s((et)t)} \lambda x_e T_i(x, [\lambda i (Q_i y) F_i(x,y)])] ([\lambda i \lambda X X(b)]) (c)$,

which λ-reduces to:

(91') $T_i($ c, $[\lambda i\ F_i(c,b)])]$,

which accurately describes the search for one individual.

We have remarked earlier that Quine's paraphrase is not meant to be perfect: there may be subtle differences between the meanings of ***seek*** and ***try to find***. Still, such differences are not vital to the above account of referential opacity. For one thing, one may argue that the constants 'T' and 'F' occurring in the translation (83) of ***seek*** need not be the same constants used in the translations of ***try*** and ***find***, respectively: we may instead regard them as denoting building blocks of lexicalized concepts that are not necessarily themselves lexicalized. (Indeed, such building blocks may be regarded as corresponding to primitive mental concepts; but they may as well be thought of as abstract theoretical entities introduced for economic reasons.) If this is so , then the same concepts may play a role in the decomposition of ***try*** and ***find*** without actually denoting the intensions of these words. In that case ***seek*** could not be paraphrased as ***try to find***; but the following implication may hold:

(92) $(\forall i)\ (\forall x)\ (\forall Q)\ [\boldsymbol{try}'_i(x, [\lambda j\ \lambda z\ (Q_j y)\ \boldsymbol{find}'(z,y)]) \rightarrow \boldsymbol{seek}'_i(x, Q)]$,

validating the schematic inference from:

(92') x ***tries to find*** *N*

to:

(92") x ***seeks*** *N*

where *N* is an arbitrary quantified NP and no quantifying in is involved. Now, whether (92) is correct or not and whether it should or should not be accepted as a meaning postulate (or a consequence thereof), it is perfectly clear that its acceptance is quite independent of the decomposition (83) of ***seek*** into 'T' and 'F'. This means that we could keep the spirit of Quine's analysis of referential opacity without committing ourselves to the validity of the '***seek*** = ***try to find***' paraphrase: the odd behaviour of ***seek*** can be explained on the basis of its analyzability into two concepts *similar, but not identical to* those expressed by ***try*** and ***find***.

For a large part of its inferential behaviour, the decomposition of ***seek*** is even completely irrelevant. The fact that raising its object makes a difference with respect to, say, existential implications, is already brought out by its complex type (plus the relevant semantic combinations). To see this, imagine we translated ***seek*** into the expression 'S_i', where S is a constant of type $s(s((et)t))(et)$. We could then still capture the decomposition (83) by means of a postulate like (92), but with a material equivalence ('↔') replacing the implication '→'. (This is essentially the way Montague proceeded.) Obviously, the semantic effect would be the same. In particular, all unwelcome inferences would still be blocked. But failure of a certain inference means that there is a model M in which the premise is true but the purported conclusion is not. Clearly, this model satisfies our postulate. But had we decided to dispense with the postulate or replace it by the weaker (92), the inference would still be blocked, because there would still exist a model that makes the premise true but the conclusion false, viz. M. Following these lines, one may even argue (as Montague did), that the treatment of opaque verbs as unanalyzed, abstract relations possesses the advantage of greater generality over Quine's original reduction by paraphrase: it will even work on verbs for which there is no suitable paraphrase at hand. However, apart from the quite remarkable fact that all known cases of referential opacity can (at least approximately) be paraphrased in a Quinean manner, the abstract analysis has the clear disadvantage of letting the denotation of ***seek*** appear rather obscure.

Let us conclude this discussion with a brief remark on referential *transparency*, i.e. the opposite of opacity. We have marked the distinction between opaque and transparent verbs by means of different types. Strictly speaking, this is not necessary. For just as in the case of noun phrases, a strategy of generalizing to the worst case may be applied to find a unique type for all transitive verbs. In fact the following decomposition of the transparent verb ***find*** into a constant of type $s(e(et))$ reveals the idea behind this strategy:

(93) $[\ \lambda Q_{s((et)t)}\ \lambda x_e\ (Q_i y)\ F_i(x,y)\ ]$

It is easy to check that this type shift has the same effect as the original type distinction: applying (93) to a quantified NP results in the NP's quantifying over the object position, just as the usual combination XY of binary relations (among individuals) and quantifiers would have it. And if we wish to take *subject-opacity* (as in ***appear***) into account, we even get more complicated types and decompositions, as an exercise will show.

5.6 Models and possible worlds

In our discussion of intensional phenomena we have frequently referred to and made use of obvious properties of situations without making sure that these properties are entailed by or only consistent with our model-theoretic framework. Thus in our discussion of:

(55) ***Bill says that Coco loves a baby.***

we showed that the *de re* reading does not imply the *de dicto* reading by pointing out that the former is true in certain situations s (in which Bill utters one sentence about some particular baby), while the latter would be false in the same situation. Now, this argument would certainly be sound if we were dealing with real situations. The problem is, we aren't. For one thing the situations may not be real at all, but merely possible. Moreover, strictly speaking, we are not dealing with situations at all but with the basic ingredients of arbitrary ontologies on which the arbitrary models of two-sorted type theory are based. So what guarantees that each such model contains a situation in which, say, Bill utters a certain sentence? In fact what does it mean for an abstract set-theoretic object like a Ty2-model to contain a situation in which Bill says something? The latter question is relatively easy to answer: for our purposes it suffices that the individual serving as the interpretation of (the translation of) the proper name ***Bill*** be in the extension of, say, ***says that Coco loves Zoë***. Now the answer to the first question is also immediate: *nothing* guarantees that an arbitrarily chosen model will contain any such situation. Now, as long as we are only interested in defeating conceivable entailments, it suffices to show that at least one model contains the kinds of situation envisaged. But the example already shows that, in our informal discussion, we have usually assumed the Ty2-models to be richer than they need be according to the definitions. There is nothing wrong with such *naturalness* assumptions: if we find that one of our tacit assumptions about situations is not met by all Ty2-models, we can simply restrict our class of intended models (as long as the assumptions are at all compatible with our notion of a model). But one should be aware of the fact that the notion of a model as such is rather poor, to poor indeed for most semantic purposes.

Restricting the class of intended models to arrive at a more feasible notion of a situation can be compared with the introduction of meaning postulates for modelling sense relations in the lexicon. Thus, e.g., if we want the inference from:

(94) ***Benjamin is an elephant.***

to:

(95) ***Benjamin is an animal.***

to come out as valid, we may postulate the extension of (the translation of) ***elephant*** to always (i.e. in every situation of every model) be a subset of the extension of (the translation of) ***animal***. Similarly, for the sake of the above-mentioned argument involving (55), we may postulate that any intended model M should satisfy:

(96) $(\exists i)\ (\forall q_{(st)})\ [\ \mathrm{S}_i(\mathrm{x}, q) \leftrightarrow (\forall j)\ [\,p(j) \rightarrow q(j)\,]\ ]$,

where ‘b’ translates Bill, ‘S_i’ translates ***say***, and p denotes the intension of ***Coco loves Zoë*** (in M). (96) says that there is at least one situation in which every proposition that Bill says is a superset of the set of situations in which Coco loves Zoë. We could then use (96) in a more rigorous version of the argument sketched above. Indeed, since the informal argument does not depend on the particular choice of the subject ***Bill*** nor the sentence he is supposed to have uttered, we may think of (96) as a special case of the more general principle:

(97) $(\forall \mathrm{x}_e)\ (\forall p_{st})\ (\exists i)\ (\forall q_{st})\ [\ \mathrm{S}_i(\mathrm{x}, q) \leftrightarrow (\forall j)\ [\,p(j) \rightarrow q(j)\,]\ ]$

However, this principle is too general to be sound. For it implies an instance of the contradictory scheme (U) discussed in section 5.4. To see this, we first define the attitude of ‘saying no more than’:

(98) $\mathrm{S}^* := [\lambda i\ \lambda p_{st}\ \lambda \mathrm{x}_e\ (\forall q_{st})\ [\ \mathrm{S}_i(\mathrm{x}, q) \leftrightarrow (\forall j)\ [\,p(j) \rightarrow q(j)\,]\ ]$

and check that the following Ty2-formulae are logically equivalent:

(99) $\mathrm{S}^*_i(\mathrm{x},p)$
(100) $(\forall q)\ [\ \mathrm{S}^*_i(\mathrm{x},q) \leftrightarrow (p = q)\ ]$

(100) implies (99) because the equation to the right of ‘$\leftrightarrow$’ becomes equivalent when we insert p for q; the other direction of the implication can be obtained by observing that (99) and the truth of ‘$\mathrm{S}^*_i(\mathrm{x},q)$’ imply that (the propositions denoted by) p and q have the same supersets and are thus identical. We leave the details to the reader. Since (97) practically contains an occurrence of (99), we can now apply this equivalence to

reformulate (97) as an instance of that evil scheme (U):

(101) $(\forall x_e)\ (\forall p_{st})\ (\exists i)\ (\forall q)\ [\ S^*_i(x,q) \leftrightarrow (p = q)\ ]$

One thing this little argument shows is that the notion of an intended model may be more problematic and harder to come by than one might at first imagine. At any rate, the notion of a possible situation is certainly more complex than the simple conceptual relations traditionally captured by meaning postulates.

We will soon see that the above argument may also be used to shed some light on the important distinction between a model on the one hand and an index, a situation, or a possible world on the other. To begin with, the two notions certainly have a lot in common. For just as the extension of an expression depends on the particular model, so does it depend on the particular situation within the model. In fact, it is tempting to think of the possible situations to which the type *s* variables refer as models determining the extensions of various expressions. So why do we not take the elements of the ontological layers D_s to be models and interpret formulae '$\alpha(i)$' by *evaluating* α in the model denoted by i: $[\![\alpha]\!]^{g(i)}$? Of course, we should not proceed quite like that. For if α itself contains i as a free variable, then either (*a*) the model g(i) would have to assign g(i) to i, or else (*b*) different (free) occurrences of 'i' would refer to different models. Now whereas (*b*) would obscure logical form, (*a*) is clearly incompatible with the set-theoretic notion of a function. However, there is a way out: if we let the variables of type *s* refer to Ty1-models, we may restrict their occurrences to expressions that can be interpreted within the latter. More specifically, let a *simple* Ty2-formula be one in which the $i \in \mathrm{Var}_s$ only occur in λ-prefixes or in sub-formulae of the form '$\alpha(i)$', where α is either a lexical expression (constant or variable) of some type *sa* and *a* does not contain *s*, or else α has the form '$[\lambda j\ \beta]$', where $j \in \mathrm{Var}_s$. In the first case, the α could be interpreted as a Ty1-expression *of type a* to be evaluated in the Ty1-model denoted by i. Thus the following two Ty2-formulae are simple:

(102) $[\lambda i\ \lambda j\ (i = j)\ ]$

(103) $(\forall i)\ [P_{s(et)}(i)\ (x) \rightarrow \neg[\lambda i\ \lambda y_e\ (y = x_e)\ ](j)\ (x)]$

Here is are two Ty2-formulae that are not simple:

(104) $R_{s(st)}(i)$

(105) $S_{e(st)}(\mathrm{x}_e)(i)$

But maybe we do not need such formulae in indirect interpretation; indeed, a brief check of the Ty2-formulae discussed so far reveals that they all meet the simplicity restriction. Now, one can show that simple formulae can be interpreted in the way indicated, by letting the type s variables range over Ty1-models. (We omit the details.) It thus seems that replacing worlds by models would amount to essentially the same treatment of intensionality as the one given above.

