2. SEMANTICS: the game of scope and intensionality

2.1 THE WAYS OF MEANING

Language refers. It cannot help doing so. As soon as it hits the mind – and is experienced as language – it is *about something other* than its forms. Language refers to something outside of itself. It is the only system with this property. Therefore, no other system needs to be interpreted the way language is interpreted. Language refers by encoding propositions – statements expressing the way of being of, or the state of affairs in, another system. This system we may call a *model*. A system is a model because it is described by a proposition, to the extent to which it is described by that proposition. For semantic analysis, it is irrelevant whether the model is independent of the proposition – that is a matter of philosophy, and has been intensively discussed in medieval logic. A proposition refers to a model, by definition. This way of *being about a model* we call meaning.

The interesting thing about propositional meaning is that it is not conventional. It is not given by some canonical prescription. Therefore, it is not a genuine topic of debate among language users. The meaning of a proposition is something on which those that know a language converge – on penalty of failed communication. In this respect, the meaning of a proposition is something completely different from the meaning of a word or a phrase. A proposition does not mean something *because* its phrases are point-wise meaningful: it means something *notwithstanding* the fact that we do not know or do not agree what the words or the phrases mean. That is: we do not know whether our interpretation of words is the same as someone else's – the speaker's, for example – and we do not need to know. To paraphrase Frege's *Nur im Zusammenhang eines Satzes bedeuten die Wörter etwas*: phrases mean something if

and only if they occur in meaningful propositions. The proposition's meaning is primary; the meaning of a phrase is derived.

In this respect, natural language essentially differs from formal languages. There we find finite, fixed and of course conventional interpretations for all meaningful phrases in a sentence. It would be almost perverse to come up with artificial languages the lexicon of which is semantically flexible. In natural languages, however, only functional elements – those that many speakers consider to be meaningless or indefinable – can boast that they are well-defined, to a certain degree. The compositionality of natural languages is so impressive that it obscures lexical indeterminism. Words do not mean a thing, but they are saved by the proposition; the proposition assigns types to the words, semantic roles to be played in a functional space. From this type, the nature of a phrase's contribution to the overall meaning is induced. This is how we discover, and never stop discovering, (aspects of) words and phrases, according to Bloom (2002). We are able to use and understand words to the extent to which we are able to use and understand sentences.

In this vein, it is intriguing that in Wierzbicka's Natural Semantic Metalanguage (NSM, *e.g.* Goddard 2002) words represent extremely complex concepts, the definition of which almost amounts to propositions – which we would expect for meanings of types <α, t>. Here follows, as an example, Goddard's (2002) analysis of a phrase of type *X is watching Y*:

(233) for some time X was doing something because X thought: when something happens in this place I want to see it; because X was doing this, X could see Y during this time

The idea that this meaning is built from universal conceptual atoms is less intriguing for the present exposal than the definitely propositional format which Goddard resorts to. Unfortunately, NSM hardly deals with the logical structure of phrasal meanings. In the logical framework of DELILAH to be discussed below, however, (233) would have to be converted into an extensive spell-out of logical units and structure:

(234) at some period *p* before *now*, there was an event *e* with agent X and place *pl* and there is a state of thinking *s* with agent X and theme *ts*
and
s was the reason for *e*
and

t is the proposition
(there is a state *w* of wanting with agent *X* and theme t*w*
and
tw is the property of X such that if there is an event *ev* at *pl* there is a state *si* of seeing with patient X and theme *ev*)
and
there is a state *se* of seeing with patient X and theme Y during *p*
and
there is a state *sp of* being possible with theme *se*
and
sp had reason *e*.

Not everyone will be convinced that (233) or (234) represents the canonical meaning of the verb *to watch* in its progressive form. One may even consider these to be defective, or overdone. Yet, the proposition-like structure of lexical meanings in this universalistic approach of NSM can be seen as a statement that we need propositions to register, encode and store lexical meanings, and consequently, that propositions underlie lexical meaningfulness. The meaning of a proposition is not an agglomeration of simpler or atomic meanings: it is generated by structure.

These remarks concern neither the organization of grammar nor the process of parsing and generation. In the production and processing of language, human beings rely on compiled routines rather than on philosophical hierarchies. In particular, the lexicon is the place where knowledge (experience, skills, insights, generalizations, guesses) of language aggregates to complex symbolizations – not necessarily elegant, not necessarily principled but extremely efficient. These very complex symbolizations reflect our versatile competence, including its semantic creativity.

The role of a lexicon in the *computation* of natural language is addressed in chapter 3. In the present chapter we are concerned with the way Dutch propositions are meaningful. There is not much that is particular about the meaning of Dutch. We do not have reason to assume that ways of being meaningful vary with the language. We must assume, however, that the propositional meaning is conveyed by the structure of the language, and thus by its particular grammar. This assumption is known as *compositionality* – according to Janssen (1986) wrongly attributed to Frege, but underlying all formal languages and as such effectuated for the interpretation of natural language by Richard Montague (*e.g.* Montague 1972) and many others. Compositionality implies that propositional meaning results from the way a proposition is

built. Since grammar is not debated among those who know a language, the construal of propositional meaning in that language is not either. Thus, the fundamental meaningfulness of a language depends both on the generality and on the particularity of grammar.

As it stands, we do not claim that bare structure is meaningful. We assume that the grammar both structures and labels sentences, that this combined effort induces types, and that these types – semantic functions – are sensitive to specification. It is not the semantic specification that is computed but the typological, functional semantic tier. That is where meaning grows and lives. Here is what we consider to be the backbone of meaning.

(235) labeled structure

$[[_{np}\ no_{det}\ dutchman_{n}]\ [_{ip}\ can_{aux}\ [_{vp}\ be_{aux}\ fooled]_{vp}\ always_{adj}]]]$

(extensional) types

<<et> <et>t> et <<et><et>> <<et><et>> et <<et><et>>,
<<et> <et>t> et <<et><et>> <<et><et>> et <t, t> ...

meanings

$f_{no}(g_{dutchman})\ (k_{always}\ (h_{can}(\ i_{be}\ (j_{fooled}))))$,
$f_{no}(g_{dutchman})\ (h_{can}\ (k_{always}\ (\ i_{be}\ (j_{fooled}))))$ ...

Our understanding of compositionality is not that there be a *function*al relation between labels, types and meanings. Compositionality in the montegovian sense sees to a well-defined relationship between the specifications of form and the specifications of meaning. In Montague (1972), this relationship was conceived of as a mapping between syntactic and semantic rules of construal. The rules of construal, however, were produced neither by mathematics nor by aesthetics or economy – they were opportunistic but operational. Though it is wise to look for principles and restrictions governing the rules of construal, compositionality is not a derivative of these constraints on grammar. In particular, it is not the case that syntactic labeling and semantic typing must be homomorphic for compositionality to be guaranteed. A functional relation between syntactic labeling and semantic typing is a valid anchor for compositionality, but it is neither necessary nor sufficient to entertain a computable relation between form and meaning. Approaches to natural-language semantics like Discourse Representation Theory (Kamp and Reyle 1993) and Head-Driven Phrase Structure Theory (*e.g.* Sag 2003) exploit all kinds of mechanisms, strategies and principles to assure compositionality. These instruments are beyond form-meaning homomorphisms of the sort elaborated in Hendriks (1993) and in type-logical grammar more generally (*cf.* Carpenter 1997, Morrill 1994).

This chapter offers a quite eclectic view on the construal of meaning. It defines three different but related layers of propositional semantics. It marks unification as the process feeding into semantic construal. It takes no position whatsoever on the meaning of particular items. In particular, it leaves open the possibility that lexical items have complex, proposition-like meanings – like (233) or (234) – which also contribute to the overall propositional semantics by unification. The abducted representations live on a much richer language than first-order predicate logic, in order to account for the variety of quantification, for event structure, for tense, mood, aspect and other issues beyond the borders of classical logic. The chapter also leaves room for post- and extra-derivational algorithms dealing with semantically defined constructions like anaphora and ellipsis. Moreover, the following sections will focus on formal entailment such as the litmus test for semantic representations. In short, the syntax-semantics interface may be more like a jungle than like a French garden.

The jungle enjoys some ordering, however, by a tight and intrinsic relationship between entailment and logical form. Entailment is a convergence of natural-language speakers on the necessity of the relationship between two sentences. An entailment is a converging judgement, and can be measured among language users. Logical form is a linguistic artefact, meant to model linguistic meaning. The following definition reflects the relationship between the entailment judgements and the calculus of logical form.

(236) *Logical Form*
[[X]] is *the* logical form of a sentence X if for some logic L,
S entails P *iff* $[\![S]\!] \vdash_L [\![P]\!]$ and $[\![S]\!] \models_L [\![P]\!]$.

The definition stresses the need for a well-defined deductive mechanism defined on logical forms that calculates entailments. We take the development of that logic to be a major task for semantic theory.

2.2 THE FORMS OF MEANING

Montague (1972) – hereafter: *PTQ* – introduced a package for meaning representation that we take as a basis for our efforts to interpret Dutch mechanically. The package includes

(237) higher-order functional terms with application and composition
semantic typing
intensionality and intensional embedding
full scoping
post-derivational meaning postulates.

The DELILAH semantics implements this package in the following way:

(238) higher-order functional terms with application and composition
implicit extensional semantic typing
intensional embedding
underspecified representation at derivation (but)
post-derivational, syntactically constrained unfolding of full representation.

In this sense, DELILAH is tributary of PTQ. By extending the 'fragment', however, with an account of peculiarities of Dutch, by focusing on inference and entailment and by adding many additional features, our system comes up with representations which are hardly recognizable as montegovian derivatives. Here is an example, reconstructing the diversity of modules in PTQ, with its intended restyling in DELILAH's various semantic formats; the comparison is enhanced by the fact that DELILAH uses English words to represent lexical concepts – just like PTQ.

(239) *Every man seeks a unicorn*
$\lambda\mathcal{P}.\ \forall z.\ [\textbf{man'}\{z\} \rightarrow \mathcal{P}\{z\}]$
$(\hat{}\lambda y.\ \textbf{try-to'}(y, \lambda\mathcal{Q}.\exists x.\ [\textbf{unicorn'}\{x\} \ \&\ \mathcal{Q}\{x\}](\hat{}\lambda z.\textbf{find'}(y, z)]))$

This formula is the semantic yield of a particular derivation of the sentence *every man seeks a unicorn* in PTQ. It says, roughly, that the function from intensional properties to propositions corresponding to *every man* is to be applied to the function from individual concepts to propositions that interprets *seeks a unicorn* as the relation *try-to* between individuals and the property of finding a unicorn. The (restyled) derivational product in DELILAH that corresponds to this formula and its derivation reads like (241); we abstracted away from all 'book-keeping' information which occurs in the machine, and just give the mere functional structure. It is formatted as a storage of lambda terms, called *Stored Logical Form*; its main structure is indicated in (240), by abstracting over all operators and lexical specification.

(240) *Stored Logical Form* < Store, Body> *abstract*

```
< [store0
<<[store1 man 1erots] every >>,
<<[store2 <<<[store3 <<<<[store4 unicorn 4erots], some >>>> 3erots],
find >>> 2erots], agent_of(U, F) & theme_of(U, P) & attime(U, W)
> 1erots] property(X(L))>>, try 0erots],
agent_of(D, I) & theme_of(D, Z) & attime(D, A) & tense(D, pres)
>
```

(241) *Stored Logical Form* < Store, Body> *full detail*

```
< [store0
<<[store1 λI.state(I, man) 1erots] λJ.λK. quant(I, every) & J(I) &
entails1(I, decr) & K(I) & entails(I,incr)>>,
<<[store2 <<<[store3 <<<<[store4 λP.state(P, unicorn) 4erots], λQ.λR.
quant(P, some) & Q(P) & entails1(P, incr) & and(R(P) & entails(P,
incr) >>>> 3erots],
λT.quant(U, some) & event(U, find) & entails1(U, incr) & T(U) &
entails(U,  incr)>>> 2erots],
λW.λU.λP.λF. agent_of(U, I) & theme_of(U, P) & attime(U, A) >
1erots],
λX.λY.quant(Z, the) & property(X(L)), Z) & entails1(Z, decr) &
Y(Z), entails(Z, incr)>>,
 λC. quant(D, some) & event( D, try) & entails1(D, incr) & C(D)
& entails(D, incr)
0erots],
λI.λZ.λD.λA. agent_of(D, I) & theme_of(D, Z) & attime(D, A) &
tense(D, pres) >
```

The structure establishes the following. There is a finite thematic signature of an agent and a theme to a *try*-event: the second element in the main structure <α,β>. The agent is a universal quantifier over *man:* the first element of *store0*. The theme is a quantifier over a property marking an intensional domain – there are no intensional types. This property is a thematic signature to an event *find*. This event is a signature over a free (but controlled) agent and a theme introduced by an existential quantifier over *unicorn.* In all cases, the target variables for quantification are identified with the bound variable, to compensate for the lost book-keeping of conversion. The specification of temporal objects is left unresolved, but they are identified in the two event-signatures. The structure of SLF is further discussed in section 2.6.3.

An immediate difference between (239) and (241) is that in (241) the scopal relation between the quantifiers involved is still underdetermined. The PTQ formula may, however, be subject to meaning postulates to the effect of reordering the dependencies between the existential and the universal quantifier and the intensional domain of the complement of *try.* Moreover, the lambda terms in the Stored Logical Form abound in analytical specifications from the

lexicon, whereas PTQ is very reticent in this respect. Partially, these specifications follow from adopting an event-structure analysis. Other specifications like *entail*, *entail1* and *property* are meta-predicates indicating semantic domains relevant to the spell-out of full scopal semantics. Here are the three fully-scoped analyses, called *Applied Logical Form,* which DELILAH derives from (241) by a post-derivational algorithm; they correspond to the scope orderings *every-intension-a*, *every-a-intension* and *a-every-intension*, respectively. The event-related existential quantifiers are not scoped, for reasons to be discussed in section 2.5.