Or wouldn't it? Whereas syntactically the case is clear, the interpretation via models differs from that via indices in various subtle but important respects. For instance, no two distinct models can agree on the extensions of all constants and variables: if the Ty1-models $M = ((D_a)_{a\in \mathrm{T}}, \mathrm{F}, \mathrm{g})$ and $M' = ((D'_a)_{a\in \mathrm{T}}, \mathrm{F}', \mathrm{g}')$ satisfy $\mathrm{F}(\mathrm{c}) = \mathrm{F}'(\mathrm{c})$ and $\mathrm{g}(\mathrm{x}) = \mathrm{g}'(\mathrm{x})$ for arbitrary c and x, we can immediately conclude that $\mathrm{F} = \mathrm{F}'$ and $\mathrm{g} = \mathrm{g}'$, by the extensionality of functions; and the two ontologies must coincide because $D_e = \mathrm{dom}(\mathrm{g}(\mathrm{x}_{et})) = \mathrm{dom}(\mathrm{g}'(\mathrm{x}_{et})) = D'_e$ etc. On the other hand, two worlds in which all lexical expressions happen to have the same extensions may still be distinct – even if they coexist within the same model. To illustrate the point, let $E = \mathrm{D}_e$ be the set of all philosophers (dead or alive) and let $S = \mathrm{D}_s$ be the set of natural numbers. We can (somewhat arbitrarily) define a *designated* element d_a from each type a by putting: d_e = Descartes, $d_s = 0$, $d_t = 1$, and $d_{ab}(\mathrm{u}) = d_b$, whenever a is a type and $\mathrm{u}\in \mathrm{D}_a$. We then let M^d be the model $((D_a)_{a\in \mathrm{T}}, \mathrm{F}^d, \mathrm{g}^d)$, where $\mathrm{F}^d(\mathrm{c}) = \mathrm{g}^d(\mathrm{x}) = d_a$, for arbitrary types a, $\mathrm{x}\in \mathrm{Var}_a$, and $\mathrm{c}\in \mathrm{Con}_a$. Now any two worlds n and $m \in S$ are lexically indistinguishable in the following sense: if α is a variable or constant of some category sa and a does not contain s, then $[\![\alpha(i)]\!]^{\mathrm{F}^d,\mathrm{g}^d[i/n]} = [\![\alpha(i)]\!]^{\mathrm{F},\mathrm{g}[i/m]}$, i.e. the two have the same extension. So M^d has infinitely many indices each of which corresponds to the same Ty1-model. Intuitively, the differences among these indices can be understood in terms of expressive power: Ty1 cannot distinguish between any two of these worlds, but maybe other languages can. Indeed, Ty2 is one of them because, e.g., the equation

(106) $(\mathrm{f}_{ss}(i) = i)$

is true in world 0 but false in every other world:

$\llbracket\ (f(i) = i\ )\ \rrbracket^{\mathrm{F}^d,\mathrm{g}^d[i/n]} = 1$
iff $\llbracket\ f \rrbracket^{\mathrm{F}^d,\mathrm{g}^d[i/n]}(\llbracket\ i\ \rrbracket^{\mathrm{F}^d,\mathrm{g}^d[i/n]}) = \llbracket\ i\ \rrbracket^{\mathrm{F}^d,\mathrm{g}^d[i/n]}$
iff $d_{ss}(\ n) = \ n$
iff $0 = n,$

because $d_{ss}(n) = d_s = 0$. (106) is a simple example of a formula expressing a *purely modal fact*, i.e. a formula whose truth in one world only depends on the relation that this world bears to others. Now, this example only shows that such purely modal facts *can* be expressed in Ty2. But are there any natural examples that also establish the *need* for their expressiblity? There is no simple and clear-cut answer to this question, which is one reason why we will leave the issue open; the other one is that these considerations would lead us far away from natural language semantics and into the metaphysics. But we do note that the replacement of worlds by models is not entirely unproblematic.

How about replacing models by worlds, then? There may be several ways of doing so but the most popular of them employs *modal realism*, i.e. the assumption that there are possible worlds apart from our reality. Given a realistic attitude towards possible worlds, one can replace the central notions of model-theoretic semantics by corresponding modal concepts: instead of models we would thus have different possible worlds on which the extensions depend, logical implication would become *strict implication*, i.e. subsethood among the propositions expressed, logical constants would literally have the same extension in every world, etc. And, most importantly, there is only one interpretation, i.e. only one function assigning meanings or intensions to non-logical constants. Thus, e.g., a word like ***sheep*** (or the constant corresponding to it) can be assigned that unique function from possible worlds to sets of individuals that yields the set of sheep in any world *w* to which it is applied. And this absoluteness of intension is common to all expressions that do not contain any free variables. And this brings us back to (U). For the interpretation of

(107) $[\lambda i\ \lambda p\ \ S^*_i(\mathrm{b},p)]$

does not contain a free variable and should thus denote one fixed property of propositions, viz. the property of implying everything that Bill said. In view of the special case (101) of (U), then, we know that the following formula must be true:

(108) $(\exists p_{st})\ (\forall q)\ \neg\ (\exists i)\ \ [S^*_i(b,q) \leftrightarrow (p = q)]$

(108) is equivalent to the negation of (101): the rejection of (U) still goes through if we assume D_s to be the fixed set of all possible worlds, as we would do as modal realists. Now (108) is not just true in some artificial model, but true *simpliciter*, so that there must be some set of possible worlds *p* satisfying:

(109) $\neg\ (\exists i)\ S^*_i(b,p)$

which is a direct consequence of (108). (For simplicity we are using the Ty2-variable '*p*'as a meta-variable for propositions.) According to (109), it is impossible for Bill to utter a sentence which expresses *p* without saying more than that. Now which set could *p* be? It seems that whatever we may pick, it is at least conceivable that we have a possible situation (or even world) in which Bill utters one weird sentence of some weird language that expresses just this proposition. But what could this strange, unspeakable *p* be? A check of the definition of the diagonal proposition used to reject (U) in section 5.4 reveals that the following *English* sentence comes pretty close to expressing it:

(110) ***Whatever Bill say is false.***

which shows that, under the present account of propositional attitudes, there is a close connection between the liar paradox and the diagonal argument that was used to refute (U). Now the difference between the model-theoretic and the realistic approach becomes important because the former does not make any commitment as to which proposition is expressed by (110) or, indeed, which propositions satisfy (109): this depends on the model. Whether this is actually an advantage of the model-theoretic approach or only a sign of its vagueness will, however, be left open here.

5.7 Remarks on Montague's IL

The type-theoretic approach to intension sketched in the present part was originally formulated by by Montague in the late sixties and has since then been a standard tool in the logical analysis of natural language. It must, however, be pointed out that Montague and most of those who followed him did not use Ty2 as a medium of indirect interpretation but rather a somewhat more complex language usually referred to as *Intensional Logic* or *IL*. The present section gives a short sketch of IL from a Ty2 point of view. In particular, it will be shown how to express IL-formulae in Ty2 and thus be able to read the relevant literature.

The notions of a type and an ontology are almost identical to the corresponding concepts in Ty2. The only difference is that, whereas s behaves completely analogous to e in Ty2, occurrences of s in IL-types are restricted to functional domains. This reflects the fact that we only need indices to let extensions (of arbitrary types) depend on them. The definition of an IL-type, then, replaces the clause '$S \rightarrow s$' by the rule '$S \rightarrow (s+ S)$'. And the corresponding clause in the definition of an ontology is just the special case of the scheme for D_{ab} when a happens to be s:

(111)$D_{(sb)} = D_b{}^S$,

where S is the set of indices.

The lexical expressions are as in Ty2 except, of course, that there are no constants or variables of non-IL types (like s, ss, es, $(es)t$, etc.). However this similarity is deceiving. For although IL-variables get interpreted by assignments that map them on extensions of the same type, the notion of a model (as the interpretation of constants) is quite different from that in Ty2: the interpretation F of an IL-model assigns *intensions* rather than extensions to constants $c \in \mathrm{Con}_a$:

(112) $F(c) \in D_{(sa)}$ $(= D_a{}^S)$

An IL-constant of type a is thus interpreted like a Ty2-constant of type sa. In fact, in translating IL-formulae into Ty2 we better assign to a constant $c \in \mathrm{Con}_a$ a constant $c^+ \in \mathrm{Con}_{sa}$. But it should be kept in mind that, syntactically speaking, an IL-constant of type a is an expression of category a, not sa. We will soon see how it still gets its intended interpretation.

To build complex IL-expressions one can use Application, Abstraction, and Identity as in Ty2 but, of course, restricted to IL-expressions. In particular, the following Ty2-formulae are not IL-expressions because s is not an IL-type and, consequently, i and j $(\in \mathrm{Var}_s)$ are not IL-variables:

(113) (a) $c_{se}(i)$

(b) $[\lambda i\ x]$

(c) $(i = j)$

However, there is no doubt that we need expressions like (113a) and (113b) in our translations. In order to capture them, IL has various notational devices. To begin with, functional application to the variable $i \in \mathrm{Var}_s$ is expressed by a *cup operator* ‘$\breve{}$’ preceding the functor: if α is an IL-formula of some category sb (where b is an IL-type), then $\ulcorner \breve{}\alpha \urcorner$ is of category b and we will soon see that it gets interpreted in the indicated way. Similarly, abstraction from i gets expressed by the *cap operator* ‘$\hat{}$’: $\ulcorner \hat{}\alpha \urcorner$ is of category sb whenever α is an IL-formula of category b. Since we have seen that, in principle, we do not need any variables of type s other than i, the two operators actually suffice to express anything we have done so far. Thus the fact that IL does not allow for formulae like (113c) when i and j are distinct variables of type s, is irrelevant for our purposes, or so it seems.

We can now give a precise definition of the

<u>Syntax of IL</u>:

For any IL-type a the set IL_a of IL-expressions of category a is defined by the following recursion:

(Lex) $(\mathrm{Var}_a \cup \mathrm{Con}_a) \subseteq \mathrm{IL}_a$;

(Id) if $\alpha \in \mathrm{IL}_a$ and $\beta \in \mathrm{IL}_a$, then $\ulcorner(\alpha = \beta)\urcorner \in \mathrm{IL}_t$;

(App) if $\alpha \in \mathrm{IL}_{ab}$ and $\beta \in \mathrm{IL}_a$, then $\ulcorner\alpha(\beta)\urcorner \in \mathrm{IL}_b$;

(Abs) if a is an IL-type, $x \in \mathrm{Var}_a$, and $\alpha \in \mathrm{Ty}_b$,
then $\ulcorner[\lambda x\ \alpha]\urcorner \in \mathrm{IL}_{ab}$;

(Cup) if $\alpha \in \mathrm{IL}_{sb}$, then $\ulcorner \breve{}\alpha \urcorner \in \mathrm{IL}_b$;

(Cap) if $\alpha \in \mathrm{Ty}_b$, then $\ulcorner \hat{}\alpha \urcorner \in \mathrm{IL}_{sb}$.

Instead of directly defining models for IL, we will interpret the formula indirectly, by translating them into Ty2. (This is not the way it is usually done in the literature, but the results are equivalent, and our procedure is simpler, in various respects.) We thus assign to any IL-expression α a Ty2-expression α^* of the same category. To begin with, we have to translate the lexical expressions, which is easy in the case of variables: any IL-variable x gets translated by itself: $x^* = x$. With the constants, a little bit of care must be taken. For, as we have pointed out, an IL-constant c of some category a is meant to correspond to a Ty2-constant c^+ of category sa. But if we simply *translate* c by c^+, we won't get what we want, because the two are of different categories. On the other hand, we can think of the ordinary IL-interpretation of c as assigning the *intension,* whereas its translation should express its extension, i.e. whatever we get when we apply the intension to the actual world. We can thus let c* be the Ty2-expression $c^+(i)$. Given what we have said so far, the rest of the procedure is straightforward:

Translation of IL into Ty2:

The following recursion assigns every IL-expression α a Ty2-expression α* of the same category:

(Lex) if α is a variable, then $\alpha^* = \alpha$;
if α is a constant, then $\alpha^* = \alpha^+(i)$;

(Id) if $\alpha \in IL_a$ and $\beta \in IL_a$, then $\ulcorner(\alpha = \beta)\urcorner^* = \ulcorner(\alpha^* = \beta^*)\urcorner$;

(App) if $\alpha \in IL_{ab}$ and $\beta \in IL_a$, then $\ulcorner\alpha(\beta)\urcorner^* = \ulcorner\alpha^*(\beta^*)\urcorner$;

(Abs) if a is an IL-type, $x \in Var_a$, and $\alpha \in Ty_b$,
then $\ulcorner[\lambda x\ \alpha]\urcorner^* = \ulcorner[\lambda x\ \alpha^*]\urcorner$ (= $\ulcorner[\lambda x^*\ \alpha^*]\urcorner$) ;

(Cup) if $\alpha \in IL_{sb}$, then $\ulcorner[\breve{}\alpha]\urcorner^* = \ulcorner\alpha^*\ (i)\urcorner$;

(Cap) if $\alpha \in Ty_b$, then $\ulcorner[\hat{}\alpha]\urcorner^* = \ulcorner[\lambda i\ \alpha^*]\urcorner$.