(242) Applied Logical Form

(a)

```
quant(B,every).[man(B) ⟹ quant(C,some).[event(C, try) &
quant(D,the).[property(D).[ quant(A,some).[unicorn(A) &
quant(E,some).[event(E, beat) & agent_of(E,B) & theme_of(E,A) &
attime(E,F)]]] & agent_of(C,B) & theme_of(C,D) & attime(C,F) &
tense(C,pres)]]]
```

(b)

```
quant(B,every).[man(B) ⟹ quant(C,some).[event(C, try) &
quant(A,some).[unicorn(A) & quant(D,the).[property(D).[
quant(E,some).[event(E, beat) & agent_of(E,B) & theme_of(E,A) &
attime(E,F)]] & agent_of(C,B) & theme_of(C,D) & attime(C,F) &
tense(C,pres)]]]]
```

(c)

```
quant(A,some).[unicorn(A) & quant(B,every).[man(B) ⟹
quant(C,some).[event(C, try) & quant(D,the).[property(D).[
quant(E,some).[event(E, beat) & agent_of(E,B) & theme_of(E,A) &
attime(E,F)]] & agent_of(C,B) & theme_of(C,D) & attime(C,F) &
tense(C,pres)]]]]
```

In this representation, the event structure is naturally maintained, but the meta-predicates are mostly resolved. The quantification over *property* is preserved, though, as it marks intensional embedding in an extensionally typed logic. Apart from the lexical decomposition, this representation is on a par with the result of applying β-conversion and meaning postulates to (239):

(243) ∀x. [**man'**{x} → **try'**{x,^∃z. [**unicorn'**{z} & **find'**{x, z}]})

Apart from the 'classical' analysis (242), DELILAH derives another representation from (241), the *Flat Logical Form*. In this logical form, all scopal dependencies induced by quantification and intensionality are compiled onto every occurrence of every bound variable. Moreover, the scopal operators them-

selves are reduced to indices on the variables, with their main inferential attribute. For example: the clause `event(E+↑+some+[D], beat)` is to be read as follows.

(244) `event(E+↑+some+[D], beat) ::`
the (meta-)relation *event* holds between the bound variable *E* and the concept *beat*,
E is referentially (as for its valuation) dependent on the (bound) variable *D*,
E is bound by the quantifier *some* and here allows for *increasing* inferences with respect to the predicate of which it is an argument.

The remaining structure is essentially a flat conjunction of fully-specified small clauses of this nature, each of which is entailed under a certain regime accounting for referential dependency and quantification (see section 2.6.5). Below is the *Flat Logical Form* derived from (241) and equivalent to the family of formulas (242).

(245) Flat Logical Form

```
state(B+↓+every+[], man) &
event(C+↑+some+[B], try) &
property(D+↓+the+[]) &
event(E+↑+some+[D], find) &
agent_of(E+↑+some+[D],B+↑+every+[]) &
      (theme_of(E+↑+some+[D],A+↑+some+[D]);
       theme_of(E+↑+some+[D],A+↑+some+[B]);
       theme_of(E+↑+some+[D],A+↑+some+[]))
&
      (state(A+↑+some+[D], unicorn);
       state(A+↑+some+[B], unicorn);
       state(A+↑+some+[], unicorn) ) &
attime(E+↑+some+[D],F) &
agent_of(C+↑+some+[B],B+↑+every+[]) &
theme_of(C+↑+some+[B],D+↑+the+[]) &
attime(C+↑+some+[B],F) &
tense(C+↑+some+[B],pres)
```

It spells out all possible semantic dependencies in the sentence, as represented by the applied logical forms in (242). In Flat Logical Form, distinct readings accumulate as a disjunction of those clauses for which scopal alternatives are available. Thus, in (245) the scopal variation shows up as a disjunction of small clauses `state(A+↑+some+[...], unicorn`), differing from each other only in the fourth term of the variable specification – the dependency marking. The conjunctive frame of Flat Logical Form can be seen as an index – and a meta-representation – of Applied Logical Form.

Clearly, the relationship between PTQ and DELILAH's semantics is not straightforward. In order to deal with the extremely rich semantic profile of natural languages and to make automated interpretation 'inference proof', we need more extended and more versatile formalisms than the one Montague applied. Yet, we consider our way of dealing with automated interpretation as anchoring in Montague's program of computing meanings for forms.

2.3 SCOPE AND SPECIFICATION

2.3.1 Quantifiers

Basically, sentences express predication, modification, mood and quantification. Take the sentence

(246) Enkele klassieke teksten waren niet goed bestudeerd
some classical texts were not well studied
'Some classical texts were not studied well'

The semantic structure may be pictured as follows, with predicates in standard font, mood in bold, modifiers in italics and the quantifier in small capitals

(247) SOME (text, *past*(**not**(*well*(studied))))

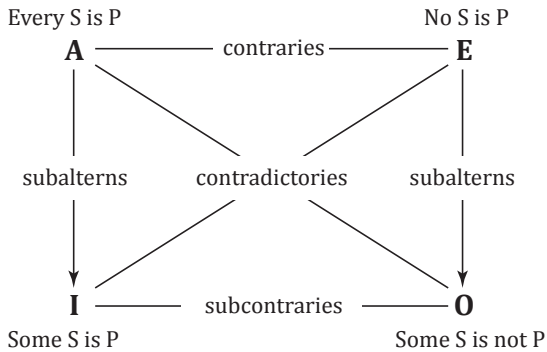
In our approach, modification reduces to predication. By reification of predicates in event structures (see section 2.5), all modification of basic predicates can be cast as properties of the object associated with that predicate: if *walks* is interpreted as an event *e*, the modifier in *walks fast* predicates over that object *e*. Mood certainly is an operator over predication: negation has to be expressed on predicates, and is always introduced as a distinct lexical unit. Quantification, however, is a distinct phenomenon. It relates predicates independently of their nature, and cannot be reduced to predication by enriching that notion. This was Aristotle's insight in the *Prior Analytics*. From these insights, Boethius constructed the square of opposition that governed medieval logic. He defined four quantifiers: A and I (from Lat. *AffI*rmo 'I affirm') and E and O (from Lat. n*Eg*O 'I deny'). A, I and E correspond to *all*, *some* and *no*, respectively. O is best approximated to *not all* or *some...not* – no language

has a lexicalized determiner for this quantifier, and that is not an accident (Jaspers 2005). The four quantifiers were related as follows:

(248) by necessity, the propositions $A\varphi$ and $O\varphi$ differ in truth
by necessity, the propositions $N\varphi$ and $I\varphi$ differ in truth
by necessity, the propositions $A\varphi$ and $N\varphi$ cannot be both true
by necessity, the propositions $O\varphi$ and $I\varphi$ cannot be both false
by necessity, $A\varphi$ entails $I\varphi$
by necessity, $N\varphi$ entails $O\varphi$.

Usually, this family of relations is pictured as the Square of Opposition (source: http://plato.stanford.edu/entries/square/):

(249)

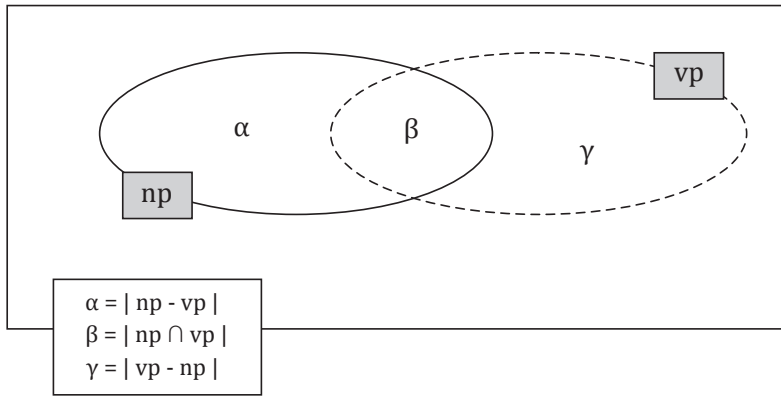


In this diagram, the counterparts to the quantifiers in natural language are added. From these, one can read that the I and the E corners express symmetric quantifications in that the S and P predicates can be inverted *salve veritate.* In the course of the centuries, it has been debated *passim* whether the O corner has existential import and is proper, and, in the same vein, whether S is true of at least one entity for the A corner to be true, or cannot be empty generally. The relation in the square between the two conjectures is revealed in Kneale and Kneale (1962). Suppose there are no Ss. Then I over S and P is false. By contradiction, E is true. By entailment (sub-alternation) O is true. But then, I is not necessarily false, as was assumed. So either S cannot be empty or O has no existential import. Jaspers (2005) argues convincingly that the O corner is linguistically, psychologically and computationally exceptional and must be abandoned. Seuren (2006) reverts to boethian logic in stressing that natural logic for natural language does not deal with empty sets and properties that no object has.

Part of this discussion on the logic of quantification can be projected to the need felt in Generalized Quantifier Theory to restrict the set of possible quan-

tifiers. Particularly illuminating in this respect is the numerical approach of Van Benthem (1986), illustrated by the following figure.

(250)



The open oval indicates the extension of the nominal predicate (S in the Square) in a quantified sentence; the grey one is the extension of the verbal predicate (P in the Square). Van Benthem observes that most quantifiers in natural language can be defined as restrictions on α and β, the cardinality of their respective subsets. That is, these quantifiers are 'about' the *NP* and in that sense *conservative*. For their evaluation, the only domain needed is the *NP*. Moreover, the restrictions are numerical. Below are two alternative definitions of the classical quantifiers.

(251)

A:	$\alpha = 0$	;	$\alpha < 1$
E:	$\beta = 0$	;	$\beta < 1$
I:	$\beta \neq 0$	;	$\alpha > 0$
O:	$\alpha \neq 0$	;	$\beta > 0$

One could choose, in the light of the discussion above, to add numerical statements in order to guarantee existential import.

(252)

A:	$\alpha < 1, \alpha + \beta > 0$
E:	$\beta < 1, \alpha + \beta > 0$
I:	$\beta > 0$
O:	$\alpha > 0, \alpha + \beta > \alpha$

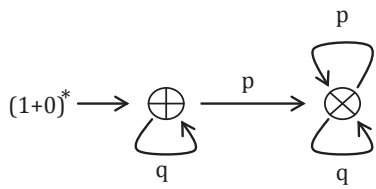
The γ value in (250) is only needed for certain classes of quantifiers, called *non-conservative*, like *only*.

(253)

only:	$\gamma = 0$	$(\beta + \gamma > 0)$
not only:	$\gamma > 0$	$(\beta + \gamma > 0)$

One of the problems with the Square of Opposition and the logics built on it for linguistic analysis is that there are many more quantifiers in natural language than these four, and that not all of them can be reduced to some finite combination of these four. They are called *higher-order quantifiers*, and Van Benthem (1986) proves them to be those quantifiers that cannot be modelled by finite state automata. *Many* and *most*, for example, are among them, but *only* is not. To start with, we give the definitions of finite state automata for the classical quantifiers and for *only.* They are taken to operate on sequences over {1,0}, where, for the classical quantifiers, 1 indicates an object in the β section of (250) and 0 an object in the α section, and for *only* 0 is an object in the γ section. As a matter of fact, the automata for A and for *only* are the same, under this proviso.

(254)



A: + = accepting, x = rejecting,
p = 1, q = 0
E: + = accepting, x = rejecting,
p = 0, q = 1
I: + = rejecting, x = accepting,
p = 1, q = 0
O: + = rejecting, x = accepting,
p = 0, q = 1

It follows that all quantifiers of the type *at least n* for finite integers *n* can be processed by automata with *n*+1 states, and quantifiers of the types *at most n* and *precisely n* by automata with *n*+2 states. The particular status of the O quantifier (*some ... not* or *not all)* can be read from the circumstance that its automaton's initial state rejects 'positive' evidence: the canonical 'models' for the predication – the members of β, represented by 1s in the input of the automaton – are irrelevant to the accepting state.
Clearly, the processing of quantifiers like *most* and *many* needs memory, since they express relations between numbers; in terms of (250):

(255) *most:* $\beta > \alpha$
many: $(\alpha + \beta) / \beta > k$ *or*: $(\gamma + \beta) / \beta > k$, for some *k*, $0 < k < 1$ or $0 \leq k \leq 1$

Most can be processed with simple stack memory or a counter. *Many* and its like require more sophisticated but not very complicated, memory management beyond context-freeness. Notice, though, that *many*, according to (255), has both a conservative and a non-conservative interpretation.

Quantifiers are relations between predicates, and predicates can be complex. In particular the predicates related by quantifier X may give rise to a quanti-

fication Y. Typically, in that case X and Y share at least one of their predicates, while at least one of the four predicates involved is independent. Here are some classical examples, with an informal analysis where *smthing* indicates the locus of dependency.

(256) Every man loves a woman

P^{every}:	*love_someone*	S^{every}:	*man*
P^{a}:	*to_be_loved_by_smthing*	S^{a}:	*woman*

(257) Every farmer who owns a donkey, beats it

P^{every}:	*beat_smthing*	S^{every}:	*farmer_who_owns_smthing*
P^{a}:	*be_beaten_by_smthing*	S^{a}:	*donkey*

(258) Every gentleman who meets her will salute a lady he respects

P^{every}:	*salute_smthing*	S^{every}:	*man_who_meets_smthing*
P^{a}:	*to_be_saluted_by_smthing*	S^{a}:	*lady_who_is_respected_by_smthing*

(257) has been recorded as the donkey-anaphora phenomenon, and (258) as the Bach-Peters paradox (Bach 1970). The occurrence of anaphora here, however, and its syntax should not obscure the underlying phenomenon of semantic dependency. When quantifiers share a predicate, their valuation gets intertwined. For some quantifiers, this intertwining leads to dependency: the relation they express is governed by an instantiation controlled by the other quantifier. Some quantifiers, typically I and its derivatives, are sensitive to this government: their valuation may vary with the valuation of the other quantifier. Other quantifiers, like A and O, are unaffected. In (256) we have the choice to have the evaluation of I vary or not vary with the evaluation of A; the sentence is ambiguous in this respect. In the first case, when the evaluation of I varies with the instantiation controlled by A, we say that I is *in the scope* of A. It is not very useful – though common practice – to say that in the other case, A is in the scope of I, since the evaluation of A will never vary with the instantiation controlled by I. In general, we can say that only those quantifiers that have a definition in terms of (250), like

(259) Q: $\beta \geq n$

for some independent integer $n > 0$ are sensitive to scope. That is, quantificational ambiguity is relevant neither in sentences with only A, E or O quantifiers nor in any of the following complex quantifications.

(260) Every man loves *at most three / many / most / the / only* women

The source of the sensitivity of I quantifiers must lie in the fact that they may be in competition with other quantifiers as for *aboutness* of the sentence. By

definition, they introduce entities that are both S and P. No other type of quantifier does. The sentence is *about* these entities, by definition. The *aboutness* claim of another quantifier, however, may interfere with I's one, if that quantifier introduces a different focus – as A does, for example, by not introducing entities in the intersection of the predicates but requiring one difference between them, *NP-VP*, to be zero. In that situation, the *aboutness* claims have to be ordered with respect to each other. Thus, we would expect that *only* and *not all* may also have an *aboutness* conflict with I. And they do: the next sentences are as ambiguous as (256).

(261) Only men loved a favourite actress of Hitchcock's
Not every man loved a favourite actress of Hitchcock's

No conflict arises when two Is meet. Since they share at least one predicate and target at the intersection of the predicates only, they focus on the same subset of the universe. In that case, both I quantifiers are *dynamic*, in the sense that they can both occur referentially and license anaphora outside their syntactic domain – the Dekker (1993) litmus test to tell dynamic from static quantifiers:

(262) Some_i men read a_j book by Chomsky. They_i had bought it_j for 10 bucks.
(263) All_i men read a_j book by Chomsky. * They_i had bought it_j for 10 bucks.

The determination of semantic dependencies between quantifiers must be a target of linguistic analysis. In DELILAH, it is taken care of by a complex post-derivational procedure dubbed *apply_store* that inspects Stored Logical form and – among other tasks – determines for each quantifier whether or not it might be in the scope of one or more others. It spells out possible ambiguity, and thus maps the Stored Logical Form on a family of disambiguated readings. In doing so, it takes into account syntactic as well as semantic information accumulated in the derivation. It will not mechanically produce representations of all possible scopal variation; it restricts its output to semantically distinctive readings. In Flat Logical Form, these readings are assembled as local disjunctions of the relevant predicates.

2.3.2 The structures of quantification

In all languages, proper names and quantificational *NPs* share most paradigms. In particular, argument positions at predicates are open to both. Yet, they are semantically different. Names denote rigidly, as Kripke (1972)

argues, but quantifiers denote contingently, varying with situations and the extension of predicates. It took linguistics some millennia to reconcile the distribution and the typing of the different sorts of nominal constituents, and it took a logician to find the key. Montague (1972) typed both proper names and quantificational *NPs* as characteristic functions over predicates, and thus puts them in the same semantic league as higher order sets, in compliance with their equal distribution. The general format of a one-place predication in natural language, then, is that a sentence $[_S$ NP VP] is to be interpreted as the proposition that the meaning of the NP applies to the meaning of the VP, in this case: ⟦VP⟧ ∈ ⟦NP⟧. Barwise and Cooper (1981) argued that the rich algebraic structure of these characteristic sets can, and indeed should, be exploited for linguistic analysis. They coined the term *Generalized Quantifier* for the algebraic objects that could be identified in the realm of the power-set of the universe (the universal predicate) as interpretations for natural-language NPs. The general definition boils down to this:

(264) *Generalized quantifier*
A set of subsets Q of the universal predicate U is a generalized quantifier *iff* there is a subset A of U and a well-defined relation R, such that X is in Q iff R(A,X) applies.

That is: a generalized quantifier requires a property to *live on* and a relation to generate the set from this property. For normal quantificational *NPs*, the property-to-live-on comes with the nominal predicate. In the case of proper names, it is the singleton of being a particular individual, like {j}. This property limits the *aboutness* of the quantifier. The class of relations is given with the algebra of sets < ∪, ∩, ⊆, − > and numerical conditions on sets. To find the restrictions limiting this class of relations is a main target of modern semantics. The classical quantifiers A, E, I and O and the proper name Socrates are defined below as generalized quantifiers.

(265) ⟦every N⟧ = { X ⊆ U | ⟦N⟧ ⊆ X }
⟦some N⟧ = { X ⊆ U | ⟦N⟧ ∩ X ≠ ∅ }
⟦no N⟧ = { X ⊆ U | ⟦N⟧ ∩ X = ∅ }
⟦not every N⟧ = { X ⊆ U | ⟦N⟧ ⊆ X }
⟦Socrates⟧ = { X ⊆ U | {s} ⊆ X }

The structures that these quantifiers establish are far from random. Barwise and Cooper (1981) and Zwarts (1982, 1986) identify numerous typical algebraic objects like filters and ideals among them, and relate these structures to distributional characteristics of the corresponding *NPs*. This, then, is the real

revolution of generalized quantifiers: language users apparently recognize the algebraic characteristics of the quantifiers when constructing or parsing sentences. In this vein, it is not remarkable that Emon Bach is reported to have said that generalized quantification is the major development in linguistics in the past millennia.
Another very valuable feature of generalized quantifiers is that literally any quantifier can be captured in these terms. Here are a few more.

(266) $[\![\text{only N}]\!] = \{ X \subseteq U \mid X \subseteq [\![N]\!] \}$
$[\![\text{most N}]\!] = \{ X \subseteq U \mid |[\![N]\!] \cap X| > |[\![N]\!] - X| \}$
$[\![\text{many N}]\!] = \{ X \subseteq U \mid |[\![N]\!] \cap X| > k,$ for some k depending on $|[\![N]\!]|$ or $|X| \}$
$[\![\text{at most n N}]\!] = \{ X \subseteq U \mid |[\![N]\!] \cap X| < n \}$

Quantifiers are of type <<et>t> in a (t, e)-based type-logic. As extensional predicates (like nouns) are of type <et>, determiners are of type <<et><<et>t>> or, equivalently, of type <<<et><et>>t>. Although Montague (1972) characterized determiners *syncategorematically*, consequent typing identifies the relevant determiners as relations between sets. The definitions are fairly straightforward, given (265).

(267) $[\![\text{every}]\!] = \{ \langle X,Y \rangle \mid X \subseteq Y \}$
$[\![\text{some}]\!] = \{ \langle X,Y \rangle \mid Y \cap X \neq \emptyset \}$
$[\![\text{no}]\!] = \{ \langle X,Y \rangle \mid Y \cap X = \emptyset \}$
$[\![\text{not every}]\!] = \{ \langle X,Y \rangle \mid X \not\subseteq Y \}$
$[\![\text{many}]\!] = \{ \langle X,Y \rangle \mid |Y \cap X| > k \}$

This amounts to an analysis of determiners as relations between predicates, almost in the medieval sense, but now with access to all the concepts coming with modern algebra, as practiced in Zwarts (1983). Among the fruits of this approach, we have the insight that certain determiners cannot occur in a language but on the penalty of violating contingency. For example, if a language were to have a determiner Q to validate the syllogism in (268), this determiner would only produce trivially true sentences.

(268) Q *A B*, Q *A C* ⊨ Q *B C*

Thus, Q *X Y* would be true for any *X* and *Y*. As languages may be assumed not to lexicalize triviality, such a quantifier (dubbed 'Euclidean') cannot exist. For details of the proof, see Zwarts (1983).

In the same relational vein, determiners can be projected on a Pascal triangle of pairs of numbers <α, β> following (250), with α standing for the cardinality of NP-VP and β standing for the cardinality of NP ∩ VP,

(269)

<0,0>
<1,0> <0,1>
<2,0> <1,1> <0,2>
<3,0> <2,1> <1,2> <0,3>
....

Clearly then, these pairs represent the way a determiner may divide an NP with cardinality α+β. For example, the quantifier A selects the rightmost edge of the triangle, and E the leftmost. This way, one can construct a geometry of determiners and try to find out which planes do, or do not, qualify as representing (lexical) natural-language determiners (cf. Van Benthem 1984).

2.3.3 Compositionality and underspecification

Generalized quantification opens an algebraic window on semantic structures. It assumes – or rather: hypothesizes – that language entertains higher-order objects, the structure of which is understood and/or computed in action. It adds structure to meaning – it lends a face to meaning. For the theory of grammar, it poses the question to what extent Montague's idea of synchronizing syntactic and semantic structure can or must be preserved. This strict concept of so-called *compositionality* is expressed in the categorial grammar emanating from Lambek (1958), by identifying syntactic and semantic types, and by correlating logic and derivation. This strategy has been explored in-depth in Moortgat (1988), Hendriks (1993), Morrill (1994) and more work by these and other authors. The bottom line in this approach is that interpretation and derivation are parallel, and that semantic variation comes with derivational variation. In particular, variation in semantic dependency – like scope – is (partially) accounted for by syntactic processes. In this sense, compositional categorial grammar is montegovian by nature.

Delilah does not observe strict compositionality. In particular, scopal relationships – semantic dependencies between variable binding or other operators – are not computed during the derivation, but afterwards. The derivation itself delivers an underspecified storage of functional terms, assembled by sheer unification. This structure is subject to a kind of spell-out algorithm,

establishing full scopal relations in a fine-grained propositional frame. The ideology and benefits of this post-derivational after-burn will be discussed in section 2.6. What matters here is the proper balance between compositionality and semantic specificity.

It is not a secret that syntax – or form in general – underdetermines the subtleties of interpretation. It is simply not the case that scopal ambiguities are completely resolved by syntactic or prosodic marking, or that intensional embedding always comes with specific morphology. The reverse is true, too. If you have an interpretation for a sentence, not every formal aspect of a sentence carrying that interpretation can be read from it. This is established in section 2.7 on generation. In strictly compositional approaches to parsing, derivational flexibility plays the role of marking by form. If a sentence S has several meanings P, then for each P there is a different way to construe S. These different construals may even yield the same configuration: it is the procedure, the *orde*r of combining terms that makes the difference. There is a serious problem with this approach. Although structural ambiguity is reflected in variation of interpretation, no common representational layer for this variation can be computed: nothing is *the* (core of the) interpretation(s) of a sentence that is fully interpreted by derivation.
Of course, for interpretation we do not need a semantic object that is shared by the readings; the relationship between readings is defined logically and not syntactically. For processing purposes, however, a computable object capturing the core of a family of readings might be very welcome. This idea has become the heart of the approach to the computation of semantics that is simply dubbed *underspecification.* It rejects the idea that every aspect of sentential semantics must be computed by derivation. Bunt (2008) identifies lexical ambiguity, syntactic ambiguity, structural semantic ambiguity, imprecision and missing information as the main motives for underspecification. At the same time, he observes that the different techniques for underspecification also fulfill different needs, and that no existing technique meets all semantic requirements – a conclusion formalized in Ebert (2005) as *expressive incompleteness*. The conclusion affects approaches like Cooper storage (Cooper 1983, Frank and Reyle 1994), Hole Semantics (Bos 2002), Minimal recursion Semantics (Copestake *et al*. 2005) and Glue Semantics (Dalrymple *et al.* 1995, Crouch and Van Genabith 1999).

With Delilah we pretend to live in two worlds: the world of underspecification, and the world of full logic. Underspecification is necessary when meaning is to be computed by derivation and no information on the deriva-

tion is retained elsewhere. Full logic requires distance from the derivational processes, as many of them do not convey any information relevant to full logic. DELILAH stores all relevant derivational and structural information in an accessible manner, in the very same data structures. This information is accessible afterwards to provide full logic out of underspecification. The underspecified semantic object, generalizing over a family of readings, is just one of the yields of the derivation. *Information management* provides the algorithm spelling out that family with all the pieces of information that it needs and the availability of which would interfere with strict compositional and derivational interpretation. Under this strategy, fully-specified semantics like (242) or (245) can be derived from a well-defined derivational product, Stored Logical Form. A comparable strategy is proposed in Koller and Thater (2006). Consequently, the family of analyses ALF and FLF is protected against spurious redundancy while it is being constructed by incorporating grammatical knowledge.

Below, an outline is presented of the algorithm that takes care of the translation from SLF into full specification.

(270) APPLY(f, g) ⇒ f(g) iff *g* is of type α and *f* of type <α, β>
APPLY(f, g) ⇒ g(f) iff *f* is of type α and *g* of type <α, β>
undefined otherwise

(271) ORDEN(ListOfTerms, Template) ⇒ the list of terms Ordered such that for every a and b in ListOfTerms, a and b occur in Ordered and a precedes b *iff* according to Template, b is scope-sensitive and a is not.

(272) APPLYSTORE(Store, Body, Template) ⇒ APP(LIED)LOGI(CAL)FORM
(1) APPLYSTORE([], Body, Template) ⇒ Body;
(2) *otherwise* Store = [First|Rest] *and*
(2.1) *if* First = <S, B> *then*
(2.1.1) *if* B is not an island according to Template *then*
(2.1.1.1) APPLY(B, Body) ⇒ $Body^B$ *and*
(2.1.1.2) APPLYSTORE(SURest, $Body^B$, T) ⇒ APPLOGFORM *and*
(2.1.1.3) APPLYSTORE(S, B, Template) ⇒ B^S *and*
(2.1.1.4) APPLYSTORE([B^S|Rest], Body) ⇒ APPLOGFORM
(2.1.2) *otherwise*
(2.1.2.1) APPLYSTORE(S, B, Template) ⇒ B^S *and*
(2.1.2.2) APPLYSTORE([B^S|Rest], Body, Template) ⇒ APPLOGFORM
(2.2) *otherwise*
(2.2.1) *if* First is scope-sensitive according to Template *then*
(2.2.1.1) *if* Rest contains structures <S', B'> *then*
(2.2.1.1.1) APPLYSTORE(Rest, Body, Template) ⇒
<NStore, NBody> *and*

(2.2.1.1.2) APPLYSTORE(NStore∪[First], NBody, Template) ⇒ APPLOGFORM
(2.2.1.2) *otherwise*
(2.2.1.2.1) ORDEN(Rest∪[First], Template) ⇒ OrderedStore *and*
(2.2.1.2.2) APPLYSTORE(OrderedStore, Body, Template) ⇒ APPLOGFORM
(2.2.2) *otherwise*
(2.2.2.1) APPLY(First, Body) ⇒ B^{First} *and*
(2.2.2.2) APPLYSTORE(Rest, $Body^{First}$, Template) ⇒ APPLOGFORM

Below is an example of the algorithm's effect. Let (273) stand for the derivation of a simple transitive sentence, which marks *Det2* as scope-sensitive in a non-island. Let (274) be the Stored Logical Form produced by the derivation, with the lambda terms related to the constituents italicized in a <Store, Body> format. The application of algorithm (272) to this storage yields, in some logical formalism, the family of readings in (275).

(273) [[Det1 NP1] [Vfin [Det2 NP2]]]
(274) <<*NP1*, *Det1*> <[*NP2*] *Det2*><[ ],*V*>> *FIN*>
(275) *Det1(NP1)(Det2(NP2)(FIN(V)))*
Det2(NP2)(Det1(NP1)(FIN(V)))

In this example, *FIN* represents the term associated to finiteness, and V holds the event structure. As such, *V* contains a quantifier but the derivational template excludes this quantifier from taking scope.
In general, two scope-sensitive operators are unordered with respect to each other. So is each pair of non-scopers. The cardinality of the readings created by (272), therefore, is less than the faculty of the number of operators. This can be deduced as follows. In each semantic island (see Szabolcsi and Den Dikken 1999, Honcoop 1998) the derivation *de facto* distinguishes between scope-sensitive and scoping, independent, non-sensitive operators. A reading is any particular configuration of the members of these two classes. Let a domain contain three scope-sensitive operators {x, y, z} and two independent operators {a, b}. In general, indefinite operators are scope-sensitive, or dynamic, while definite operators are scope-insensitive, or static. The order of the members of the second class is irrelevant. A scope-sensitive operator is in the scope of an independent operator if it occurs to the right of it. The algorithm (272) ends up with the following set of readings

(276)

x y z a b	x y a z b	x y a b z	y z a x b	y z a b x
x z a y b	x z a b y	x a y z b	x a y b z	x a z b y
x a b y z	y a x z b	y a x b z	y a z b x	y a b x z
z a x y b	z a x b y	z a y b x	z a b x y	a x y z b
a x y b z	a x z b y	a y z b x	a x b y z	a y b x z
a z b x y	a b x y z			

If *s* is the number of scope-sensitive operators and *i* the number of independent scope-taking operators, the number of different readings is s^{i+1}. In this case there are 27 readings. Naive permutation would yield 5! = 120 readings. In general, we compare *n!* to k^{n-k+1} for $k < n$.