This indirect interpretation of IL has first been given by Daniel Gallin (in his 1972 dissertation); but the whole construction of IL makes it clear that Montague must have been aware of its possibility. With Gallin's translation procedure in mind, it is now easy to understand arbitrary IL-formulae like Montague's version of the unspecific reading of ***John possibly believes that some unicorn neighs*** :

(114) $\Diamond\, B(j,[\hat{}\,\exists x\,[\,U(x)\ \&\ N(x)\,]\,]\,)$,

which translates into:

(114*) $(\exists i)\ B^+{}_i(j^+{}_i,\ [\lambda i\ \exists x\,[\,U_i(x)\ \&\ N_i(x)\,]\,]\,)$

As one can guess from this example, the *diamond operator* is just a shorthand notation for the existential quantifier over i: '◊φ' translates as '(∃i) φ*'; ◊ is IL-expressible by the reduction techniques discussed in part 4. Similarly, the *box operator* '' expresses universal quantification over indices (denoted by i). The two expressions are occasionally claimed to express certain (*metaphysical*) senses of the English words ***possibly*** and ***necessarily***, respectively.

A closer look at (114*) shows that it contains the constant $j^+ \in Con_{sa}$ as corresponding to the proper name ***John***, where we would expect (and have hitherto used) a constant of type *e*. The reason for this does not lie in some aspect of the meaning of ***John*** covered by the IL-formula (114). Rather, the world-dependant constant c^+ appears because there are no world-independent constants in IL. This is clearly a nuisance because, as we have seen, the extension of a proper name should not change with the situations described in an intensional construction. In order to capture this fact in an IL-translation one would thus have to assume a particular meaning postulate for the translations c of proper names:

(115) $(\exists x)\ (c = x)$

This so-called *rigidity postulate* says that the intension of c must be a constant function from indices to individuals. This becomes clear when we look at the Ty2-version of (115):

(115*) $(\exists x)\ (\forall i)\ (c^+(i) = x)$

No such postulate was necessary in our above Ty2-treatment of proper names, because we treated them as exceptions to the usual constant-applied to index scheme (33) of lexical translation.

We have seen that Montague's IL can be thought of as (a notational variant of) a certain sub-language of Ty2. However, this does not mean that it also shares all the nice and important formal features that we know Ty1 and Ty2 to have. In particular, contrary to Ty2, IL does not obey the laws of λ-conversion. To see why this is so, it suffices to present a counter-example:

(116) $[\lambda x_e\ [\,\hat{}\ P(x)\,]\,]\,(c)$

We take P and x to be constants of category *et* and *e*, respectively. Applying λ-conversion would yield:

(117) $[\,\hat{}\ P(c)\,]$

which is *not* equivalent to (116), as becomes clear from comparing the respective translations:

(116*) $[\lambda x_e\, [\, \lambda i\ P^+_i(x)\,]\,]\, (c^+_i)$

(117*) $P^+_i(c^+_i)$

The transition from (116*) to (117*) clearly violates the variable-condition on λ-conversion; the correct result, after renaming, would be:

(118) $[\, \lambda j\ P^+_j(c^+_i)\,]$

which is not the translation of any IL-formula.

In spite of the failure of ordinary λ-conversion it is possible to formulate a restricted version of this law that is valid throughout IL, but would block in cases like (116). (Unfortunately, this version of λ-conversion does not possess the diamond property.) The idea is to express the variable-condition on the invisible *i* in terms of syntactic features of IL-expressions. However, we will not do so here, for it would lead us too far away; and if in doubt about the applicability of λ-conversion in some particular case, one should always look at the corresponding Ty2-formulae anyway.

In view of these difficulties one may wonder why IL has ever been proposed as a medium of indirect interpretation. A likely answer lies in its *restricted expressibility*: the very fact that one can perform indirect interpretation within a highly restricted sub-language of two-sorted type theory may have its deep reason in some conceptual feature of natural language. The use of IL would thus be seen as expressing the hypothesis that human language does not exceed a certain logical complexity. Alas, such a view is mistaken. For virtually anything that can be expressed in Ty2 can also be expressed in IL, albeit in a more roundabout way. Instead of going into the proof of this fact, we quote one example of a formula that looks as if it essentially employs nested variable-binding of the sort not available in IL:

(119) $[\lambda i\ [\lambda j\ (i = j)\ ]\ ]$

(Note that (119) is of category *s*(*st*), which is also an IL-type, and that it does not contain any free variables.) In order to get an IL-version of (119), we first reformulate the equation by means of Leibniz's Law, which brings us closer to IL, where identity among worlds cannot be expressed directly:

(120) $[\lambda i\ [\lambda j\ (\ [\lambda p_{st}\ p_i] = [\lambda p_{st}\ p_j])\ ]\ ]$

Now we observe that, due to λ-conversion, the subformula (121) of (120) is equivalent to (121'):

(121) $[\lambda j\ (\ [\lambda p_{st}\ p_i] = [\lambda p_{st}\ p_j])\ ]$

(121') $[\lambda X_{(st)t}\ [\lambda j\ \ (X = [\lambda p_{st}\ p_j])\ ]\ ]\ ([\lambda p_{st}\ p_i])$

No variable confusion arises, because i is free within (121). Since the two above formulae are logically equivalent, we can replace (121) by (121') in (120) and obtain:

(122) $[\lambda i\ [\lambda X_{(st)t}\ [\lambda j\ \ (X = [\lambda p_{st}\ p_j])\ ]\ ]\ ([\lambda p_{st}\ p_i])\ ]$

Now we can rename bound variables:

(123*) $[\lambda i\ [\lambda X_{(st)t}\ [\lambda i\ \ (X = [\lambda p_{st}\ p_i])\ ]\ ]\ ([\lambda p_{st}\ p_i])\ ]$

and it is not difficult to see that the result is the *-translation of:

(123) $[\hat{}\ [\lambda X_{(st)t}\ [\hat{}\ \ (X = [\lambda p_{st}\ \check{}p])\ ]\ ]\ ([\lambda p_{st}\ \check{}p])\ ]$

-

Exercises

16. The purpose of this exercise is to tighten intuitions about the rigidity of proper names. The following table partially describes hypothetical situations in which two individuals (to whom we neutrally refer as '*a*' and '*b*') carry various names and differ with respect to their wealth:

	s_1		s_2		s_3		s_4	
a:	name	rich?	name	rich?	name	rich?	name	rich?
	John	yes	***John***	yes	***John***	no	***John***	no
b:	name	rich?	name	rich?	name	rich?	name	rich?
	Frank	yes	***Frank***	no	***Frank***	yes	***Frank***	no

	s_5		s_6		s_7		s_8	
a:	name	rich?	name	rich?	name	rich?	name	rich?
	Frank	yes	***Frank***	no	***Frank***	yes	***Frank***	no
b:	name	rich?	name	rich?	name	rich?	name	rich?
	John	yes	***John***	yes	***John***	no	***John***	no

We now reveal the real identity of *a* and *b*: *a* is Mick Jagger (i.e. the person we normally refer to as ***Mick Jagger***), whereas *b* is Frank Sinatra (the famous singer). Now determine the following three sets of situations among s_1 - s_8:

S_0: the set of situations in which Mick Jagger is rich;
S_1: the set of situations in which Frank Sinatra is rich;
S_0: the set of situations in which a person whose name (in those situations) is ***Frank*** is rich.

17. Let B and K be Ty2-constants of category *s*((*st*)(*et*)) standing for the attitudes of belief and knowledge, respectively. Try to give an approximate decomposition of the verb ***doubt*** in terms of these two constants.

18. Describe a situation of which the *de dicto* reading of

(55) ***Bill says that Coco loves a baby.***

is true but the *de re* reading is not.

19. Show that the two readings ascribed to (62):

(62) ***Bill says that Coco loves every baby.***
(*dd*) $S_i(b,[\lambda i\ (\forall y)\ [B_i(y) \rightarrow L_i(c,y)]\ ]\ )$
(*dr*) $(\forall y)\ [B_i(y) \rightarrow S_i(b,[\lambda i\ L_i(c,y)]\ )\ ]$

are indeed of the general forms (61):

(61) (*dd*) $A_i(x,[\lambda i\ (Q_i y)\ P_i(y)])$
(*dr*) $(Q_j y)\ A_i(x,[\lambda i\ P_i(y)])$;

i.e. replace ‘x’ by ‘b’ and ‘A’, ‘P’, and ‘Q’ by Ty2-expressions α, β, and γ (of the appropriate types) such that the results will be logically equivalent to the formulae in (62).

20. Apply the decomposition (80) to (]) and show that the result is logically equivalent to the *de re* analysis of (⟧).

(80) ***seek***' = [λy λx T_i(x, [λi F_i(x,y)])]

(]) ***Alain seeks a dinosaur.***

(⟧) ***Alain tries to find a dinosaur.***

21. Give the translations of the two readings (*de re* and *de dicto*) of:

(85) ***Ernie owes Bert a horse-size cookie.***

as based on the paraphrase:

(85') ***Ernie is obliged to give Bert a horse-size cookie.***

22. The sentence

Julius worships a Greek goddess.

obviously does not imply the existence of goddesses. Does that mean that ***worship*** is referentially opaque? If so, find a suitable paraphrase. If not, find a suitable inference distinguishing ***worship*** from ***seek***.

23. Give a lexical decomposition of ***to be*** as generalized to the case of a binary relation among NP_{quant}-intensions; as in part 4, ***to be*** should be taken as expressing identity among individuals.

24. Which of the following two schematic IL-formulae is valid?

(a) $([{}^{\vee}[{}^{\wedge}\alpha]] = \alpha)$

(b) $([{}^{\wedge}[{}^{\vee}\beta]] = \beta)$

Note that, for (b) to be well-formed, β must be of some category *sb*.

Solutions exercises

1. Show that, under the assumption that predicate logic formulae denote truth-values, the substitutional interpretation (!) of quantification is non-compositional. Hint: Assume that the extension of a given predicate P is neither empty nor identical to the universe of discourse D and consider the formulae $(\exists x)\, P(x)$ and $(\exists x)\, \neg P(x)$.

Solution:

Let P be a predicate as described in the hint, i.e. $\emptyset \neq \llbracket P \rrbracket \neq \mathrm{D}$. Both $\llbracket P(x) \rrbracket$ and $\llbracket \neg P(x) \rrbracket$ must be truth-values and one of them, $\llbracket \varphi \rrbracket$, must be 0, by the interpretation of ¬. Moreover, by the (standard) interpretation (!) of existential quantification, we find that $\llbracket (\exists x) P(x) \rrbracket = \llbracket (\exists x)\, \neg P(x) \rrbracket = 1 = \llbracket (\exists x)\varphi \rrbracket$. But if we replace φ by any contradiction, the result is 0: $\llbracket (\exists x) [P(x) \wedge \neg P(x)] \rrbracket = 0$, because $\llbracket [P(x) \wedge \neg P(x)]\, [{}^{x}/_{a}] \rrbracket = \llbracket [P(a) \wedge \neg P(a)] \rrbracket = 0$, no matter which name a we pick. On the other hand, $\llbracket \varphi \rrbracket = 0 = \llbracket [P(x) \wedge \neg P(x)] \rrbracket$, i.e. the two parts have the same meaning (= truth-value!). We have thus found a counter-example to compositionality.