In Flat Logical Form (see (245)) different readings are clustered into one single formula of local disjunctions of small clauses. Thus, the backbone of the final representation for the sentence with the operators {x, y, z} and {a, b} will be a conjunction of small clauses. The different readings occur in the following way: every small clause in which a scope-sensitive operator *x* binds is replaced by a disjunction of small clauses each of which represents exactly one scopal dependency under which *x* can occur. In a scheme: instead of having a conjunction of small clauses (277), we have a conjunction of disjunctions of small clauses – the left arrow marks semantic dependency:

(277) x *and* y *and* z

(278) (x *or* x↤a *or* x↤a,b)
and
(y *or* y↤a *or* y↤a,b)
and
(z *or* z↤a *or* z↤a,b)

This disjunction is all that remains of the 27 readings in Delilah's semantic set-up. The whole process from Stored Logical Form to Flat Logical Form is a transition from under-specification to full specification, steered by derivational information, amounting to a kind of *logarithmic* interpretation: it decreases the order of semantic multiplicity. This implements the idea that one single semantic object should be assigned to every sentence that is lexically disambiguated, at every level of interpretation – its interpretation. Scopal ambiguity is brought back to sentence structure, and that is compositionality.

In section 2.7, finally, we will consider the use of Flat Logical Form as a program for generation. Although (278) is a conjunction of disjunctions, one can still read it as an instruction to a generator to produce a family of sentences.

2.4 INTENSIONALITY AND SEMANTIC DEPENDENCY

2.4.1 Intensionality

Phrases in a sentence may not directly contribute to determining its reference. They are not meant to refer directly to semantic objects that can be identified in the situation or state-of-affairs the sentence aims to *be about.* The crucial configuration is an embedding clause. If a sentence φ properly contains a sentence ψ, they cannot refer to the same semantic object. If φ refers to the situation-at-hand, ψ can't. In that case, ψ refers either to a sub-situation of the coordinates proposed by the main sentence, or it introduces new interpretative coordinates or indices. By now, we are referring to the latter situation as intensional embedding. It is typical for sentences or situation-type expressions because we assume that interpretation of a complex sentence amounts to its interpretation in one particular situation – that is what model theory tells us. Sentences can be about many objects, relations and moods, but about only one situation. So, the initial definition of intensionality or, better, intensional embedding is like this.

(279) *Intensional embedding*
where $[\![\psi]\!]^W$ is the interpretation of ψ with respect to situation W

A proposition φ is intensionally embedded in a context $[_s$ X φ Y] *iff*
$[\![[_s$ X φ Y] $]\!]^W$ is not a function of $[\![\varphi]\!]^W$.
In that case, φ is an intensional domain.

(280) *Extensional embedding*
A proposition φ is extensionally embedded in a context $[_s$ X φ Y] *iff*
$[\![[_s$ X φ Y] $]\!]^W = f([\![\varphi]\!]^W)$ for some *f* associated with $[_s$ X _ Y].

As was said before, Montague (1972) considers *every* embedding to be intensional, unless explicitly denunciated as such post-derivationally. This view is the same as the one expressed here since Montague's lambda terms, which were lifted to intensional functions under embedding, are all of type <α,t>, mapping onto the canonical type for propositions.

For DELILAH, we assume that every sentential embedding amounts to the creation of an explicit intensional domain. This domain, however, is not reflected at the level of types, as in Montague (1972), but by semantic operators that may or may not lead other variables into semantic dependency. Embedding

comes with the introduction of definite quantifiers over restrictions like *proposition* or *property* and scoping over or below the content of the embedded sentential or infinitival construction. The relevant distinction between properties and propositions is made within the class of infinitival complements, following Cremers (1983): all finite sentences map onto propositions, and some but not all infinitival complements do. For reasons explained there, the internal semantic construal of a property is much more restricted than the construal of a proposition, and verbs select for that construal: *proberen* 'to try' selects a property, *beloven* 'to promise' selects a proposition.

This strategy leads to the following Applied and Flat Logical Forms; in FLF, the arguments of predicates occurring in an intensional domain are marked for dependency on the proposition or property that establishes the domain and scopes over them at ALF.

(281) Jan probeert te slapen
'Jan tries to sleep'

NLF

```
exists(A) & quant(A,henk).[quant(B,some). event(B, try) &
quant(C,the).[property(C).[quant(D,some).[event(D, work) &
agent_of(D,A) & attime(D,E)]] & agent_of(B,A) & theme_of(B,C) &
attime(B,E) & tense(B,pres)]]]
```

FLF

```
exists(A+↑+henk+[]) &
event(B+↑+some+[], try) &
property(C+↓+the+[]).[
work(D+↑+some+[C]) &
event(D+↑+some+[C], work) &
agent_of(D+↑+some+[C],A+↑+henk+[]) &
attime(D+↑+some+[C],E)] &
agent_of(B+↑+some+[],A+↑+henk+[]) &
theme_of(B+↑+some+[],C+↑+the+[]) &
attime(B+↑+some+[],F) &
tense(B+↑+some+[],pres)
```

(282) Henk zei elke jongen te slaan
Henk said every boy to beat
'Henk said that he beats every boy'

ALF

```
exists(A) & quant(A,henk).[quant(B,some).[event(B, promise) &
quant(C,the).[proposition(C).[quant(D,every).[man(D) & young(D) ->
quant(E,some).[beat(E) & event(E) & agent_of(E,A) & theme_of(E,D) &
attime(E,F)]] & agent_of(B,A) & theme_of(B,C) & attime(B,F) &
tense(B,past)]]]
```

```
exists(A) & quant(A,henk).[quant(B,some).[event(B, promise) &
quant(D,every).[man(D) & young(D) -> quant(C,the).[proposition(C).[
quant(E,some).[beat(E) & event(E) & agent_of(E,A) & theme_of(E,D) &
attime(E,F)] & agent_of(B,A) & theme_of(B,C) & attime(B,F) &
tense(B,past)]]]]
```

FLF

```
exists(A+↑+henk+[]) &
event(B+↑+some+[], say) &
proposition(C+↓+the+[]).[
      (man(D+↓+every+[C]); man(D+↓+every+[] )) &
      (young(D+↓+every+[C]; young(D+↓+every+[])) &
event(E+↑+some+[C,D], beat) &
agent_of(E+↑+some+[C,D],A+↑+henk+[]) &
      (theme_of(E+↑+some+[C,D],D+↑+every+[C]);
       theme_of(E+↑+some+[C,D],D+↑+every+[]))&
attime(E+↑+some+[C,D],F)] &
agent_of(B+↑+some+[],A+↑+henk+[]) &
theme_of(B+↑+some+[],C+↑+the+[]) &
attime(B+↑+some+[],G) &
tense(B+↑+some+[],past)
```

The logical forms of (282) reflect the classical ambiguity between the *de dicto* and the *de re* readings of *(every) boy* – the genuine target of research into intensionality. There are two important questions to consider, though. First, it is not clear whether intensionality affects the interpretation of the embedded quantifier as a whole, or just of its restrictions, or both. Second, one can wonder whether the *de dicto / de re* ambiguity also affects the event of *beat*.
As for the first question: does it make sense to assume that (282) is three-way ambiguous?

(283) ... *say*...[... for every *boy(x)* ... beat(...x)
for every *boy(x)* ... *say* ...[... beat(...x)
for every x ... *say* ...[... if *boy(x)* then beat(...x)

As far as we can see, no interesting truth-functional distinction can be constructed between the second and the third readings. Yet, it is certainly reasonable to consider the 'responsibility' for the qualification *boy*; it is with the speaker or the utterer *(de re)* in the second and with the alleged 'sayer' in the third. That distinction becomes quite clear when a less neutral qualification for the restrictor is chosen:

(284) ... *say*...[... for every *traitor(x)* ... beat(...x)
for every *traitor(x)* ... *say* ...[... beat(...x)
for every x ... *say* ...[... if *traitor(x)* then beat(...x)

Now, it certainly matters whether one is quoting or summarizing the proposition made by the sayer. The second, but not the third, reading entails that those who are said to be beaten are traitors. The quantifier, however, is irrelevant for this opposition, in this case. If the quantifier were existential or indefinite, however, truth-functional considerations would make a difference independently of your modal model: *there is someone who is said to be a traitor and who is said to have been beaten* vs. *there is a traitor said to have been beaten*. In our view, the first reading amounts to the referential reading of indefinites pointed at by Fodor and Sag (1982:380): '... a referential indefinite can be used for the purpose of making an assertion *about* an individual, even though the individual in question is not identified by the speaker'.
Ruys (2006) proposes to deal with certain scope phenomena around indefinites precisely by distracting the quantificational aspect from the referential aspect. He assumes, however, an analysis of indefinites in terms of existentially quantified *choice functions* – functions from a non-empty set to members of that set – of type <<et>e>. One can reconstruct the interpretation scheme of his example (64) *Every country's security will be threatened if some building is attacked by terrorists* as follows.

(285) for all x, *if* Country(x*)*
then there is a choice function *f* and ThreatenedIfAttacked (x, *f*(Q))
for all x, there is a choice function *f* such that *if* Country(x)
then ThreatenedIfAttacked(x, *f*(Q))
there is a choice function *f* and for all x, *if* Country(x)
then ThreatenedIfAttacked(x, *f*(Q))

Clearly, the quantification over the choice function is made dependent on the universal quantifier in the first two, but not in the last reading. The second one is relevant here: the existential quantifier is in the scope of the universal quantifier but not in its restriction. Quantified choice functions, however, do not come with a labour division between restriction and nuclear scope, which is characteristic for natural-language quantifiers (see section 2.3.1). Moreover, choice functions themselves are hardly referential – the axiom of choice is non-constructive – and thus not subject to *de dicto/de re* alternations. A choice function is a way to identify members of a set. By the axiom of choice, a choice function exists for every set (Jech 1977; cf. section 2.4.2). The function is not subject to interference with other quantifiers or operators. The introduction of entities by existential closure of choice functions imposes the referential dependency on the selection of entities, but is not scoped itself. In this respect, we do not follow Ruys (2006), Winter (2001) and Reinhart (1997). Consequently, as for (285), we take the third reading to be the crucial one: the selec-

tion and identification of set members is postulated by existential closure, but the ordering of the relevant set is parameterized by other terms.
Although it is not impossible to account for three-way ambiguity in Applied Logical Form, Flat Logical Form does not qualify for this representation: all quantification is compiled into the predications. We must conclude, therefore, that this representational strategy may be defective in this sense. But this imperfection hardly disqualifies the logical form with respect to other approaches.

The second question raised by (282) was whether this intensional ambiguity would also arise in the case of the event *beat*. This question seems to be provoked by our choice to quantify over events, but even if in non-event semantics, the question who is responsible for the embedded predicate is relevant. For independent reasons to be addressed in section 2.5, events do not participate in scopal dependencies – reference to events is not influenced by their embedding. As a matter of fact, we take events to be introduced by the ultimate choice functions: assuming – philosophically rather than linguistically – that on every index of interpretation or in every situation there is always at least one event, labelling the event chosen is always possible and is not an act of reference as such. The event is identified by its roles and their performers, rather than by labelling it. Thus, in order to lift the event out of the intensional domain and still have it participate in meaningful scopal dependencies, it would be necessary to bring its full class of related propositions to the main clause level.

(286) for some event x of sort A there is an agent z and a patient y such that … INTENSIONAL OPERATOR …[…x …y … z

This would render the proposition, the theme of *say*, for example, vacuous as to its content: at best, the remaining proposition could be interpreted repeating the lifted constituent, namely that there is an event so and so. To put it boldly: it amounts to lifting the proposition out of itself.
It is difficult to see the gain here. Therefore, we assume that events are dwelling inside their intensional domain, being semantically dependent by definition. In general, we will assume that events have narrow scope in the following sense: the interpretation of no variable in a sentence ever depends on the valuation of the event term. Or, in different phrasing: the verb may be the queen of syntax, but she surely is the beggar of reference.
The domains of intensionality are introduced as a lexical feature of the embedding. Verbs taking infinitival complements – but not the standard auxiliaries – subsume the semantics of that complement under a *property* or a *proposition.* Overt complementizers come with a propositional domain. An

infinite verbal form is not intensional by definition, but a sentence with an overt complementizer is.
Intensional dependency, then, is represented by semantic dependence. If φ is to be interpreted in an intensional domain, it is marked for being dependent on the valuation of a variable of type *s* – the intensional basic type in Montague's logic. φ can be inferred only with respect to this valuation. And this is the main goal of an intensional logic: to shield certain semantic objects from unconditioned inference and from interpretation at the default indices.

2.4.2 Skolemization of dependent events

Indefinites may be referential or quantificational, specific or non-specific, with wide or narrow scope. Their proper interpretation has received strong focus in modern semantic reflection. In order to deal with their magic, Reinhart (1997) proposes to interpret them partially by choice functions, for 'assigning wide scope to existentials without moving their restriction', *i.e.* in order to avoid manipulations on logical form by quantifier raising and the like. She targets on those indefinites that come with bare numerals and that were classified by Kamp and Reyle (1993) as introducing a discourse referent.
Events are candidates for introduction by choice functions because they are indefinite – their existence can be denied consistently – and because they introduce discourse referents – they can easily be referred to by definite anaphora. Moreover, the phrases restricting the events – all sorts of predicates – hardly qualify for quantificational manipulation. It seems as if choice functions are the only but also the optimal option to handle the semantic behaviour of events. To see why, we have to project the particular nature of quantification over events on the concept of choice function.
Choice functions are assumed to exist as executors of the *axiom of choice*. In set theory, this axiom states that in every non-empty set a member can be identified (chosen), even if that set is infinite and no selection rule is given. The axiom amounts to the claim that every set is well ordered so that its members can be distinguished from each other. This is not trivial for infinite sets or for sets the construction of which is unknown. As a matter of fact, the axiom is independent of set theory but has by now become accepted as an important non-constructive but intuitively valid tool. The axiom was introduced by Zermelo in 1904 and is defined as follows in Jech (1977):

(287) For every family $\mathcal{F}$ of nonempty-sets, there exists a function *f* such that $f(S) \in S$ for each set S in $\mathcal{F}$.

The axiom reduces the identification of members of a set to the function by which they are selected. More precisely, the identification of a member of a set is made relative to the identification of a member of another set. This means that we can identify the member of a set the ordering of which is not evident in the same way as we identify the member of a set the ordering of which is less problematic and in which identification of individual members is less problematic: we can always pick the first, the second or the *n*-th member by refering to a set with an established well-ordering.

As Winter (2001) notes, choice functions can be generalized to Skolem functions: functions that identify an object with reference to independently identified other objects. Skolem functions are applied in the valuation of sentences in a predicate calculus, *e.g.* by the programming language Prolog (cf. Clocksin and Mellish 1984). A Skolem function f_φ for a formula $\varphi(v_0, ...,v_n)$ is defined by

(288) $\forall v_0 ... \forall v_{n-1}[\exists v_n.\varphi(v_0, ...v_{n-1}, v_n) \rightarrow \varphi(v_0, ...v_{n-1}, f_\varphi(v_0, ...,v_{n-1}))]$ (*cf.* Morley 1977)

The function replaces the existential quantifier and its (dependent) variable v_n by a function of type $<<e^n t>e>$ that is essentially a choice function over the set $[\![\lambda v_n.\ \varphi(v_0, ...v_{n-1}, v_n)]\!]$ with respect to some assignment of values to each v_i, $0 \leq i \leq n\text{-}1$. The Skolem function identifies a dependent object by using parameters from other sets (valuations of variables into those sets) and by getting rid of the related (indefinite) quantifier. Thus, it turns the object scope-independent, frees it from quantification and makes it addressable; and it does so by leaning on the axiom of choice. Basically, the Skolem function is the procedural counterpart of scopal dependency.

We need Skolem functions to deal with events, for at least three reasons. Firstly, events are existential and semantically dependent, intuitively. The use of a predicate asserts the existence of an event or state but the event is identified by refering to the arguments and adjuncts, like the specifications of time and place. From the point of view of modelling, it does not make sense to claim that there is an event of a certain sort without refering to its thematic, temporal and, possibly, spatial correlates (cf. Verkuyl 1993, ch. 13). There are no events outside the gates of Eden. Consequently, Skolemization of the interpretation of events reflects their real but secondary referentiality.

Secondly, the algebraic structure of the set of events and states is such that identification of events by simple quantification is not enough. Events are mass-like, rather than discrete and countable (cf. Bach 1986). We have to guarantee the selection and identification of an event and its labelling with predicates by more fundamental means, like contextual fixation.

Thirdly, the algebra of events is such that manipulating scope does not buy us much – except in the case of truly collective events or states like the *kolchoz collectivity* or *the angles of a triangle are 180 degrees* – the predicative and semantic structure of which is difficult anyway. Except in the *kolchoz* cases, one event or state for all can always be split into sub-events or sub-states per individual, just as events per individual easily aggregate to just one; since events combine in many ways, their scope is referentially not distinctive.

The three reasons amount to just one from an ontological point of view: events are entities that differ essentially – *i.e.* algebraically – from other entities living in the realm of natural-language semantics, and therefore deserve some logical respect.

In the following excursions, we will assume that every predicate comes with a Skolem function, identifying a state or event by refering to its parameters, without assigning scope to its value. Logically, however, the interpretation of predicates is supposed to be existentially quantified, though the quantifier is redundant in the valuation. In the sections to come, events and states are marked as such. They do not enter into exciting interactions with other operators, because their valuation is handled by Skolem functions. In Flat Logical Form, however, quantifiers do not return as scopal objects anyway.

2.5 EVENTS AND STATES: REIFICATION OF PREDICATION

2.5.1 Reference to events

In natural language, predicates come in some variety: nouns, verbs, adjectives, adverbs, prepositions may all introduce properties of entities introduced by other phrases in a sentence. For reasons of inference, all these predications have to be made explicit. *John put the wood in the garden* entails, among others, that after a certain moment the wood is in the garden, and this predication must be made explicit in the sentence's representation in order to make that inference viable. In the same vein, something is predicated to be wood and to have been put in the garden. Is there also something of which the sentence predicates that it is putting-wood-in-the-garden, or putting-something-by-John, or putting-wood-in-the-garden-by-John? Yes, there is. In both sequences of (289) the italicized phrase in the second sentence refers

to a "something" that is not addressed or introduced by any phrase in the sentence other than the one headed by *put.*

(289) John put the wood in the garden. But Mary did not approve of *it*.
John put the wood in the garden. He did not do *it* alone, though.
John put the wood in the garden. He was all alone*, then*.

Davidson (1967) was the first to propagate that the predicative "something" of action verbs should be addressable in natural-language semantics. The strategy of assigning a referential backbone to real predicates is named after him: '(neo-)Davidsonian'. Here is an example from Reckman (2009).

(290) I flew my spaceship to the Morning star
∃x. Flew(i, my_spaceship, x) & to(morningstar, x)

Here the predicate *flew* is expressed as a three-place relation, one of the arguments of which is a bound variable that occurs as an argument in a typical adverbial modification of the predicate.

At least three questions immediately arise: (1) what is the nature of the predicative entity? (2) should or can *all* natural-language predicates be *reified* this way and (3) how does it get quantified over?

The first question can be answered as: that is a matter of philosophical taste. If you prefer a sharp ontology, you can introduce a particular type or sort for the predicative entity: apart from entities and/or propositions, there are properties representing a 'basic' type, as has been proposed by Turner (1989). Alternatively, you have the predicative entity's nature defined by some predicate, e.g. the predicate itself. This is more or less the strategy undertaken in Delilah. But there we introduce reified predicates by a metapredicate like *event*, relating the predicative thing to a name and introducing explicit role labels for the predicate's arguments. Delilah's version of (290) looks like this:

(291) I flew my spaceship to the Morning star
∃x. Event(x, flew) & Agent_of(x, I) & Theme_of(x, My_spaceship) &
To(x, MorningStar)

The main advantage of this way of representing *flew* is that it makes explicit, and thus inferable, that I was actively engaged in something, and that my spaceship was in a different way engaged in the same thing, no matter what we call it. In other words, the sentence establishes a semantic term in which

the phrases act as values to attributes. All phrases are born equal, so to speak, but it is the main verb here that comes with the thematic frame: it still is the sentence's queen. The headness of the verbal predicate can only be read from the derivation, though. In the lexicon, the verb is endowed with a thematic frame that unfolds in the transition from Stored to Applied and Flat Logical Form. For example, a lexical template for the finite verb *slaapt* 'sleeps' at SLF not only has a variable for the subject term in store, but also a genuine quantifier introducing an event; abstracting away from bookkeeping, the SLF in the lexicon looks like this.

(292) *SLF of* slaapt *'sleeps'*

```
< [store0 SubjectTerm, λY.some(U).[event(U, sleep) & Y(U)] 0erots],
λA.λI.λV.agent_of(V, A) & attime(V, I) & tense(V, pres) >
```

Basically, the event is 'quantified in' the finite or infinite structure of the verb, just like, at some moment in a derivation, the subject or any other argument is. The verb imposes its argument structure on the whole sentence, including its own semantic impact. In the ultimate representation, after derivation, grammatical headness can no longer be traced, as it is irrelevant to meaning and may obscure logical operations. In any case, this explicit way of handling events allows for a representation language that is independent of syntactic realization – an interesting feature with respect to generation (see section 2.7).

As for the typological status of the event variable – *U* and *V* in (292) – we take their type to be *e*, essentially. Assigning an elementary type to them implies that in the semantic representation of sentences we cannot exploit the algebraic structure and the partial ordering of the set of events. Event variables are sorted by the system-predicate *event*, and identifiable as such. As far as we can see, there is no technical objection to typing them differently, however. Montague's *s* for situational indices may qualify. In that case, the intuitive reading of an event structure would be that there is a particular situation to be labelled with a certain verbal concept. *John sings* is read as: there is a particular situation that we label as a *singing* event in which John acts as an agent. There is, however, a linguistic reason not to resort to any different type. All languages are able to put predicative concepts in nominal jackets. These *nominalizations* may or may not have a distribution which slightly differs from other nominal constituents, but they certainly are part of a nominal paradigm. Reckman (2009) and Reckman and Cremers (2007) present an analysis of really predicative nominalizations in the present framework, which shows that it is possible to give full credit to the predicative semantics

of nominalizations while maintaining normal nominal syntax, in particular, quantification and adnominal modification. The sheer concept of nominalizations asks for a syntactically homogeneous treatment. In that respect, there is no reason to switch types; at the end of the day, verbal and nominal predicates converge. This view is also expressed by Khalaily (1997), albeit in a completely different framework.

The second question with respect to reification is whether it applies to all predicates. It is clear that *event* is not the right predicate to classify all verbal predicates. We need *states* of some kind to refer to predications like *lie* and *be at*, following Reckman (2009: ch. 3). There it is also argued that proper nouns and adjectives, at least intersective ones, can be modelled as introducing states. The linguistic evidence is less convincing here. But in Dutch nominal and adjectival predicates can be referred to with the very same set of pronouns that is used for reference to propositions, events and verbal states, and (neutral) noun phrases.

(293) Karel lijkt nogal zenuwachtig de laatste weken. *Dat* was hij eerder niet
Karel seems rather nervous the last weeks. That *was he before not.*
'Karel seems rather nervous these weeks. He was not before'

(294) Hij is dokter. Iedereen kan *dat* hier worden.
He is doctor. Everybody can that *here become.*
'He is a doctor. Everybody can become one here.'

This anaphorical reference is to nominal and adjectival phrases in predicative positions. But there is no evidence that these predicates are different from the ones used in nominal or adnominal positions. Consequently, we take all nouns and adjectives to refer to states, following Parsons (2000). So, the first reading of *Every man seeks a unicorn* would be (295) rather than (242)a.

(295)
```
quant(B,every).[[quant(P, some).[state(P, man) &
theme_of(P,B) ] ⟹ quant(C,some).[event(C, try) &
quant(D,the).[property(D).[quant(A,some).
[quant(Q,some).[state(Q, unicorn) & theme_of(Q,A)] &
quant(E,some).[event(E, beat) & agent_of(E,B) &
theme_of(E,A) & attime(E,F)]]] & agent_of(C,B) &
theme_of(C,D) & attime(C,F) & tense(C,pres)]]]
```

These states introduced by nouns and adjectives are not specified for time. By default, their extension in time covers the temporal extension of the verbal predicate to which their argument is thematically connected. The reasoning here is that, in Dutch, (296) is a simple claim about the present prime-minis-

ter's former career, whereas (297) seems to imply that the then prime-minister had to go to school (again?).

(296) De minister-president is in Zeeland naar school gegaan
the minister-president has <pres+perf> *in Zeeland to school gone*
(297) De minister-president ging in Zeeland naar school
the minister-president went <past> *in Zeeland to school*

This balance in temporal extension can be made explicit, but because it can also be stated as a language default, we leave it out.

With respect to quantification over events and states: in our approach, all predicative objects are bound by a dedicated existential quantifier. This quantifier, although scope-sensitive as such, does not participate in scopal ambiguities, in that it does not take wide scope, for reasons explained above: their interpretation and identification is handled by Skolem functions, providing a contextually parametrized referent from the set of events (see section 2.4.2). Thus, the existential quantifier is treated as a kind of dependent definite description. So we have to explain why events and states are bound by existential operators rather than by the iota-operator or the like. The reason is this. Even if we typologically treat events and states as normal objects, there is no reason to believe that they are unique in their sort at a certain index, as would be the implication of marking them definite, in the sense of being semantically independent and fixed. All that is claimed by the identificational use of a nominal predicate (*the man, a man, no men*) is that some way-of-being of an assumed entity can be labelled with a certain concept. It does not claim that this labelling amounts to rigid naming of the way-of-being, that no other labelling of that particular way-of-being is possible or viable, or that the labelling is axiomatic. The concepts may be rigid designators, as Kripke (1972) argues, but the ways-of-being are certainly not rigidly identified. Moreover, the Skolemization of nominal states has the advantage that the state's identification is made to be dependent on the referent of the predication. In that vein, a sentence like (298) has a quantified logical form like (299) and a skolemized representation like (300). In the latter representation, neither *man* nor *walk* introduces a quantifier. The state is identified by a Skolem function associated with the nominal predicate *man* and parametrized for the valuation of *x*, and the event is identified by a Skolem function associated with the verbal predicate *walk,* which is parametrized for the values of the individual variable *x* and the temporal object *t*.

(298) The man walks
(299) ιx. ∃t. time(t, present) & ∃s. state(s, man) & ∃e. event(e, walk) & theme(s,x) & agent(e, x) & attime(e, t)
(300) ιx. state(f_{man}(x), man) & event(g_{walk}(x, t), walk) & theme(f_{man}(x), x) & agent(g_{walk}(x,t), x) & attime(g_{walk}(x,t), t)

The dependency that comes with Skolemization is comparable to the way in which even extensional adjectives denote properties that are partly determined by the noun class over which they are predicated: the *red* in *this apple is red* denotes a colouring pattern different from the *red* in *this flag is red*. Here too, the way of being red is determined by the choice of the referent, whereas the predication as such is not ambiguous.
Adopting simple existential quantification for states and events, it is certainly tempting to exploit it for marking other semantic phenomena. What immediately comes to mind is the infamous ambiguity between collective and distributive readings for plural and plural-like predications. We could decide to interpret (301) as either one of (302)-(304).

(301) All women are singing
(302) ∀x. ∃y. state(y, woman) & theme(y, x) ⇒ ∃z. event(z, sing) & agent(z, x) & time(z, pres)
(303) ∃y. state(y, woman) & ∀x. theme(y, x) ⇒ ∃z. event(z, sing) & agent(z, x) & time(z, pres)
(304) ∃y. state(y, woman) & ∃z. event(z, sing) & time(z, pres) & ∀x. theme(y, x) ⇒ agent(z, x)

A few aspects are noteworthy. No good interpretation is available for the scopal differences between the universal quantifier and the state quantifier. It seems undecidable whether the state quantifier is semantically independent or not – even when we leave a restriction for the universal quantifier. It is impossible to design a situation in which (302) is true and (303) false. And of course that is due to the intuitive durativity and the non-eventuality of states. We cannot force non-identity of states; they are mass, rather than countable. It is considerably easier to design different models for (303) and (304). The first sentence may be true for an opera agent who observes that each of her divas is actually performing some role somewhere. In that situation, (304) may not apply. It seems, then, as if the possible collectivity of an event can be represented by scopal independency of the event. This presumes, however, that the existential quantification is dynamic, in the sense that its power to bind across sentences is influenced by its referentiality. That is not the case. The anaphorical relation in (305) is viable whether we interpret the first sentence collectively or

distributively. But the anaphor in (306) is only compatible with a wide-scope reading of the existential quantifier.

(305) All students are *having a good time*. I want *it* too.
(306) All students admire *an associate professor*. I know *her* well.

This points to a general phenomenon: propositions, properties, events and states refer *sui generis*. Their behaviour with respect to number or, more generally, their algebra differs essentially, as was argued in *e.g.* Cremers (1993a, 2002). Moreover, referentiality is maintained, even in the presence of negation.

(307) You did not hit Bill. I saw *that*.