-

2. Show that there is no compositional treatment of relative clauses that meets (A1) - (A3). Hint: Assume that the extension of the noun ***president*** is a singleton {b} and that its intersection with that of the relative clause ***is wise*** is empty. Then consider the NPs ***every president***, ***some president***, ***every president who is wise*** and ***some president who is wise***.

Solution:

$$\begin{aligned} & \llbracket \textit{\textbf{every president}} \rrbracket \\ = \; & \{X \subseteq D \mid \llbracket \textit{\textbf{president}} \rrbracket \subseteq X\} \\ = \; & \{X \subseteq D \mid \{b\} \subseteq X\} \\ = \; & \{X \subseteq D \mid b \in X\}. \end{aligned}$$

$$\begin{aligned} & \llbracket \textit{\textbf{some president}} \rrbracket \\ = \; & \{X \subseteq D \mid \llbracket \textit{\textbf{president}} \rrbracket \cap X \neq \emptyset\} \\ = \; & \{X \subseteq D \mid \{b\} \cap X \neq \emptyset\} \\ = \; & \{X \subseteq D \mid b \in X\}. \end{aligned}$$

Hence:

(*) $[\![$***every president***$]\!] = [\![$***some president***$]\!]$.

But:

$[\![$***every president who is wise***$]\!]$

$= \{X \subseteq D \mid [\![$***president who is wise***$]\!] \subseteq X\}$

$= \{X \subseteq D \mid [\![$***president***$]\!] \cap [\![$***who is wise***$]\!] \subseteq X\}$

$= \{X \subseteq D \mid \varnothing \subseteq X\}$

$= \wp(D)$.

$[\![$***some president who is wise***$]\!]$

$= \{X \subseteq D \mid [\![$***president who is wise***$]\!] \cap X \neq \varnothing\}$

$= \{X \subseteq D \mid [\![$***president***$]\!] \cap [\![$***who is wise***$]\!] \cap X \neq \varnothing\}$

$= \{X \subseteq D \mid \varnothing \cap X \neq \varnothing\}$

$= \varnothing$.

Thus:

(**) $[\![$***every president who is wise***$]\!] \neq [\![$***some president who is wise***$]\!]$.

But, according to (A1), we should be able to get the former by combining $[\![$***every president***$]\!]$ with $[\![$***who is wise***$]\!]$ and the latter by combining $[\![$***some president***$]\!]$ with $[\![$***who is wise***$]\!]$, which is impossible because of (*).

-

3. Show that, at least for classical propositional logic, the definition of negation given on p. 10f. is correct: any formula φ negates a formula ψ if and only if φ is *logically equivalent to* $\neg\psi$, i.e. if $[\![\varphi]\!]^g = [\![\neg\psi]\!]^g$ for any truth-value assignment g. You may assume that self-contradictions always get the truth-value 0 and that the valid formulae are the tautologies. Hint: One direction is simple. The other one is to show that $[\![\varphi]\!]^g = [\![\neg\psi]\!]^g$ for any g; it is best to distinguish the cases $[\![\varphi]\!]^g = 1$ and $[\![\varphi]\!]^g = 0$ and use one of the two properties of negation in each case.

Solution:

The simple direction first: If φ is logically equivalent to $\neg\psi$, then φ and ψ together imply the self-contradiction $[\varphi\wedge\psi]$, because $[\![\varphi\wedge\psi]\!]^g = [\![\varphi]\!]^g \times$

$[\![\psi]\!]^g = [\![\neg\psi]\!]^g \times [\![\psi]\!]^g = 0$, for any assignment g. Moreover, if a formula χ is true under all assignments that make ψ true and under all assignments that make φ true (and thus ψ false), χ is true under any assignment whatsoever and hence valid.

Now for the other direction: We assume that φ negates ψ (in the sense of p. 10f.) and must show that $[\![\varphi]\!]^g = [\![\neg\psi]\!]^g$, for any truth-value assignment g. But if $[\![\varphi]\!]^g = 1$, $[\![\psi]\!]^g$ cannot be 1, because otherwise g would make a self-contradiction true. Hence $[\![\psi]\!]^g = 0$, i.e. $[\![\neg\psi]\!]^g = 1 = [\![\varphi]\!]^g$. For the other case, we first observe that both φ and ψ imply $[\varphi\vee\psi]$, by the usual truth-table for disjunction ($\vee$). So $[\varphi\vee\psi]$ must be valid, because φ negates ψ. Now assume that $[\![\varphi]\!]^g = 0$. From the validity of $[\varphi\vee\psi]$ we conclude: $1 = [\![\varphi\vee\psi]\!]^g = \max([\![\varphi]\!]^g,[\![\psi]\!]^g) = \max(0,[\![\psi]\!]^g)$, and so $[\![\psi]\!]^g = 1 = [\![\varphi]\!]^g$.

-

4. Given any universe D ($\neq \emptyset$), what should be the generalized quantifier $[\![$ ***nothing***$]\!]$ $\subseteq \wp(D)$? Which function $f_{\textbf{\textit{nothing}}} \in D_{(et)t}$ does it correspond to?

Solution:

$[\![\ulcorner$***nothing*** VP$\urcorner]\!]$ should come out as true if the VP's extension does not contain any 'counter-examples', i.e. if it is empty; otherwise the sentence should be false. So:

$$\begin{aligned} & [\![\textbf{\textit{nothing}}]\!] \\ = \quad & \{X \subseteq D_e \mid \neg\exists x \in D_e\text{: } x \in X\} \\ = \quad & \{X \subseteq D_e \mid X = \emptyset\} \\ = \quad & \{\emptyset\} \end{aligned}$$

$\{\emptyset\}$ is characterized by that function $f_{\textbf{\textit{nothing}}}$ that maps $\chi^e_\emptyset$ into 1 and everything else onto 0; using the fact that $\chi^e_\emptyset$ is the function that maps every $x \in D_e$ onto 0, we obtain the following characterization of $f_{\textbf{\textit{nothing}}}$:

$$f_{\boldsymbol{nothing}}(\chi) = \begin{cases} 0, \text{ if } \chi(\mathrm{x}) = 1, \text{ for some } \mathrm{x} \in D_e; \\ 1, \text{ if } \chi(\mathrm{x}) = 0, \text{ for all } \mathrm{x} \in D_e. \end{cases}$$

5. The following clause relates any binary truth-functional connective K to a corresponding function f_K of type $(t(tt))$:

f_K(v)(u) = K(u,v), whenever u and v are truth-values.

Thus every K corresponds to a binary relation among truth-values. Specify the relations thus corresponding to $\wedge$ (conjunction), $\vee$ (disjunction),and $\rightarrow$ (material implication).

<u>Solution:</u>

For $K = \wedge$ we get:

$$f_\wedge(0) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} = \chi_\varnothing;$$

$$f_\wedge(1) = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} = \chi_{\{1\}}.$$

So $f_\wedge$ corresponds to the binary relation:

$$\begin{aligned} & \{(\mathrm{u},\mathrm{v}) \mid \text{ u is in the set characterized by } f_\wedge(\mathrm{v})\} \\ = \; & \{(1,1)\}, \end{aligned}$$

because 1 ∈ {1} (= the set characterized by $f_\wedge(1)$) but no other pair (u,v) satisfies the condition. – If K is disjunction, we have:

$$f_\vee(0) = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} = \chi_{\{1\}};$$

$$f_\vee(1) = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \chi_{\{0,1\}}.$$

So $f_\vee$ corresponds to:

{(1,0), (0,1), (1,1)}.

Finally, *K* can be material implication:

$$f_{\rightarrow}(0) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \chi_{\{0\}}\ (= \neg!);$$

$$f_{\rightarrow}(1) = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \chi_{\{0,1\}}.$$

So $f_{\rightarrow}$ corresponds to:

{(0,0), (0,1), (1,1)}.

6. Here is yet another derivation of (20) in Lambek's Calculus:

(!)

L1: $e, e(et) \rightarrow et$

L2: $e, e, e(et) \rightarrow e, et$; L1: $e, et \rightarrow t$

L3: $e, e, e(et) \rightarrow t$

L4: $e, e(et) \rightarrow et$

L2: $e, e(et), (et)t \rightarrow et, (et)t$; L1: $et, (et)t \rightarrow t$

L3: $e, e(et), (et)t \rightarrow t$

L4: $e(et), (et)t \rightarrow et$

Apply the method of indexing according to object/subject positions to determine whether (!) corresponds to *XY*.

Solution:

Indexing the *e*s turns (!) into:

L1: $e_o, e_o(e_st) \rightarrow e_st$

L2: $e_s, e_o, e_o(e_st) \rightarrow e_s, e_st$; L1: $e_s, e_st \rightarrow t$

L3: $e_s, e_o, e_o(e_st) \rightarrow t$

L4: $e_o, e_o(e_st) \rightarrow e_st$

L2: $e_o, e_o(e_st), (e_st)t \rightarrow e_st, (e_st)t$; L1: $e_st, (e_st)t \rightarrow t$

L3: $e_o, e_o(e_st), (e_st)t \rightarrow t$

L4: $e_o(e_st), (e_st)t \rightarrow e_ot$

So the outcome is as in (22) and thus *not XY.*

-

7. The type shift:

($\uparrow$) $e(et) \rightarrow n(nt)$,

where n is the type $(et)t$ of quantified NPs is usually attributed to Montague. Show that one can derive ($\uparrow$) in Lambek's Calculus. Try to find a derivation that makes the leftmost n correspond to the object position. Hint: Assume the derivation (22) for

$e(et), n \rightarrow et$

and then take the second n into account.

Solution:

We do the indexing as we go along deriving ($\uparrow$). Since we already have (22), we may start our derivation with it:

(22): $e_o(e_st), (e_st)t \rightarrow e_ot$

L2: $(e_ot)t\ , e_o(e_st), (e_st)t \rightarrow (e_ot)t\ , e_ot$; L1: $(e_ot)t, e_ot \rightarrow t$

L3: $(e_ot)t\ , e_o(e_st), (e_st)t \rightarrow t$

L4: $(e_ot)t\ , e_o(e_st) \rightarrow ((e_st)t\ )t$

L4: $e_o(e_st) \rightarrow ((e_ot)t)\ ((e_st)t\ )t))$

-

8. Show that $L_a = D_a$ whenever a does not contain any e. Hint: Proceed inductively starting with $a = t$; for complex types bc you may then assume that $L_b = D_b$ and that $L_c = D_c$.

Solution:

Starting the induction is trivial, because π_t is always id_t, and so $L_t = D_t$. So we assume that $a = (bc)$ and that we already know that (*) $L_b = D_b$ and $L_c = D_c$. In order to show that $L_{bc} = D_{bc}$, we pick some $f \in D_{bc}$ and set out to show that it satisfies:

(1) $\pi_{bc}(f) = f$,

for any permutation π. So we fix π and pick an arbitrary $u \in D_b$ for which we must prove that the two functions in (1) agree, i.e. that:

(2) $\pi_{bc}(f)(u) = f(u)$.

Note that our inductive hypothesis (*) tells us that $u \in L_a$ and that $f(u) \in L_b$, so that we conclude:

(3) $\pi_b(u) = u$,

(4) $\pi_c(f(u)) = f(u)$,

by the definition of L_a and L_b. Now (2) follows because:

$\pi_{bc}(f)(u) = f(u)$

iff $(u, f(u)) \in \pi_{bc}(f)$, by notational convention, iff $(\pi_b(u),\pi_c(f(u))) \in f$, by (31)(c),

iff $(u,f(u)) \in f$, by (3) and (4),

iff $f(u) = f(u)$, by notational convention,

which certainly is the case.

-

9. Show that the characteristic functions of Ø and D_a are in L_{et}.

Solution:

Given a permutation π and some $u \in D_e$, we must show that:

(i) $\pi_{et}(\chi^e_\emptyset)(u) = \chi^e_\emptyset(u)$;

(ii) $\pi_{et}(\chi^e_{D_e})(u) = \chi^e_{D_e}(u)$.

But this is easy:

$\pi_{et}(\ \chi^e_\emptyset)(u) = \chi^e_\emptyset(u)$

iff $(u,\ \chi^e_\emptyset(u)) \in \pi_{et}(\ \chi^e_\emptyset)$, by notational convention,

iff $(\pi_e(u),\ \pi_t(\ \chi^e_\emptyset(u))) \in \chi^e_\emptyset$, by (31)(c),

iff $(\pi_e(u),\ \pi_t(0)) \in \chi^e_\emptyset$, by definition of $\chi^e_\emptyset$,

iff $(\pi_e(u),\ 0) \in \chi^e_\emptyset$, by (31)(b),

iff $\chi^e_\emptyset(\pi_e(u)) = 0$, by notational convention,

which is the case because $\chi^e_\emptyset(v) = 0$ for *any* $v \in D_e$. Similarly, we have:

$\pi_{et}(\chi^e_{D_e})(u) = \chi^e_{D_e}\ (u)$

iff $(u,\ \chi^e_{D_e}\ (u)) \in \pi_{et}(\ \chi^e_{D_e})$, by notational convention,

iff $(\pi_e(u),\ \pi_t(\ \chi^e_{D_e}\ (u))) \in \chi^e_{D_e}$, by (31)(c),

iff $(\pi_e(u),\ \pi_t(1)) \in \chi^e_{D_e}$, by definition of $\chi^e_{D_e}$,

iff $(\pi_e(u),\ 1 \in \chi^e_{D_e}$, by (31)(b),

iff $\chi^e_{D_e}\ (\pi_e(u)) = 1$, by notational convention,

which is again trivially true.

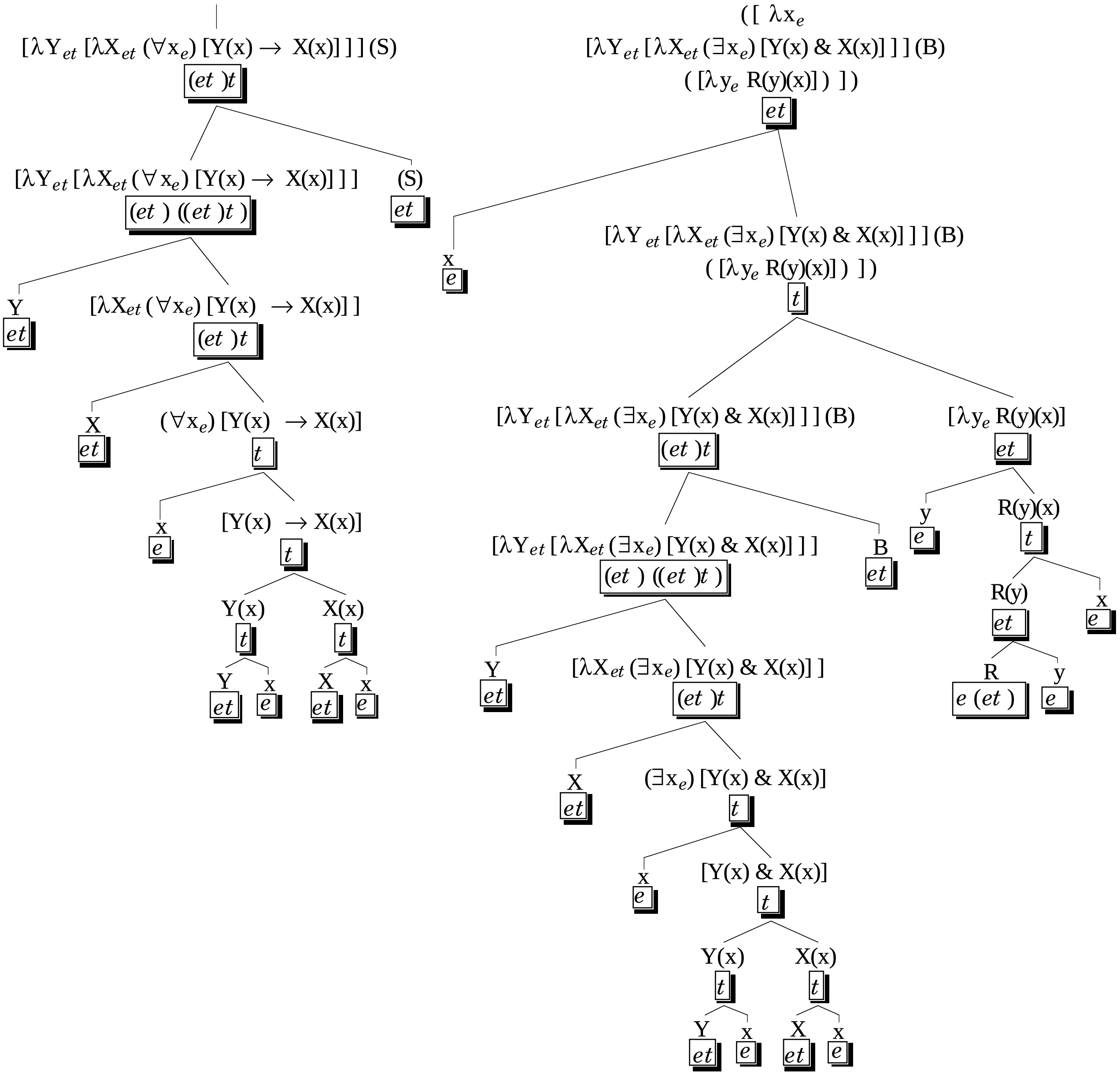
10. Let B and S be constants of category *et*, and let R be a constant of category *e*(*et*). Show that (15'), i.e.:

$$[\lambda Y_{et}\ [\lambda X_{et}\ (\forall x_e)\ [Y(x) \rightarrow X(x)]\ ]\ ]\ (S)$$
$$(\ [\ \lambda x_e\ [\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]\ (B)\ (\ [\lambda y_e\ R(y)(x)]\ )\ ]\ )$$

is a Ty1-expression of category *t*. [Hint: First draw a tree indicating the structure of (15') and then use the syntactic rules of Ty1 to recursively determine the categories of its sub-expressions.]

Solution:

The syntactic analysis is represented by way of a *Montague Tree* that gives the resulting expression, its category and the relevant syntactic operation for each node. The latter has been omitted since it is always obvious. Moreover, abbreviations from predicate logic have not been undone. The tree thus treats conjunction as a combination of two expressions of category *t* into another expression of that category and combines φ of categorry *t* and a variable x directly into ⌜(∀x) φ⌝ of category *t*, etc.


$[\lambda Y_{et}\,[\lambda X_{et}\,(\forall x_e)\,[Y(x) \rightarrow X(x)]\,]\,]\,(S)$
$(et)t$
$[\lambda Y_{et}\,[\lambda X_{et}\,(\forall x_e)\,[Y(x) \rightarrow X(x)]\,]\,]$
$(et)((et)t)$
(S)
et
Y
et
$[\lambda X_{et}\,(\forall x_e)\,[Y(x) \rightarrow X(x)]\,]$
$(et)t$
X
et
$(\forall x_e)\,[Y(x) \rightarrow X(x)]$
t
x
e
$[Y(x) \rightarrow X(x)]$
t
$Y(x)$
t
$X(x)$
t
Y
et
x
e
X
et
x
e
$([\,\lambda x_e$
$[\lambda Y_{et}\,[\lambda X_{et}\,(\exists x_e)\,[Y(x)\,\&\,X(x)]\,]\,]\,(B)$
$(\,[\lambda y_e\,R(y)(x)]\,)\,]\,)$
et
x
e
$[\lambda Y_{et}\,[\lambda X_{et}\,(\exists x_e)\,[Y(x)\,\&\,X(x)]\,]\,]\,(B)$
$(\,[\lambda y_e\,R(y)(x)]\,)\,]\,)$
t
$[\lambda Y_{et}\,[\lambda X_{et}\,(\exists x_e)\,[Y(x)\,\&\,X(x)]\,]\,]\,(B)$
$(et)t$
$[\lambda y_e\,R(y)(x)]$
et
$[\lambda Y_{et}\,[\lambda X_{et}\,(\exists x_e)\,[Y(x)\,\&\,X(x)]\,]\,]$
$(et)((et)t)$
B
et
y
e
$R(y)(x)$
t
$R(y)$
et
x
e
R
$e(et)$
y
e
Y
et
$[\lambda X_{et}\,(\exists x_e)\,[Y(x)\,\&\,X(x)]\,]$
$(et)t$
X
et
$(\exists x_e)\,[Y(x)\,\&\,X(x)]$
t
x
e
$[Y(x)\,\&\,X(x)]$
t
$Y(x)$
t
$X(x)$
t
Y
et
x
e
X
et
x
e

11. In this exercise, you will have to show that the two restrictions on the Substitution Principle also apply to λ-Conversion:

(a) Find a model $((D_a)_{a \in T}, F, g)$ and an expression $\ulcorner[\lambda x\ \alpha]\ (\beta)\urcorner$ such that $[\![[\lambda x\ \alpha]\ (\beta)]\!]^{F,g} \neq [\![\alpha']\!]^{F,g}$, where α' is the result of replacing *all* occurrences (bound or free) of x in α by β. Hint: You can adapt the example used in connection with the Substitution Principle but you would still have to present a concrete model.

(b) Find a model in which

$[\lambda x_t\ [\lambda y_t\ (x = y)\ ]\ ]\ (y)$

and

$[\lambda y_t\ (y = y)\ ]$

have different extensions.

Solution:

(a): α may be the expression:

$$[\lambda x_e\ [\ Q_{et}(f_{ee}(x))\ \&\ (\forall x)\ P(f(x))\ ]\ ]$$

of category *et*, β can be the variable $y \in Var_e$. So α' is:

$$[\ Q_{et}(f_{ee}(y))\ \&\ (\forall x)\ P(f(y))\ ]\ ]$$

We only give the relevant parts of the model; the rest is arbitrary. The domain D of individuals should contain at least *a* and *b* (where $a \neq b$) and the values of the assignment g for the free variables in the above formulae are:

$g(y) = a$;
$g(P) = g(Q) = \chi_{\{a\}}$, i.e. the characteristic function of the singleton $\{a\}$;
$g(f) = id_e$, i.e. the identity function on D_e.