In this example, the pronoun refers to a non-event, say a state: the state of not being an event of hitting, of 'you' not being the agent in that event and of Bill not being its theme. It looks as if, as Asher (1993: 215) puts it, '...negation always transforms an event inside its scope into a state outside its scope'. As a matter of fact, the interpretation that accounts for the anaphor must be

(308) the index in space and time for which there is no event of John hitting Bill.

This is the state that no inviting event occurred or John was not its agent or Bill was not its theme. This observation, however, also confirms the position in Asher (1993: ch. 1) that events are not closed under negation: the negation of an event is not an event. In our semantics, the representation of the first clause in (307) would be like (309). After Skolemization, this renders (310)a, which in turn is equivalent to the disjunction (310)b.

(309) $\neg\exists$e. event(e, hit) & agent(e, j) & theme(e, b)
(310) a. $\neg$ (event(f_{hit}(j,b), hit) & agent(f_{hit}(j,b), j) & theme(f_{hit}(j,b), b))
b. $\neg$ event(f_{hit}(j,b), hit) $\vee \neg$agent(f_{hit}(j,b), j) $\vee \neg$ theme(f_{hit}(j,b), b)

The latter representation amounts to the following proposition: whatever the choice function f_{hit} returns as a value on arguments *j* and *b* is not an invitation event or *j* is not its agent or *b* is not its theme. The first disjunct may seem weird, but it is not. A set of events and states induced by the predicate *hit* may contain all kinds of indices, which have in common only the fact that they are selected by application of the concept *hit*. The set may contain the spatio-temporal indices at which no event of invitation occurred: the state of nobody inviting anybody. This index is the value of $f_{hit}(j,b)$ if neither *j* nor *b* was involved in inviting events. This index does not identify an event, and that

is exactly what the first disjunct says: the *hit*-induced relationship between j and b is not an event. The relation of *j* not inviting *b* itself is real and addressable, however. Thus, Skolemization offers an alternative to Asher (1993: 217), who concludes that events force us to '...construe the aspectual and transformational character of negation as affecting a level of semantic interpretation other than denotations'.

The simple fact that predicates always refer, even when absorbing negation, elicits the hypothesis that they are introduced by *Skolem* functions, rather than by scope-sensitive quantifiers. Like choice functions, Skolem functions exist by the axiom of choice. Their existence does not need to be reclaimed in any particular representation. They can be considered as lexically assigned attributes of predicates. Consequently, instead of the representations in (311) and (312) with higher order quantifiers over choice functions as in Winter (2001), DELILAH would produce the Applied Logical forms (313), where e^i and s^i stand for the result of applying an indexed Skolem functions and *i* marks the temporal argument of the function.

(311) All men sing
∀x. ∃g. state(g(x,i), man) & theme_of(g(x,i), x) ⇒ ∃f. event(f(x,i), sing) & agent_of(f(x,i), x) & attime(f(x,i), i).

(312) All students admire an associate professor
∀x. ∃g. state(g(x,i), student) & theme_of(g(x,i), x) ⇒
∃y. ∃f. ∃h. state(h(y,i), assocprof) & theme_of(h(y,i), y) & event(f(x,i), admire) & agent_of(f(x,i), x) & theme_of(f(x,i), y) & attime(f(x,i), i)

(313) a. quant(every, x).state(s^i, man) & theme_of(s^i, x) ⇒
event(e^j, sing) & agent_of(e^i, x) & attime(e^i, i)

b. quant(every, x). state(s^i_1, student) & theme_of(s^i, x) ⇒
quant(some, y). state(s^i_2, assocprof) & theme_of(s^i_2,y) &
event(e^i,admire) & agent_of(e^i, x) & theme_of(e^i, y) &
attime(e^i, i)

In Flat Logical Form, no quantification of the event and state variables will be marked either.

Finally, there is a decisive reason to adopt a neo-Davidsonian event structure. *Extended lexical units* or *constructions* may be such that the optional modification of elements that do not contribute to the meaning of the predicate directly is to be interpreted as a modification at the semantic top level. For example, in the constructions of the type *honger hebben* 'to be hungry', the mass noun *honger* may carry a determiner that conveys the mood of the state: *geen honger hebben* (lit: no hunger have) means 'not to be hungry' and *weinig*

honger hebben (lit: little hunger have) means 'to be a little hungry'. The nominal modification is by-and-large free, but always interpretable at the predicative level. In the semantic set-up of the extended lexical unit *honger hebben,* this transfer of essential semantic information can only be established if at the top-level predication enough 'hooks' are available. Quantification over events and states offers these hooks. It reduces the difference between *nouniness* and *verbiness* (cf. Wetzer 1995) at the semantic level – for syntax this reduction was already pleaded for by Khalaily (1997). Without this cross-categorial equalization, systematic or even finite treatment of semi-transparent extended lexical units does not seem possible. In this respect, neo-Davidsonian event structure leads to normalization and homogenization of logical form, as pointed out by Reckman (2009). The question will be discussed further in section 2.6.

2.5.2 Pluractionality

Modern linguistics has developed an impressive tradition of applying model theory to semantics. With respect to events, this tradition has two foci: the theory of groups and related objects, and the investigation into the nature of events and states. The first domain originates from Link (1983) and enriches the model with entities built out of other entities. The idea is that these entities exist outside language and are introduced – implicitly – into natural languages by processes like pluralization and interfere with interpretation. The other domain is strongly connected to the study of aspect and *Aktionsart* and deals with the way states and events unfold in time and space, and the reflection of these model properties in the way we refer to them in natural-language propositions. There, we have Vendler's classification and, for example, Verkuyl's work on aspect (cf. Verkuyl 1993) and work on the algebra of events, like Bach (1986) and Krifka (1990). The two foci converge in the treatment of what is by now called *pluractionality*: the study of the interaction between quantification and predication, addressing issues like distributivity and collectivity, iterativity and other aspects of simultaneous multiple predication.
Here we want to outline a framework for dealing with pluractionality in a computational way, *i.e.* relating to the construal of logical form and to inference. This framework leans heavily on our conviction that the structure of the world is not reflected in the structures of languages in any interesting way.
First, we do not go along with the idea that, apart from atomic entities, the model for the interpretation of natural language contains *super-atoms* like Linkean groups, nor with its consequence that the semantics of natural languages gives

rise to group-forming operators. As explained in Cremers (1993a) and Cremers (2002), we believe that the group concept is both too strong and too weak: it multiplies entities beyond necessity and it does not properly restrict inferences. The conjunction of two proper names in the following sentence does not refer to a group, since, if it did, the sentence would entail nonsensically that Hijzelendoorn was involved in developing quantum physics.

(314) Niels Bohr and Maarten Hijzelendoorn developed quantum physics.

The sentence is distinctly false because no group {Bohr, Hijzelendoorn} exists and the latter certainly did not contribute in any interesting way to quantum physics in its seminal years or thereafter.
As far as we can see, groups may be ontologically relevant but superfluous in the model theory of natural languages. Moreover, we believe that number is nothing but an agreement feature of natural languages, with as little semantic relevance – relevance for the model and for inference – as case or gender. The tasks that groups fulfil in formal semantics – to account for collective and cumulative readings – can easily be conveyed to decomposition of predicates – a device needed anyway. This will be illustrated below.
Secondly, we do not believe that the phrasing of natural language reflects structural properties of processes in time and space. That is, we do not believe that any constituent in natural language can be held responsible for expressing properties of processes. To put it more bluntly: the algebra of events is not involved in any interesting way in the semantics of natural language. No feature of the semantics of natural-language propositions hinges on the intrinsic nature of an event or state. Natural-language semantics is not about the question which sequence of moves counts as a minimal walking event in a non-metaphorical interpretation of an occurrence of the verb *walk* in

(315) The duck was walking on the water.

Whether this sentence is true or not depends only on whether we accept the description of whatever the duck was doing as an act of walking. If not, we may not even know why we disagree, and still enjoy fruitful communication. It makes no sense to encode a definition of walking as a process of moving and muscling in the semantics of a sentence. Thus, we simply refer to the events predicated, without providing a model-theoretical definition of the event. That definition is for the encyclopedia, not for the lexicon.
Thirdly, we try to tackle problems with pluractionality by adopting an idea by Schein (1993): *essential separation*. In this view, different thematic relations

in a sentence select different events, or every eventual predicate introduces a sequence of events and states. In a sentence like

(316) The boys carried the piano upstairs

for every boy there is an event in which he is the agent and there is a state of being upstairs with a piano as a theme, for example:

(317) For every boy x, x is the agent of some event of carrying and there is a state of being upstairs for some piano.

The analysis that some predicates – *e.g.* telic ones – introduce sequences of events and states is also developed in Arsenijevic (2006), referring to Verkuyl (1972) and Ramchand (2002), among others. Arsenijevic has the following decomposition for telic VPs with three thematic arguments:

(318) [$_{vp}$ [$_{\text{INITIATING SUBEVENT}}$ Participant1 [*lexical_predicate* Participant2]
[(concat) [$_{\text{RESULTING SUBEVENT}}$ Participant2 [*lexical_predicate* Participant 3]]]

As an example, the 'template' [$_{vp}$ *John put the bag in the closet*] is analyzed as

(319) [$_{vp}$ [$_{\text{INITIATING SUBEVENT}}$ John [*add_to/control/contact/place* the bag]
[(concat) [$_{\text{RESULTING SUBEVENT}}$ the bag [*place_in* the closet]]]

In his view, this is a syntactical structure where each sub-event comes with a fully-fledged sentence. The sentences are essentially concatenated, for example by temporal ordering and/or by a causative linking of the type *and therefore*, *and thus*, or *and by that*. The nature of this link is given by the VP. The lexical templates in the analysis are also induced by, or derived from, the VP predicate; the first one, however, can be identified as the *add_to* feature of Verkuyl (1993), and the notion of *contribution* in Cremers (1993a: ch. 2). To implement analysis (318) in a syntactically less demanding framework, we assume that every predicate comes with an internal aspectual structure, by means of decomposition, which for every argument specifies one type of event. We assume that *every* non-intransitive eventive predicate can be decomposed. Consequently, we take the simple transitive sentence (320) to have an eventive structure as in (321):

(320) The boys have painted a car
(321) For every boy x, x is the agent of some *painting*-related event and (by the sum of these actions) some car is in the state of *having been painted*.

Note that in this structure an algebraic relationship between the two events is lacking. In particular, we do not claim that the agentive event is related to a painting event in any interesting algebraic sense. It is neither required to be a sub-event of painting nor is it linked in space or time to any painting event. The *concatenation* of (319) is *cumulative causation*: the sum of the painting related events *leads to* the resultive state in a way that is not specified in the sentence, let alone in the lexicon.

The quantifier on the subject determines the nature of the accumulation. If the quantifier is non-referential, the painting-related event is equivalent to painting and any individual agentive action *leads to* the state of a car having been painted. This is distribution of the subject over the predicate. If the quantifier is referential, no such equivalence is claimed. This leads to the *collective* reading. As a matter of fact, both readings instantiate the cumulative reading of Scha (1981) for sentences like

(322) Nineteen software companies own seven mainframes

where the number of mainframes owned is the sum of the mainframes of all companies. In our approach, the subject's atoms are *always* involved in, and contribute to, what is predicated of the object, and the cumulation's arithmetic – the nature of the sum to be made – is determined by the subject quantifier. Numerals like those in (322) allow for simple addition of the subject atoms' contributions, where each atom is mapped on some rational between 0 and 1, reflecting the impact of the contribution to the Boolean value 1 for the predication over the object. The sentence is true if the sum of the atomic contributions is 1. For real quantifiers, the sum is simple: every atomic contribution is taken to be 1. Real quantifiers can not induce the simple addition that underlies (322).

This approach makes a few predictions. Most prominently, it predicts that only events can introduce ambiguity into the cumulation, since only events can decompose and concatenate in the way assumed above. Nouns, for example, can only be distributive. Furthermore, it predicts that only multi-argument predicates can introduce an opposition between distribution and collectivity. Intransitive verbs are like nouns, in allowing distribution only. The sentence *The men sing* is not ambiguous but univocally distributive. The English verb *meet* is the exception here, which explains why in almost any other language *meet* is transitive, or at least reciprocal. Our analysis dooms *meet* to be an irregular intransitive.

In summary, our format for the eventual structure of a transitive sentence is a decomposition of the verbal predicate into *essential separation* (Schein

1993), with the separated events being related cumulatively according to the quantifier of the 'external argument'.

(323) $Q^1x.\ S^1(x)\ \ \exists e_1\ P'(e_1)\ \&\ agent(e_1,x)\ \&^{cumulativecausation(Q1,\ P,\ P')}\ Q^2y.\ S^2(y)\ \exists e_2\ P(e_2)\ \&$ $theme(e_2, y)$

In this set-up, the connection between the two separated events – represented here as *conjunction-plus* – carries the semantic load of the main verb: a dedicated form of causation, reduced to some arithmetical operation on quantified causes. The connection therefore depends both on the main predicate and on the subject quantifier: the predicate determines the space for what count as causes for – or, more generally, contributions to – its being brought about, and the quantifier determines the mode of addition.
Clearly, this separation can be introduced lexically and the cumulative causation can also be deduced compositionally. That is: a transitive predicate will lexically be associated with a lambda term introducing essential separation and cumulative causation, with a slot for the arithmetical constraint to be borrowed from the subject. To make this work, we have to assume that every predicate P comes with an encyclopaedically determined set of related activities RA^P, which certainly does not only contain predications derived from P 'algebraically'. Moreover, the nature of the relationship between the subject-oriented event and the object-oriented state can be spelled out as a specific instance of a general predicate λQλPλRλxλy.*contributes*(Q,P(x),R(y)) relating a quantifier *Q* and predicate-related events *x* and *y* in order to state that an event *x* of predicate *P* contributes to a degree determined by *Q* – the agentive quantifier – and *R* to bringing about a state *y* of predicate R. The *contributes-relation* is introduced lexically as part of a verb, and indexed for the nature of Q. In that case, the propositional frame (323) turns into (324), and a verb like *carry* comes with lexical specification (325):

(324) $Q^1x.\ S^1(x)\ \exists e_1.P'(e_1)\ \&\ agent(e_1,x)\ \&\ Q^2y.\ S^2(y)\ \exists e_2.P(e_2)\ \&$ $theme(e_2, y)\ \&\ contributes^{Q1}(P'(e1), P(e2))$
(325) $\lambda Q\lambda R.Q(\lambda x.\exists e1.\exists P' \in RA^{carry}\ P'(e1)\ \&\ agent(e1, x)\ \&\ P(\lambda y.\ \exists e2.\ carry(e2)\ \&$ $theme(e2, y)\ \&\ contributes^{Q1}(P'(e1), carry(e2))$

If Q is a real 'distributive' quantifier, P' is identified with *carry* and the related events are identified too: every agentive contribution is set to 1 with respect to bringing about the thematic state. This is equivalent to the merge of P' and P, resulting in normal distributivity. If Q is referential, the function specifies that the P'-event(s) contribute to some non-zero degree possibly lower than 1. It is worth noting that the relative scope of the two argument quantifiers is not

relevant to pluractionality under neo-Davidsonian semantics. The 'number' of carrying events is not related to the specificity of the objects if both quantifiers are scope-sensitive, since the quantification over the event is bound to have narrow scope, as explained before.
This analysis diverges from the approach in Cremers (1993a: ch. 2), in that no adding up of the contributions to 1 is required under referential quantification. In the new analysis, the sentence

(326) The boys carried the piano upstairs

states that each of the boys contributed to the carrying and that the piano ended up upstairs; it does not claim that the contributions of the boys exclude other essential contributions to bringing about the resultative state. The sentence does not exhaust the set of contributors, and so it does not entail

(327) No one else contributed to carrying the piano upstairs.