Informally, α(β) says that g(y)'s value under id_e is in Q's extension and (ii) P's extension is the universe. Since (ii) is false, so must be the whole formula. Using our official semantic rules, the reasoning runs as follows:

$$\begin{aligned}
& \llbracket \alpha(\beta) \rrbracket^{F,g} \\
= \;& \llbracket [\lambda x_e \, [\, Q_{et}(f_{ee}(x)) \,\&\, (\forall x)\, P(f(x)) \,]\,](y) \rrbracket^{F,g} \\
= \;& \llbracket [\lambda x_e \, [\, Q_{et}(f_{ee}(x)) \,\&\, (\forall x)\, P(f(x)) \,]\,] \rrbracket^{F,g} (\llbracket y \rrbracket^{F,g}) \\
= \;& \llbracket [\, Q_{et}(f_{ee}(x)) \,\&\, (\forall x)\, P(f(x)) \,] \rrbracket^{F,g[x/\llbracket y \rrbracket^{F,g}]} \\
= \;& \llbracket [\, Q_{et}(f_{ee}(x)) \,\&\, (\forall x)\, P(f(x)) \,] \rrbracket^{F,g[x/g(y)]} \\
= \;& \llbracket [\, Q_{et}(f_{ee}(x)) \,\&\, (\forall x)\, P(f(x)) \,] \rrbracket^{F,g[x/a]} \\
= \;& \llbracket Q_{et}(f_{ee}(x)) \rrbracket^{F,g[x/a]} \times \llbracket (\forall x)\, P(f(x)) \rrbracket^{F,g[x/a]}
\end{aligned}$$

We show that the right factor is 0:

$$\begin{aligned}
& \llbracket (\forall x)\, P(f(x)) \rrbracket^{F,g[x/a]} = 1 \\
\text{iff} \;& \llbracket P(f(x)) \rrbracket^{F,g[x/a][x/u]}, \text{ for all } u \in D_e, \\
\text{iff} \;& \llbracket P(f(x)) \rrbracket^{F,g[x/u]}, \text{ for all } u \in D_e,
\end{aligned}$$

because:

$$\begin{aligned}
& g[x/a][x/u] \\
= \;& (\, g[x/a] \setminus \{(x,g[x/a](x))\}\,) \cup \{(x,u)\} \\
= \;& (\underline{g[x/a]} \setminus \{(x,a)\}) \cup \{(x,u)\} \\
= \;& (\,(\, g \setminus \{(x,g(x))\}\,) \underline{\cup \{(x,a)\} \setminus \{(x,a)\}}) \cup \{(x,u)\}\,) \\
= \;& (\, g \setminus \{(x,g(x))\}\,) \cup \{(x,u)\} \\
= \;& g[x/u]
\end{aligned}$$

But, clearly, $\llbracket P(f(x)) \rrbracket^{F,g[x/b]} = 0$, because:

$$\begin{aligned}
& \llbracket P(f(x)) \rrbracket^{F,g[x/b]} \\
= \;& \llbracket P \rrbracket^{F,g[x/b]} (\llbracket f \rrbracket^{F,g[x/b]} (\llbracket x \rrbracket^{F,g[x/b]})) \\
= \;& g[x/b]\,(P)\,(g[x/b]\,(f)\,(g[x/b]\,(x)\,)) \\
= \;& g(P)\,(g(f)\,(b)) \\
= \;& \chi_{\{a\}}\,(id_e\,(b)) \\
= \;& \chi_{\{a\}}\,(b),
\end{aligned}$$

which is 0, because b is distinct from a. Thus $\llbracket (\forall x)\, P(f(x)) \rrbracket^{F,g[x/a]}$ is 0 and hence so is $\llbracket \alpha(\beta) \rrbracket^{F,g}$.

It remains to be shown that $\llbracket \alpha' \rrbracket^{F,g} = 1$, i.e. that $\llbracket Q_{et}(f_{ee}(y)) \rrbracket^{F,g} = \llbracket (\forall x)\, P(f(y))\,]\,] \rrbracket^{F,g} = 1$. The left conjunct is straightforward:

$$
\begin{aligned}
& [\![Q(f(y))]\!]^{F,g} \\
= \; & [\![Q]\!]^{F,g}([\![f]\!]^{F,g}([\![y]\!]^{F,g})) \\
= \; & g(Q)(g(f)(g(y))) \\
= \; & \chi_{\{a\}}(\mathrm{id}_e(a)) \\
= \; & \chi_{\{a\}}(a) \\
= \; & 1.
\end{aligned}
$$

In order to prove that $[\![(\forall x)\ P(f(y))\ ]\ ]]\!]^{F,g} = 1$, we pick arbitrary $u \in D_e$ and show that $[\![P(f(y))\ ]\ ]]\!]^{F,g[x/u]} = 1$. The idea is, of course, that the truth-value does not depend on the modification of the variable assignment:

$$
\begin{aligned}
& [\![P(f(y))\ ]\ ]]\!]^{F,g[x/u]} \\
= \; & [\![P]\!]^{F,g[x/u]}([\![f]\!]^{F,g[x/u]}([\![y]\!]^{F,g[x/u]})) \\
= \; & g[x/u](P)(g[x/u](f)(g[x/u](y))) \\
= \; & g(P)(g(f)(g(y))) \\
= \; & \chi_{\{a\}}(\mathrm{id}_e(a)) \\
= \; & 1.
\end{aligned}
$$

<u>(b)</u>: Any model would do:

$$
\begin{aligned}
& [\![[\lambda x_t\,[\lambda y_t\,(x = y)\,]\,]\,(y)]\!]^{F,g} \\
= \; & [\![[\lambda x_t\,[\lambda y_t\,(x = y)\,]\,]]\!]^{F,g}([\![y]\!]^{F,g}) \\
= \; & [\![[\lambda y_t\,(x = y)\,]]\!]^{F,g[x/[\![y]\!]^{F,g}]},
\end{aligned}
$$

which characterizes the singleton set $\{g(y)\}$. On the other hand, $[\![[\lambda y_t(y = y)]]\!]^{F,g}$ is the characteristic function of D_t, which contains two elements.

12. Assume that ***individual*** translates into the Ty1-expression:

$\ulcorner[\lambda x_e\,(x = x)]\urcorner$.

Now use the rules given in 4.5 to translate the (38) and (38') into Ty1 and show that they are equivalent to the same formula of predicate logic *without identity*.

(38) ***Every cow is four-legged.***

(38') ***Every cow is a four-legged individual.***

You can make use of all reduction principles discussed in 4.4, (including laws of predicate-logic), but you must make every reduction (including renaming of bound variables) explicit.

Solution:

every' = $[\lambda Y_{et}\,[\lambda X_{et}\,(\forall x_e)\,[Y(x) \to X(x)]\,]\,]$
, by (24)

cow' = C
($\in \mathrm{Con}_{et}$)

$[[$***every***$]_{\mathrm{Det}}\,[$***cow***$]_{\mathrm{N}}\,]_{\mathrm{NP}_{\mathrm{quant}}}$'

= $[\lambda Y_{et}\,[\lambda X_{et}\,(\forall x_e)\,[Y(x) \to X(x)]\,]\,]\,(C)$
, by (29)

≈ $[\lambda X_{et}\,(\forall x_e)\,[C(x) \to X(x)]\,]\,]$
, by λ-conversion

four-legged' = F
($\in \mathrm{Con}_{et}$)

is' = $[\lambda y_e\,[\lambda x_e\,(x = y)]\,]$
, by (42)

$[[$***is***$]_{\mathrm{Cop}}\,[$***four-legged***$]_{\mathrm{Adj}}\,]_{\mathrm{VP}}$'

= $[\lambda x_e\,(\exists y_e)\,[\,[\lambda y_e\,[\lambda x_e\,(x = y)]\,]\,(x)\,(y)\ \&\ F(y)]$
, by (47)

≈ $[\lambda x_e\,(\exists y_e)\,[\,[\lambda z_e\,(z = x)]\,(y)\ \&\ F(y)]$
, by critical λ-conversion

≈ $[\lambda x_e\,(\exists y_e)\,[\,(y = x)\ \&\ F(y)]$
, by λ-conversion

≈ $[\lambda x_e\,F(x)]$

, by a law of identity

≈ F

, by η-conversion

(38) [[***every cow***]$_{\text{NP}_{\text{quant}}}$ [***is four-legged***]$_{\text{VP}}$]'

≈ $[\lambda X_{et}\ (\forall x_e)\ [C(x) \rightarrow X(x)]\ ]\ ]\ (F)$

, by (28)

≈ $(\forall x_e)\ [C(x) \rightarrow F(x)]\ ]$

, by λ-conversion

[[***four-legged***]$_{\text{Adj}}$ [***individual***]$_{\text{N}}$]$_{\text{N}}$'

= $[\lambda X_{et}\ [\lambda Y_{et}\ [\lambda x_e\ [X(x)\ \&\ Y(x)]\ ]\ ]\ ]\ (F)\ ([\lambda x\ (x = x)])$

, by (36')

≈ $[\lambda x_e\ [F(x)\ \&\ (x = x)]]$

, by three λ-conversions

≈ $[\lambda x_e\ F(x)\ ]$

, by a law of identity and propositional logic

≈ F

, by η-conversion

a' = $[\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]$

, by (39)

[[***a***]$_{\text{Det}}$ [***four-legged individual***]$_{\text{N}}$]$_{\text{NP}_{\text{quant}}}$'

≈ $[\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]\ (F)$

, by (47)

≈ $[\lambda X_{et}\ (\exists x_e)\ [F(x)\ \&\ X(x)]\ ]$

, by λ-conversion

[[***is***]$_{\text{Cop}}$ [***a four-legged individual***]$_{\text{NP}_{\text{quant}}}$]$_{\text{VP}}$'

= $[\lambda x_e[\lambda X_{et}\ (\exists x_e)\ [F(x)\ \&\ X(x)]\ ]\ ([\lambda y_e\ [\lambda y_e\ [\lambda x_e(x = y)]\ ]\ (y)\ (x)$
$])\ ]$

, by (44)

≈ $[\lambda x_e\ (\exists z_e)\ [F(z)\ \&\ [\lambda y_e\ [\lambda y_e\ [\lambda x_e(x = y)]\ ]\ (y)\ (x)\ ](z)]\ ]$

, by critical λ-conversion

≈ $[\lambda x_e(\exists z_e)\ [F(z)\ \&\ [\lambda y_e\ (x = y)\ ](z)]\ ]$

, by two λ-conversions

≈ $[\lambda x_e (\exists z_e) [F(z) \,\&\, (x = z)]\,]$

, by one λ-conversion

≈ $[\lambda x_e\ F(x)]$

, by predicate logic with identity

≈ F

, by η-conversion

(38') $[[$***every cow***$]_{NP_{quant}}$ $[$***is a four-legged individual***$]_{VP}\ ]'$

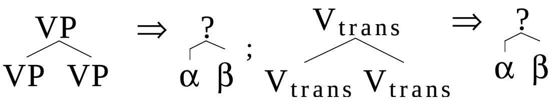
≈ $[\lambda X_{et}\ (\forall x_e)\ [C(x) \rightarrow X(x)]\ ]\ ]\ (F)$

, by (28)

≈ $(\forall x_e)\ [C(x) \rightarrow F(x)]\ ]$

, by λ-conversion

13. Give translation rules for VP and V_{trans} disjunction:



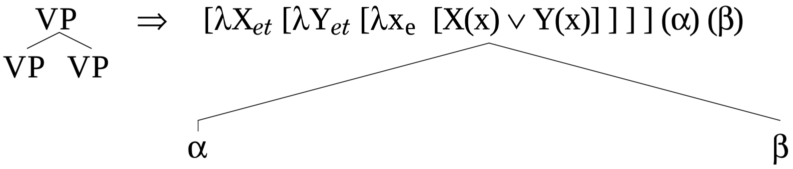
Show the correctness of your translations by applying them to (51) and (52):

(51) ***Caroline hugs Alain or kisses Tom.***

(52) ***Caroline hugs or kisses Tom.***

Solution:

(*)



Alain' = a
$(\in \mathrm{Con}_e)$

hugs' = H
$(\in \mathrm{Con}_{e(et)})$

$[[$***hugs***$]_{V_{trans}}$ $[$***Alain***$]_{NP_{ref}}$ $]_{VP}' = H(a)$
, by (30)

Tom' = t
$(\in \mathrm{Con}_e)$

kisses' = K
$(\in \mathrm{Con}_{e(et)})$

$[[$***kisses***$]_{V_{trans}}$ $[$***Tom***$]_{NP_{ref}}$ $]_{VP}' = K(t)$
, by (30)

$[[$***hugs Alain***$]_{VP}$ ***or*** $[$***kisses Tom***$]_{VP}$ $]_{VP}'$

$= \quad [\lambda X_{et}\, [\lambda Y_{et}\, [\lambda x_e\ [X(x) \vee Y(x)]\ ]\ ]\ ] \quad (\ H\ (\ a\ )\ ) \qquad (\ K\ (\ t\ )\ )$
, by (*)

$\approx \quad [\lambda x_e\ [H(x,a) \vee K(x,t)]\ ],$

by two λ-conversions and one notational convention

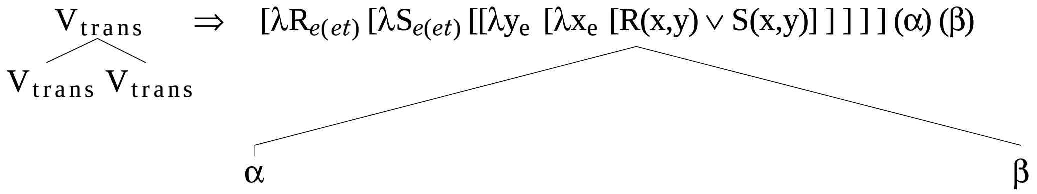
Caroline' = c (∈ Con$_e$)

(51) [[***Caroline***]$_{NP_{ref}}$ [***hugs Alain or kisses Tom***]$_{VP}$]$_S$'

$\approx$ $[\lambda x_e\, [H(x,a) \lor K(x,t)]\,]\, (c)$
, by (27)

$\approx$ $[H(c,a) \lor K(c,t)]\,]$,
by λ-conversion

(#)


V trans
V trans V trans
⇒
[λR e(et) [λS e(et) [[λy e [λx e [R(x,y) ∨ S(x,y)]]]]] (α) (β)
α
β

[[***hugs***]$_{V_{trans}}$ ***or*** [***kisses***] $_{V_{trans}}$] $_{V_{trans}}$'

= $[\lambda R_{e(et)}\, [\lambda S_{e(et)}\, [[\lambda y_e\, [\lambda x_e\, [R(x,y) \lor S(x,y)]\,]\,]\,]\,]$ (H) (K)
, by (#)

$\approx$ $[\lambda y_e\, [\lambda x_e\, [H(x,y) \lor K(x,y)]\,]$, by two λ-conversions

[[***hugs or kisses***] $_{V_{trans}}$ [***Tom***]$_{NP_{ref}}$] $_{V_{trans}}$'

$\approx$ $[\lambda y_e\, [\lambda x_e\, [H(x,y) \lor K(x,y)]\,]\, (t)$
,by (30)

$\approx$ $[\lambda x_e\, [H(x,t) \lor K(x,t)]\,]$
,by (30)

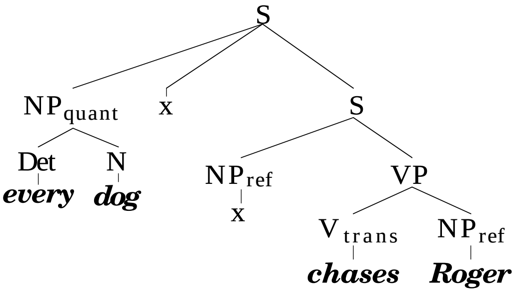
(52) [[***Caroline***]$_{NP_{ref}}$ [***hugs or kisses Tom***]$_{VP}$]$_S$'

$\approx$ $[\lambda x_e\, [H(x,t) \lor K(x,t)]\,]\, (c)$
, by (27)

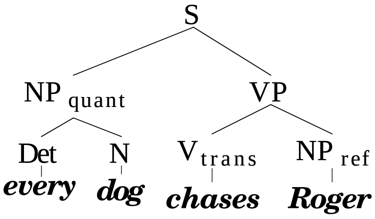
$\approx$ $[H(c,t) \lor K(c,t)]\,]$,
by λ-conversion

14. Translate (57) and (57') and show that each is equivalent to (58):

(57)



(57')



(58) $(\forall x)\ [D(x) \rightarrow C(x,r)]$

Solution:

$\boldsymbol{every}' = [\lambda Y_{et}\ [\lambda X_{et}\ (\forall x_e)\ [Y(x) \rightarrow X(x)]\ ]\ ]$
, by (24)

$\boldsymbol{dog}' = D$
$(\in \mathrm{Con}_{et})$

$[[\boldsymbol{every}]_{\mathrm{Det}}\ [\boldsymbol{dog}]_{\mathrm{N}}\ ]_{\mathrm{NP}_{\mathrm{quant}}}{}'$

$= \quad [\lambda Y_{et}\ [\lambda X_{et}\ (\forall x_e)\ [Y(x) \rightarrow X(x)]\ ]\ ]\ (D)$
, by (29)

$\approx \quad [\lambda X_{et}\ (\forall x_e)\ [D(x) \rightarrow X(x)]\ ]\ ]$
, by λ-conversion

$x' = x$
$(\in \mathrm{Var}_e)$

$\boldsymbol{chases}' = C$
$(\in \mathrm{Con}_{e(et)})$

Roger' = r
$(\in \mathrm{Con}_e)$

$[[$***chases***$]_{V_{\mathrm{trans}}}$ $[$***Roger***$]_{NP_{\mathrm{ref}}}$ $]_{VP}$' = C(r)
, by (30)

$[\ [x]_{NP_{\mathrm{ref}}}$ $[$***chases Roger***$]_{VP}$ $]_S$'
= $C(r)(x)$
, by (27)
= $C(x,r)$
, by notational convention

(57):

$[\ [$***every dog***$]_{NP_{\mathrm{quant}}}$ x [x ***chases Roger***$]_S$]'
≈ $[\lambda X_{et}\ (\forall x_e)\ [D(x) \rightarrow X(x)]\ ]\ ]\ ([\lambda x\ C(x,r)])$
, by (55)
≈ $(\forall x_e)\ [D(x) \rightarrow [\lambda x\ C(x,r)](x)]\ ]$
, by λ-conversion
≈ $(\forall x_e)\ [D(x) \rightarrow C(x,r)]\ ]$
, by λ-conversion
= **(58)**

(57'):

$[\ [$***every dog***$]_{NP_{\mathrm{quant}}}$ $[$***chases Roger***$]_{VP}$]'
≈ $[\lambda X_{et}\ (\forall x_e)\ [D(x) \rightarrow X(x)]\ ]\ ]\ (C(r))$
, by (28)
≈ $(\forall x_e)\ [D(x) \rightarrow C(r)(x)]\ ]\ ]$
, by λ-conversion
≈ $(\forall x_e)\ [D(x) \rightarrow C(x,r)]\ ]$
, by notational convention
= **(58)**

▪

15. Interpret the Lambek proof (!) from exercise 6 (part 3) by associating a Ty1-definition with it:

(!) L1: $e, e(et) \to et$

L2: $e, e, e(et) \to e, et$; L1: $e, et \to t$

L3: $e, e, e(et) \to t$

L4: $e, e(et) \to et$

L2: $e, e(et), (et)t \to et, (et)t$; L1: $et, (et)t \to t$

L3: $e, e(et), (et)t \to t$

L4: $e(et), (et)t \to et$

Solution:

$$\text{L1:}\ \begin{array}{lll} e, & e(et) & \to et \\ y_e & R_{e(et)} & R(y) \end{array}$$

$$\text{L2:}\ \begin{array}{llll} e, & e, & e(et) & \to e, et \\ x_e, & y_e & R_{e(et)} & x, R(y) \end{array} \qquad \text{L1:}\ \begin{array}{lll} e, & et & \to t \\ z_e & X_{et} & X(z) \end{array}$$

$$\text{L3:}\ \begin{array}{llll} e, & e, & e(et) & \to t \\ x_e, & y_e & R_{e(et)} & [\lambda X_{et}\,[\lambda z_e\, X(z)\,]\,]\,(R(y))\,(x)\ \ (\approx \underline{R(y)(x)\,!}) \end{array}$$

$$\text{L4:}\ \begin{array}{ll} e, e(et) & \to et \\ y_e, R_{e(et)} & [\lambda x_e\, R(y)(x)]\ \ (\approx \underline{R}\,!\,) \end{array}$$

$$\text{L2:}\ \begin{array}{ll} e,\ e(et),\ (et)t & \to et, (et)t \\ y_e,\ R_{e(et)},\ P_{(et)t} & R, P \end{array} \qquad \text{L1:}\ \begin{array}{lll} et, & (et)t & \to t \\ X_{et}, & Q_{(et)t} & Q(X) \end{array}$$

$$\text{L3:}\ \begin{array}{ll} e,\ e(et),\ (et)t & \to t \\ y_e,\ R_{e(et)},\ P_{(et)t} & [\lambda Q_{(et)t}\ [\lambda X_{et}\, Q(X)\,]\,]\,(P)\,(R)(y))\ \ (\approx \underline{P(R(y))}\,) \end{array}$$

$$\text{L4:}\ \begin{array}{ll} e(et),\ (et)t & \to et \\ R_{e(et)},\ P_{(et)t} & [\lambda y_{et}\, P(R(y))\,] \end{array}$$

So the Ty1-definition corresponding to (!) is:

$$[\lambda P_{(et)t}\,[\lambda R_{e(et)}\ [\lambda y_e\, P(R(y))\,]\ \ ]\,]\,.$$

16. The purpose of this exercise is to tighten intuitions about the rigidity of proper names. The following table partially describes hypothetical situations in which two individuals (to whom we neutrally refer as '*a*' and '*b*') carry various names and differ with respect to their wealth:

	s_1		s_2		s_3		s_4	
a:	name	rich?	name	rich?	name	rich?	name	rich?
	John	yes	***John***	yes	***John***	no	***John***	no
b:	name	rich?	name	rich?	name	rich?	name	rich?
	Frank	yes	***Frank***	no	***Frank***	yes	***Frank***	no

	s_5		s_6		s_7		s_8	
a:	name	rich?	name	rich?	name	rich?	name	rich?
	Frank	yes	***Frank***	no	***Frank***	yes	***Frank***	no
b:	name	rich?	name	rich?	name	rich?	name	rich?
	John	yes	***John***	yes	***John***	no	***John***	no

We now reveal the real identity of *a* and *b*: *a* is Mick Jagger (i.e. the person we normally refer to as ***Mick Jagger***), whereas *b* is Frank Sinatra (the famous singer). Now determine the following three sets of situations among s_1 - s_8:

S_0: the set of situations in which Mick Jagger is rich;

S_1: the set of situations in which Frank Sinatra is rich;

S_0: the set of situations in which a person whose name (in those situations) is ***Frank*** is rich.

Solution:

$S_1 = \{s_1, s_2, s_5, s_6\}$; $S_2 = \{s_1, s_3, s_5, s_7\}$; $S_1 = \{s_1, s_3, s_5, s_6\}$.

-

17. Let B and K be Ty2-constants of category *s*((*st*)(*et*)) standing for the attitudes of belief and knowledge, respectively. Try to give an approximate decomposition of the verb ***doubt*** in terms of these two constants.

Solution:

If you have doubts, you are not a believer. If you know the contrary to be the case, you are no longer in doubt. So the following decomposition of ***doubt*** may be quite close:

$$\lambda p_{st}\, \lambda x\, [\neg B(x,p)\ \&\ \neg K(x, [\ \lambda j\ \neg p(j)\ ])\ ]$$

18. Describe a situation of which the *de dicto* reading of

(55) ***Bill says that Coco loves a baby.***

if true but the *de re* reading is not.

<u>Solution:</u>

Imagine a situation s_0 in which Bill uttered:

(58') (e) ***Coco loves a fat baby.***

and nothing else. The intension p_0 of (58') is the set of all situations in which Coco there is some baby that is fat and to which Coco stands in the relation of love. This p_0 is clearly a subset of the set p of all situations in which Coco loves some baby, i.e. the intension of ***Bill loves a baby***. So according to our criterion (57'), Bill stands in the relation of saying to p in s_0 and consequently (55) is true in s_0 on its *de dicto* reading:

(55) (*dd*) $S_i\,(b,[\lambda j\ \ (\exists x_e)\,[B_j\ \ (x)\ \&\ L_j\,(c,x)]\,]\,)$

In order to show that (55) is not true if taken *de re*, we must ensure that s_0 does not satisfy:

(55) (*dr*) $(\exists x_e)\,[B_i\ \ (x)\ \&\ K_i\ \ (b,[\lambda j\ \ L_j\,(c,x)\,]\,)]$

Using our criterion (57') again, this means that Bill must not utter a sentence in s_0 whose intension is a subset of some set q_b consisting of all situations in which Coco loves some individual b that happens to be a baby in s_0. Since (58') is the only sentence uttered by Bill in s_0, it only remains to be shown that its intension p_0 is not a subset of any such q_b, i.e. that for every baby b in s_0 there is some situation $s_b \in p_0$ such that $s_b \notin q_b$. However, this is certainly the case: for a given b we can easily imagine some s_b in which Coco loves some fat baby c but hates b. (In particular, c would have to be distinct from b!) This s_b will then be in p_0 (because c is a fat baby loved by Coco in s_b) without being in q_b (because Coco does not love b in s_b). QED.