This lack of exhaustion correlates to the referential nature of the quantifier that is responsible for the *possibly-less-than-one* contribution. The sentence *John and Pete sang a song* does not exhaust the list of song singers: why would *John and Pete built a house* exhaust the list of contributors to the construction of the house, while not exhausting the list of those who built a house?

The *essential separation* approach unfolded above is not yet implemented in DELILAH, in order to not complicate the logical form beyond necessity. Instead, we represent the structure (323) 'simply' in the format

(328) $Q^1x.\ S^1(x)\ Q^2y.\ S^2(y)\ \exists e\ P(e)\ \&\ agent(e, x)\ \&\ theme(e, y)$

The translation from (328) into (324) is straightforward, once the predicate *P* is offered in a decomposed manner.

2.6 EXPLOITING LOGICAL FORM FOR PARSING

2.6.1 Logical Form, Grammar and Computation

We take logical form to be that mode of linguistic analysis in which lexical concepts, inferential semantics and information structure interact. The required analysis is formal, in the sense that it should account for the inter-subjective and – thus – systematic aspects of sentential and lexical meaning. In particular, logical form is a level of grammatical representation at which semantic consequences can be computed – a view already expressed by Higginbotham (1985). Logical form is therefore one of the ultimate targets of linguistic analysis and the representation of what makes natural language a unique module of human cognition. Typically, logical form emanates from deep processing; there are no shallow routes to semantic precision, flexibility and coverage.

The claim that logical form is formal by necessity does not imply that it is bound to comply with first-order predicate logic. Predicate logic does not entertain a privileged relation to semantic interpretation: the set of meanings of natural language is neither a subset nor a superset of the well-formed and interpretable propositions of predicate logic or their complements. It may be a helpful tool in describing certain aspects of the relations between concepts and operators in natural language; in our view, however, it is neither the target nor the anchor of natural-language semantics. A logical form is a representation for which some adequate notion of semantic consequence (entailment) can be defined.

Overlooking modern grammar, it makes sense to state that the relationship between logical form and syntactical structure defines the arena. That is not a trivial observation: though linguistics ranks among the older sciences in the world, it took millennia for the proper balance between form and interpretation to be questioned from a grammatical perspective. In recent times, Bertrand Russell (1949) has argued that logical form is not even homomorphic to syntactical structure. Generative grammar split on the question of with which aspects of meaning grammar could afford to deal (Seuren 1998: ch. 7). In the same period, Montague (1972) defined logical form by compositional interpretation, but correlated it to pseudo-syntax. Pursuing this semantic perspective, categorial grammar started taking the syntax seriously and producing interpretations by derivation, deduction or unification (Ades and Steedman 1982,

Moortgat 1988, Morrill 1994 and much other work in this spirit). Categorial grammar's strong generative capacity, however, challenges linguistic relevance (see chapter 1). At the same time, modern grammar theories appear to converge at some formal link between structure and interpretation. The compositional nature of the main semantic configurations is by now undisputed, after the generative self-reflection of Heim and Kratzer (1998) – that is, if we agree on the lexicon being the only source of semantic wisdom, and on meaning being computable. For a deeper assessment of the relationship between syntax and semantics in grammar, however, see the final chapter.

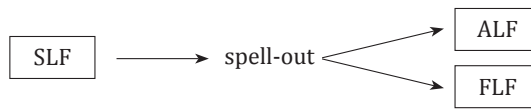
In computational linguistics, logical form is not a very common module of language processing systems. Approaches that avoid incorporating explicit grammar deliberately refrain from logical form, *a fortiori*. Many scholars working on such systems seem to share the scepsis about the computability of meaning that some theoretical linguists entertain. These 'agnostic' strategies govern the field in our days. As a matter of fact, for each language that is targeted by computational efforts, the number of systems doing semantic analysis is quite restricted. And among these, computation of logical form for inference is rare. Notorious icons here are the LINGO enterprise (Copestake and Flickinger 2000), the grammars involved in the Verbmobil project (Wahlster 2000), the PARC XLE parser (Maxwell and Kaplan 1993) and DRT-related approaches, like Bos (2001) and Bos (2004). None of these deal with Dutch.
As for Dutch: memory-based language learning as established in Daelemans and Van den Bosch (2005), does not seem hitherto to have targeted propositional interpretation or any other semantic level. The problem for these learning approaches is twofold: there is hardly any tagging from which propositional semantics can be induced – there is nothing to be stored or learned – and if it existed, the scarceness of data might be overwhelming with regard to the subtlety that propositional interpretation for inference calls for. Robust and wide-coverage parsers like Alpino (Bouma *et al.* 2001, http://www.let.rug.nl/~vannoord/alp/Alpino/) and the older Amazon (http://lands.let.ru.nl/amazon/) do not aim at full logical semantic representations. Inference – the core business of computational semantics according to Bos (2004) – is not addressed.

2.6.2 Logical form in Delilah

Delilah computes three related levels of logical form: Stored Logical Form or SLF, Applied logical form or ALF and Flat Logical Form or FLF (section 2.2).

SLF is *derivational*: it is constructed by applying rules of grammar, which induces unification of complex symbols. It is *compositional* in that it expresses and reflects important aspects of the grammatical structure, and by being built derivationally. Moreover, it is *underspecified*. Important features of the interpretation are not made explicit at SLF. SLF underlies the construal of both ALF and FLF. Specialized algorithms translate SLF into ALF and FLF post-derivationally. Consequently, ALF and FLF are no longer fully compositional in the strict sense: not every aspect of these logical forms is functionally related to the grammatical structure. ALF and FLF both specify all definable features of the interpretation. They differ in the ways of specifying semantic properties. In ALF, matters of scope and semantic dependency are encoded globally and implicitly, as in standard predicate logic. In FLF, scope and semantic dependency are compiled out and made explicit at local levels. FLF acts as a semantic index on ALF.
Therefore, in DELILAH, fully-specified logical form is derived from underspecified SLF by an algorithm that section 2.6.4 argues is exploiting grammatical knowledge.
Alshawi *et al.* (1991) explicitly address the question of why one should entertain multiple levels of logical form. They claim the Quasi Logical Form of the Core Language Engine to be a single level of semantic representation, with additional procedures of resolution providing values for free variables. In that sense, our three logical forms also provide one single representation that consists of several layers. We claim that these different layers can have distinct functions. We certainly do not claim that the logical forms arise from different semantic modes. There is just one process of logical form construal, unfolding at different stages. Here is the picture.

(329) Derivation



2.6.3 Stored Logical Form

2.6.3.1 Construal

Stored Logical Form is constructed independently of derivation. The structure of SLF is defined lexically. It is instantiated during the derivation, but particular derivational moves or procedures do not influence that instantiation. Neither syntax nor control can modify SLF. As a consequence, syntactically different sentences can have the same SLF. Different derivations of the same

sentence will have equal SLFs by definition. As an example: the following sentences will have equal SLFs in a DELILAH-approach.

(330) John was beaten by nobody.
Nobody beats John.
John, nobody beats.
It is John who nobody beats.
It is John who is beaten by nobody.

Therefore, DELILAH realizes the old idea of deep structure. There is a distinctive level of derivation at which all these sentences are equal. They are not born equal, though, but simply converge semantically by unification of lexical specifications. This unification process is steered by syntax and parsing control, but its outcome is independent of that steering. SLF, in this sense, owes its architecture to the concept of Quasi Logical Form introduced and explored by Alshawi (1992).

Each lexical phrase is provided with a template – an HPSG-style attribute-value matrix – that specifies the structure *Store+Body* as the value of the lf-feature *semantics*. *Body* is a lambda term, representing the canonical meaning of the phrase. *Store* contains slots for the logical forms (lf) of dependent phrases: a verb stores the lf of its arguments, a preposition the lf of its complement, *etc.* The storage reflects the combinatoric patterns specified in the syntax. So the store contains lambda terms for all dependent constituents, and not just for scope sensitive operators, as was proposed by Cooper (1983) for early Montague grammar. For each slot in *Store*, the variable it operates on in *Body* is specified. Here is a general scheme for the *Store+Body* structure in a lexical template; all markers are variables, except for the predicates and relations in *Store*.

(331) *Store*: { $\text{DependentLF}_1 \lceil X_1$, ..., $\text{DependentLF}_n \lceil X_n$ }
Body: $\text{operator}_1 \hat{}$...$\text{operator}_q \hat{}\, \text{relation}_1(X_i, X_j)$ & ... & $\text{relation}_p(X_k, X_m)$

The variables in *Store* that stand for the logical forms of dependent constituents – $DependentLF_i$ in (331) – are linked to these logical forms inside the template. In general, they are instantiated by *Store+Body* structures themselves in the process of unification. As the outcome of this graph unification, the template of the computed phrase provides the *Store+Body* structure that contains all relevant logical forms of the sub-phrases involved but underspecifies the interaction of operators. In general, the *relations* in (331) are constants specified in the template itself.

The store may contain elements that do not correspond to syntactic constituents. In particular, the store of a finite verb will contain meanings indexed by its stem. This choice typically reflects the level of morphological decomposition fixed in the lexicon and the nature of the morpho-syntactic interface. For a very simple sentence like *Every man works,* after unifying the relevant templates for *every, man* and *works,* the *Store+Body* structure looks like this:

(332) *Store:* { {*Store:* {*Store:* ∅, *Body:* λX. man(X)⌈Q}
Body: λP.∀y. Q(y) → P(y)}⌈S},
∃z. event(z, *work*)⌈E }
Body: agentof(E, S) &attime(E,T) & time(T, present)

The operator $\varphi \lceil Y$ indicates that the formula φ in store has to be converted as to the variable X in the body: depending on the implicit typing, either the body is a term $\lambda X.\psi$ and type-sensitive β-conversion renders $\psi[\varphi/X]$, or φ is a term $\lambda R.\sigma$, the body is $\lambda X.\psi$ and β-conversion can only yield $\sigma[\lambda X.\psi/R]$. Neglecting these bookkeeping features of the storage, (332) could be read like this:

(333) *Store:* { {*Store:* {*Store:* ∅, *Body:* λx. man(x)}
Body: λQ.λP.∀y. Q(y) → P(y)},
λR. ∃z. event(z, *work*) & R(z)}
Body: λT.λE.λT.agentof(E, S) & attime(E,T) & time(T, present)

The interpretative formula before conversion would be this:

(334) λQ.λP.∀y. Q(y) → P(y)
(λR. ∃z. event(z, *work*) & R(z)
(λT.λE.λS. agentof(E, S) & attime(E,T) & time(T, present)(now)))
(λx. man(x))

The resulting conversions are here, stepwise.

(335) (i) *[now/T]*
λQ.λP.∀y. Q(y) → P(y)
(λR. ∃z. event(z, *work*) & R(z)
(λE.λS. agentof(E, S) & attime(E, now) & time(now, present)))
(λx. man(x))
(ii) *[(λE.λS. agentof(E, S) & attime(E, now) & time(now, present)))/R]*
λQ.λP.∀y. Q(y) → P(y)
(λS. ∃z. event(z, *work*) & agentof(z, S) & attime(z, now) &
time(now, present))
(λx. man(x))

(iii) *[(λx. man(x)/Q]*
λP.∀y. man(y) → P(y)
(λS. ∃z. event(z, *work*) & agentof(z, S) & attime(z, now) &
time(now, present))
(iv) *[(λS.∃z. event(z, work)&agentof(z, S)&attime(z, now)& time(now, present))/P]*
∀y. man(y) → ∃z. event(z, *work*) & agentof(z, y) & attime(z, now) &
time(now, present))

The representation (333) is, therefore, a kind of abbrevation for reasons of bookkeeping of the typed conversion protocol.

The representations for the quantifier, the noun, the verb and the inflectional element occur in a structured and labelled way in the sentential SLF: they are explicitly linked to constituents in the grammatical analysis. In (336), part of the lexical specification of the verb *zag* 'saw', as in *Elke man zag Henk werken* 'every man saw Henk work', is represented. Links between SLF and the other tiers of the analysis are underlined; the link is on indices of type X+Y. All structural variables are in bold face.

(336) *lexical template for* zag *'saw'*

```
ID:A+B
HEAD:CONCEPT:see
     PHON:zag
     SLF:see
     SYNSEM:ETYPE:state  ....
            PERSON:or([2, 3])
            TENSEOP:at-past
            VTYPE:semi_aux
PHON:C
PHONDATA:lijnop(zag, A+B, [arg(left(1), 0, D),
         arg(left(11),wh,E), arg(right(1), 9, F)], C)
SLF:{{[G&(B+H)↑I, {{[J$K&(B+L)↑M], [], []},
    N@some^O^and(quant(O, the), property~[M, O],
    entails1(O, decr), N, entails(O, incr))}&(B+L)↑P,
    Q@some^R^and(quant(R, some), state~[R, see],
    entails1(R, incr),
    and(Q, entails(R, incr)))&(A+B)↑S],[], []},
    and(and(and(and(experiencer_of~[S, I],
    goal_of~[S, T]), theme_of~[S, P]), attime(S, K)),
    tense(S, past))}
SYNSEM:CAT:s_vn
       CONTROL:controls(goal_of~[A+B, T], U~[B+L, T])
       PREDTYPE:nonerg
       TENSE:tensed  ....
TYPE:s_vn\0~[np^0#B+W, np^wh#B+H]/0~[vp^9#B+L]
ARG:ID:B+H
    PHON:E
    SLF:G
    SYNSEM:CASE:nom
           CAT:np
           NUMBER:sing
           OBJ:subject_of(A+B)
           PERSON:or([2, 3])
           THETA:experiencer_of
ARG:ID:B+L
    PHON:F
    SLF:J
    SYNSEM:CAT:vp
           EXSEM:X
           EXTTH:U~[B+L, T]
           THETA:theme_of
ARG:ID:B+W
    PHON:D
    SLF:X
    SYNSEM:CASE:obliq
           CAT:np
           OBJ:dirobject_of(A+B)
           THETA:goal_of
```

This template clarifies that whatever plays an active role in SLF is anchored in the grammatical analysis by a network of variables, open to dynamic instantiation. In this sense too, SLF is compositional and derivational.