-

19. Show that the two readings ascribed to (62):

(62) ***Bill says that Coco loves every baby.***
(*dd*) $S_i(b,[\lambda i\ (\forall y)\ [B_i(y) \rightarrow L_i(c,y)]\ ]\ )$
(*dr*) $(\forall y)\ [B_i(y) \rightarrow\ S_i(b,[\lambda i\ L_i(c,y)]\ )\ ]$

are indeed of the general forms (61):

(61) (*dd*) $A_i(x,[\lambda i\ (Q_i y)\ P_i(y)])$
(*dr*) $(Q_j y)\ A_i(x,[\lambda i\ P_i(y)])$;

i.e. replace ‘A’, ‘P’, and ‘Q’ by Ty2-expressions α, β, and γ (of the appropriate types) such that the results will be logically equivalent to the formulae in (62).

Solution:

If we put:

α = ‘S’;
β = ‘$[\lambda i\ \lambda y\ L_i(x,y)]$’;
and: γ = ‘$[\lambda i\ \lambda X\ (\forall y)\ [B_i(y) \rightarrow X(y)]\ ]$’ (= ***every baby***') ,

substitution gives us:

(61') (*dd*) $S_i(b,\ [\lambda i\ [\lambda i\ \lambda X\ (\forall y)\ [B_i(y) \rightarrow X(y)]\ ]\ (i)$
$(\ [\lambda y\ [\lambda i\ \lambda y\ L_i(x,y)]\ (i)\ (y)\ ]\ )\ ]\ )$
(*dr*) $[\lambda i\ \lambda X\ (\forall y)\ [B_i(y) \rightarrow X(y)]\ ]$
$(\ [\lambda y\ S_i(b,\ [\lambda i\ [\lambda i\ \lambda y\ L_i(x,y)]\ (i)\ (y)\ ]\ )]\ )$

The desired equivalences are then easy to verify using λ-reduction.

-

20. Apply the decomposition (80) to (]) and show that the result is logically equivalent to the *de re* analysis of (⟧).

(80) ***seek***' = $[\ \lambda y\ \lambda x\ T_i(\ x,\ [\lambda i\ F_i(x,y)\ ]\ )\ ]$
(]) ***Alain seeks a dinosaur.***
(⟧) ***Alain tries to find a dinosaur.***

Solution:

a dinosaur' = $[\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]\ (D_i)$
≈ $[\lambda X_{et}\ (\exists x_e)\ [D_i(x)\ \&\ X(x)]\ ]\ ]$
, by λ-conversion

seeks a dinosaur'

≈ [λx_e ***a dinosaur***' ([λy_e ***seek***'(x,y)])]

≈ [λx_e (∃y_e) [D_i(y) & ***seek***'(x,y)]] , by three conversions

≈ [λx_e (∃y_e) [D_i(y) & [λy λx T_i(x, [λ*i* F_i(x,y)])](x,y)]]

, by (80)

≈ [λx_e (∃y_e) [D_i(y) & T_i(x, [λ*j* F_j(x,y)])]]

Alain seeks a dinosaur'

≈ [λx_e (∃y_e) [D_i(y) & T_i(x, [λ*j* F_j(x,y)])]] (a)

≈ (∃y_e) [D_i(y) & T_i(a, [λ*j* F_j(a,y)])]

For the *de re* reading of (]) we proceed as follows:

find y'

= F_i(y)

try to find y'

= ***try*** '([λ*i* F_i(y)])

, by (77)

= [λ$P_{s(et)}$ λx_e T_i(x, [λ*i* P_i(x)])] ([λ*i* F_i(y)])

, by (75)

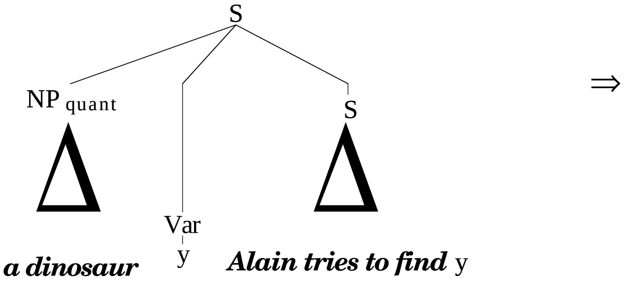
≈ [λx_e T_i(x, [λ*j* F_j(x,y)])]

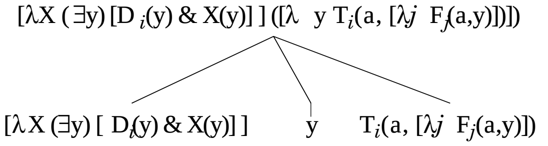
, by λ-reduction

Alain tries to find y'

≈ T_i(a, [λ*j* F_j(a,y)])

We can now apply the quantifying in rule:




[λX (∃y) [D i(y) & X(y)]] ([λ y Ti(a, [λj Fj(a,y)])])
[λX (∃y) [Di(y) & X(y)]]
y
Ti(a, [λj Fj(a,y)])

By λ-reduction, the resulting formula is equivalent to the translation of (]).

-

21. Give the translations of the two readings (*de re* and *de dicto*) of:

(85) ***Ernie owes Bert a horse-size cookie.***

as based on the paraphrase:

(85') ***Ernie is obliged to give Bert a horse-size cookie.***

Solution:

The lexical decomposition of ***owe*** is:

$$[\lambda z_e\ \lambda Q_{s((et)t)}\ \lambda x_e\ O_i(\ x,\ [\lambda i\ (Q_i y)\ G_i(x,y,z)\ ]\ )\ ],$$

where 'O(x,p)' can be read: 'x has the obligation to bring it about that p'; 'G_i' is the translation of ***give***, the second argument corresponding to the direct object. We then have:

a horse-size cookie'

$\approx$ $[\lambda Y_{et}\ [\lambda X_{et}\ (\exists x_e)\ [Y(x)\ \&\ X(x)]\ ]\ ]\ ([\lambda x\ [H_i(x)\ \&\ C_i(x)]])$

$\approx$ $[\lambda X_{et}\ (\exists x_e)\ [H_i(x)\ \&\ C_i(x)\ \&\ X(x)]\ ]$

owe Bert'

$=$ $[\lambda z_e\ \lambda Q_{s((et)t)}\ \lambda x_e\ O_i(\ x,\ [\lambda i\ (Q_i y)\ G_i(x,y,z)\ ]\ )\ ]\ (b)$

$\approx$ $[\lambda Q_{s((et)t)}\ \lambda x_e\ O_i(\ x,\ [\lambda i\ (Q_i y)\ G_i(x,y,b)\ ]\ )\ ]$

owe Bert a horse-size cookie'

$\approx\quad [\lambda Q_{s((et)t)}\, \lambda x_e\, O_i(\, x, [\lambda i\, (Q_i y)\, G_i(x,y,b)\,]\,)\,]$

$\qquad\qquad (\, \lambda i\, [\lambda X_{et}\, (\exists x_e)\, [H_i(x)\, \&\, C_i(x)\, \&\, X(x)\,]\,]\,)$

$\approx\quad [\lambda x_e\, O_i(\, x, [\lambda i\, (\exists y_e)\, [H_i(y)\, \&\, C_i(y)\, \&\, G_i(x,y,b)\,]\,]\,)\,]$

Ernie owes Bert a horse-size cookie'

$\approx\quad [\lambda x_e\, O_i(\, x, [\lambda i\, (\exists y_e)\, [H_i(y)\, \&\, C_i(y)\, \&\, G_i(x,y,b)\,]\,]\,)\,]\, (e)$

$\approx\quad O_i(\, e, [\lambda i\, (\exists y_e)\, [H_i(y)\, \&\, C_i(y)\, \&\, G_i(e,y,b)\,]\,]\,)$

This is the *de dicto* reading that can be glossed as: 'Ernie has (in the situation talked about) the obligation to bring it about that there is a horse-size cookie which Ernie gives to Bert'. The *de re* reading is obtained by quantifying ***a horse-size cookie***' into:

Ernie owes Bert y'

$\approx\quad O_i(\, e, [\lambda i\, G_i(e,y,b)\,]\,)\,]$

We thus get:

$[\lambda X_{et}\, (\exists x_e)\, [H_i(x)\, \&\, C_i(x)\, \&\, X(x)]\,]\, (\lambda y\, O_i(\, e, [\lambda i\, G_i(e,y,b)\,]\,))$

$\approx\quad (\exists y_e)\, [H_i(y)\, \&\, C_i(y)\, \&\, O_i(\, e, [\lambda i\, G_i(e,y,b)\,]\,)\,]$,

which can be glossed as 'There is a horse-size cookie such that Ernie has the obligation to bring it about that Ernie gives that cookie to Bert.'

-

22. The sentence

Julius worships a Greek goddess.

obviously does not imply the existence of goddesses. Does that mean that ***worship*** is referentially opaque? If so, find a suitable paraphrase. If not, find a suitable inference distinguishing ***worship*** from ***seek***.

Solution:

As Montague already noted, it seems to be impossible to reduce ***worship*** by paraphrase. However, in spite the failure of existential implication in ***worship*** must be due to something different than referential opacity. For indefinite objects still are taken to be specific rather than an arbitrary:

(|) ***Julius worships a Greek god.***
Every Greek god is mentioned in Martin's book.
Julius worships a god that is mentioned in Martin's book.

Note that every sentence of this inference may be true even if there are no gods. We now replace ***worship*** by ***seek***:

(2) ***Julius seeks a Greek god.***
Every Greek god is mentioned in Martin's book.
Julius seeks a god that is mentioned in Martin's book.

the inference only goes through if the first sentence is taken in its *de re* reading; on the *de dicto* reading, Martin would not be looking for any god in particular. But if the first premise is *de re*, it does imply the existence of Greek gods – unlike the first premise in (|). The sense in which ***worship*** does not entail existence thus seems to be rather different from the non-specificity of ***seek***.

-

23. Give a lexical decomposition of ***to be*** as generalized to the case of a binary relation among NP$_{\text{quant}}$-intensions; as in part 4, ***to be*** should be taken as expressing identity among individuals.

Solution:

The decomposition is:

$$[\ \lambda Q_{s((et)t)}\ \lambda Q'_{s((et)t)}\ (Q_i y)\ (Q'_i x)\ (x = y)\ ],$$

which is short for:

$$(M)\quad [\ \lambda Q_{s((et)t)}\ \lambda Q'_{s((et)t)}\ Q_i(\ [\ \lambda y\ \ Q'_i(\ [\ \lambda x\ (x = y)\ ]\ )\ ]\ )\ ],$$

which comes close to Montague's translation of ***to be***. Note that the shift from identity as a binary relation among individuals to (*M*) can essentially be obtained by interpreting the Lambek proof discussed in exercise 7.

-

24. Which of the following two schematic IL-formulae is valid?

(a) $([\breve{}\ [\hat{}\alpha]\ ] = \alpha)$
(b) $([\hat{}\ [\breve{}\beta]\ ] = \beta)$

Note that, for (b) to be well-formed, β must be of some category *sb*.

Solution:

The Ty2-translations of (a) and (b) are:

(a*) $(\ [\lambda i\ \alpha^*]\ (i) = \alpha\ )$
(b*) $(\ [\lambda i\ \alpha^*(i)\ ] = \alpha)$

(a*) is an instance of λ-conversion and hence valid. The law is sometimes called *Down-Up Cancellation*.

But (b*) can go wrong when α* contains a free occurrence of *i*. In fact, if α* is R_i where *i* is a constant of category *s*(*st*), (b*) becomes:

$$(\ [\lambda i\ R_i(i)\ ] = R_i),$$

which is equivalent to :

(!) $(\ [\lambda i\ R(i,i)\ ] = [\lambda j\ R(j,i)\ ]),$

by η-conversion. (!) expresses that one index stands in the relation expressed by 'R' to another index if and only if the latter bears that relation to itself. This condition is, e.g., violated by any non-empty, anti-symmetric relation.