2.6.3.2 Application of SLF: disambiguation

SLF reflects the morphological-syntactical complexity of the phrase: every constituent that contributes to the meaning of the whole phrase is represented in the relevant *Store+Body* structure. For this reason, different SLFs of one sentence correlate to different meanings. This property of SLF can be used to determine to what extent *extended lexical units* or 'constructions' have contributed to an SLF and, thus, what degree of lexical aggregation it represents. Normally, an interpretation with a high degree of lexical aggregation is preferred over an interpretation that was not, or less, based on extended lexical units. For the sentence *Nobody has kicked the bucket until now* the reading *Until now nobody has hit the bucket such that it fell* should have a considerably lower priority than the reading *Until now nobody has died.* In DELILAH, comparing SLFs in this respect suffices for selecting the least naive interpretation.
It has been widely acknowledged by now – and it has never been denied, for all we know – that the lexicon of a natural-language processing system not only specifies single words; it must also – and maybe even predominantly – contain phrases or phrase structures that have a specialized syntax and/or semantics. At present, scholars proclaiming Construction Grammar (cf. Croft 2001) stress the central role of non-atomic construal in natural language. In formal grammar and computational linguistics applicants of HPSG, Categorial Grammar and Tree Adjoining Grammar always have accounted for considerably more involved structures than just atomic units.
Poß and Van der Wouden (2005) show that there is a huge variety of ways in which words and structures cluster to build complexes with specialized syntax or semantics. In general, lexical units limiting lexical selection or syntactical transparency will come with a meaning to which not every proper part contributes according to its proper class. Here are just some examples from Dutch:

- *hebben* with a DP can mean *possess*, among others; together with mass nouns like *honger* and *dorst*, it expresses in all its morpho-syntactic varieties the property of being hungry (thirsty); the verb itself hardly contributes to this meaning; the construal *to possess hungriness (thirstiness)* cannot be barred from being parsed;
- prepositional phrases may introduce adjunctive modifiers, but very often verbs select certain prepositions to express specialized meanings, e.g. *werken op iemands DP*, meaning *affect somebody's DP*, where the DP

introduces a psychological concept; parsing cannot be withheld from an adjunctive construal, but the selected meaning deserves priority;
- intransitive verbs V can be part of the so-called Dutch *way*-construction, expressing the meaning of moving to a certain position by V-ing, e.g. *to laugh oneself a way/path to DP*; the characteristic DP with the *way*-type NP does not play a role in the semantics.

In all these cases, the *Store+Body* structure representing the meaning of the extended lexical unit is simpler than the SLF resulting from a parse that does not account for the lexicalized meaning. In particular, the Store will contain less distinct lambda terms. Crucially, we do not presume that the semantic structure of an extended lexical unit is simpler in any logical or arithmetical sense. All we claim is that the store of an extended lexical unit will show less internal structure than its distributed counterpart. This follows from the construal of SLF. Therefore, a simple measurement on the stores of SLFs may select the simplest and highly aggregated and most 'normal' reading.
Here is an example. Any sentence containing phrases of the type *honger hebben* will come with two analyses: one reflecting the reading 'be hungry', the other reflecting the reading 'possess hunger'. The latter SLF will contain a store where the lambda term for the noun 'hunger' is specified. The former SLF will have an empty store instead, as the lambda term for 'to be hungry' irrespective of its logical complexity will not be stored, but specified in a body field of SLF. Computing the aggregated complexity of the stores and comparing them will identify the former SLF as the simpler one. Since this SLF is part of a full analysis, we can select this analysis as the preferred interpretation. The case is illustrated for the sentence *ik heb honger* 'I possess hungriness / I am hungry', with simplified representations of the *Store+Body* structures for the sake of transparency. The preferred reading has a simpler store than the non-preferred one.

(337)
SLF 1: `{ { i:x }, be_hungry(x) }`
SLF 2: `{ { i:x, hungriness:y }, possess(x, y) }`

Delilah produces all readings permitted by the grammar. For each SLF, it defines the structural complexity, *i.e.* the degree of embedding of *Store+Body* structures. For every acceptable parse, it computes a number that expresses the ratio between the structural complexity of the SLF and the total number of terms specified in the stores. This ratio marks the semantic complexity of the SLF. Parses can be ordered according to these ratios. In addition, Delilah computes the total number of predicates or small clauses in the bodies of the

SLFs. This number is used as a secondary marker for semantic complexity, but does not necessarily discriminate between lexical units and semantically composed phrases.
The approach to reading selection on the base of SLF complexity is steered by some principles:

(338) *Compositional complexity rul*e
The degree of compositionality of a phrase corresponds to the number of lambda terms in its store as compared to the number of syntactical arguments.

This gives an immediate first approach to the compositional complexity of a lexical sign.

(339) *Compositional complexity of a lexical sign*
The compositional complexity of a lexical sign equals 0 if the store is empty; otherwise, it is the number of lambda terms in its store divided by the number of proper parts that the sign specifies.

The compositional complexity is 0 for, *e.g.*, simple one-word phrases. It is $n/n = 1$ if all n specified proper parts of a construction contribute to the meaning of the whole phrase. It is a rational number if the construction is not purely compositional.
Apart from this simple metric, one could also count the number of semantic primitives and/or the number of variables in the compositional logical form, either in store or in the body or both, but we consider those data to be secondary, though not necessarily uninformative. Of course, metrics other than the two above for measuring compositional complexity are also conceivable.
Measure (339) extends to non-lexical signs after unification by making it recursive.

(340) *Complexity of Stored Logical Form*
The (compositional) complexity of a Stored Logical Form is the sign complexity plus the sum of the Stored Logical Form complexity of all the terms in its store.

In this we have a purely semantic, yet quantitative criterion for selecting readings. If two analyses differ only in their logical forms, choose the one with the lower compositional complexity of its Stored Logical Form. That analysis will contain the most aggregated meaning. It entertains at least one more extended lexical unit (elu) than the competitor.

In NLP, the most aggregated reading will be preferred, generally, since elus are listed in the lexicon because their aggregated meaning is more prominent than the full composition of their individual meanings. The complexity measurement therefore results in the following selectional strategy:

(341) *Underspecification selection rule*
If two signs differ only in compositional complexity of Stored Logical Form, the one with the lower compositional complexity of Stored Logical Form represents the lexically preferred prominent reading.

This way, the selection is driven by the identification of lexical or constructional units and the evaluation of their interpretations. As an example, we can see that among the two Stored Logical Forms for (342), the underspecification selection rule easily identifies the *be hungry* reading of sign (343) as lexically preferred over the *possess hunger* reading of sign (344), by comparing their SLF complexity.

(342) Elke man heeft honger
Every man has hungriness

(343) *reading 1*

```
head:[phonology:/heeft/, semantics:() ...]
      argument(1):[semantics:< store:{
                               < store: [],
                                 body:λP.λQ.∀Z.P(Z)→ Q(Z) > #Y},
                               body: λY.man(Y) >,
                  cat:np ...]
      argument(2):[phonology:/honger/, cat:np, semantics:....]
      semantics: < store:{
                  < store:{
                    < store:[],
                      body: λP.λQ.∀Z.P(Z)→ Q(Z)> #Y }
                    body: λY.man(Y)> #X },
                  body: λX.be_hungry(X) >...
```

SLF complexity according to (339): 1/2 + 1/1 = **1.5**

(344) *reading 2*

```
head: [phonology:/heeft/, semantics: possess, ... ]
argument(1):[cat:np, role:theme, ...,
             semantics: < store:{
                          < store: [],
                            body: λP.λQ.∀Z. P(Z) → Q(Z) > #Y},
                          body: λY.man(Y) >]
argument(2):[phonology:/honger/, cat:np,
             semantics: < store:{
                          < store: [],
                            body: λP.λQ.∃Z. P(Z)& Q(Z)> #X},
                          body: λX.hungriness(X) >]
semantics: < store:{
             < store:{
               < store: [],
                 body: λP.λQ.∀Z. P(Z) → Q(Z) > #Y },
               body: λY.man(Y) > #W,
             < store:{
               < store: [],
                 body: λP.λQ.∃Z. P(Z)& Q(Z)> #X },
               body: λX.hungriness(X) > #V }
             body: λW.λV.possess(W,V) >
```

SLF complexity according to (339): 2/2 + 1/1 +1/1 = **3**

Delilah is primed to pick the simplest reading for a sentence by exploiting the underspecification of SLF. This method is very reliable, because it is anchored in lexical specifications. In the analysis of the sentence *Sommige kinderen hadden weinig honger* ('some children were not very hungry') the *possess*-reading will be suppressed. This is because the lexicon contains an extended lexical unit relating *honger* and *hebben*, the SLF of which is simpler than the SLF constructed from independent lemmas for *to have* and *hunger*. The SLF for the whole sentence is constructed by sheer unification, including an integrated interpretation for the quantificational parameter *weinig* 'little'. Therefore, the complexity of the lexically specified SLFs is conserved during the derivation.

This approach to resolving lexical ambiguity lives off the layered structure of SLF. The basic idea is that the semantic elements which are contributed freely – without pre-derivational lexical commitment – are stored, whereas all other semantic specification dwells in the *body*-part of SLF. Free combinatorics thus increases the complexity of the stores, by definition and exclusively. Measuring the storage is therefore measuring the degree of combinatorial freedom. Note that this variation in degrees of freedom can only be accounted for at SLF, that is, before all semantic interaction is mixed up in a complex proposi-

tion. Consequently, in DELILAH, the structured SLF is an essential feature of lexical items, whether complex or simple.

Underspecified logical form therefore offers a natural and grammatically precise way to tell 'heavy' compositional readings from 'light' constructional readings while parsing, without any additional information being needed. It only refers to the semantic nature of the construction and the way in which (proper) parts of the construction contribute or do not contribute to the overall meaning. Since the overall meaning of constructions must be specified lexically anyway, no additional data are required for the metric to be applicable. The selection procedure comes free with a lexicon that allows for deep processing and semantic inference by specifying constructional meanings.

2.6.4 Applied Logical Form

In our approach, there is no special grammatical task for ALF. At the same time, this format – or a variant of it – complies best with requirements of human readability. To illustrate that, we repeat the (labelled, edited and 'normalized') SLF, the disjunctive FLF and the ALFs of the sentence *Elke man zoekt een eenhoorn* 'Every man seeks a unicorn' (from section 2.2).

(345) Stored Logical Form

```
< [store0
<<[store1 λX.state(X, man) 1erots] λJ.λK. quant(I, every) & J(X) &
theme_of(X,I) & entails1(I, decr) & K(I) & entails(I,incr)>>,
<<[store2 <<<[store3 <<<<[store4 λY.state(Y, unicorn) 4erots],
λQ.λR.quant(P, some) & Q(Y) & theme_of(Y, P) & entails1(P,
incr) & R(P) & entails(P, incr) >>>> 3store],
λT.quant(U, some) & event(U, find) & entails1(U, incr) & T(U) &
entails(U, incr)>>> 2erots],
λW.λU.λP.λF. agent_of(U, I) & theme_of(U, P) & attime(U, A) >
1erots]
λX.λY.quant(Z, the) & property(X(L)), Z) & entails1(Z, decr) &
Y(Z), entails(Z, incr)>>,
λC. quant(D, some) & event( D, try) & entails1(D, incr) & C(D)
& entails(D, incr)
0erots],
λI.λZ.λD.λA. agent_of(D, I) & theme_of(D, Z) & attime(D, A) &
tense(D, pres) >
```

(346) Flat Logical Form

```
state( s^i, man) &
theme_of( s^i, G+↓+every+[] ) &
event( e^j, try) &
property(D+↓+the+[]) &
event(e^k+[D], find) &
agent_of(e^k+[D], G+↑+every+[]) &
  (theme_of(e^k+[D], H+↑+some+[D,G] );
   theme_of(e^k+[D], H+↑+some+[G] );
   theme_of(e^k+[D], H+↑+some+[] ) ) &
state( s^j, unicorn) &
  (theme_of(s^j, H+↑+some+[D,G] );
   theme_of(s^j, H+↑+some+[G] );
   theme_of(s^j, H+↑+some+[] ) )&
attime(e^k+[D], F) &
agent_of(e^j, G+↑+every+[]) &
theme_of(e^j, D+↑+the+[]) &
attime(e^j, F) &
tense(e^j, pres)
```

(347) Applied Logical Form

(a)

```
quant(G,every).[[state(s^i, man) & theme_of(s^i, G)] ⟹ event(e^j,
try) & quant(D,the).[property(D).[quant(H,some).[state(s^j,
unicorn) & theme_of(s^j, H) & event(e^k, find) & agent_of(E,G) &
theme_of(e^k,H) & attime(e^k,F)]]] & agent_of(e^j,G) &
theme_of(e^j,D) & attime(e^j,F) & tense(e^j,pres)]
```

(b)

```
quant(G,every).[[state(s^i, man) & theme_of(s^i, G)] ⟹ event(e^j,
try) & quant(H,some).[state(s^j, unicorn) & theme_of(s^j, H) &
quant(D,the).[property(D).[event(e^k, find) & agent_of(e^k,G) &
theme_of(e^k,H) & attime(e^k,F)]] & agent_of(e^j,G) &
theme_of(e^j,D) & attime(e^j,F) & tense(e^j,pres)]]
```

(c)

```
quant(H,some).[state(s^j, unicorn) & theme_of(s^j, H) &
quant(G,every).[[state(s^i, man) & theme_of(s^i, G)] ⟹ event(e^j,
try) & quant(D,the).[property(D).[event(e^k, find) &
agent_of(e^k,G) & theme_of(e^k,H) & attime(e^k,F)]] &
agent_of(e^j,G) & theme_of(e^j,D) & attime(e^j,F) &
tense(e^j,pres)]]
```

Just like FLF, ALF is spelled out post-derivationally from SLF. It is cast as enriched predicate logic, in the sense that it must accommodate operators and quantifiers other than strictly logical ones. Contrary to FLF, it is neither a conjunction, nor operator-free. Just like FLF, all semantic dependencies are unambiguously and fully encoded. ALF is structured by logical operators.

ALF can best be seen as the logical label of the semantic process. We assume that its role in natural-language processing is limited. Yet, the number of ALFs delimits the number of readings at FLF. The multiple disjunction of local readings in (346) suggests an explosion of readings. This is not intended. Reckman (2009) presents an index on FLF to the effect that in a certain disjunction the choice of one reading implies the choice of a local reading elsewhere – to wit, the same one. This index on readings is not represented here, but is assumed to be part of the FLF-generation procedure.

2.6.5 Flat Logical Form

2.6.5.1 Construal

Each SLF that passes the complexity test is submitted to a post-derivational algorithm, outlined in section 2.3.3, in particular in (272). This algorithm performs two tasks: it converts the SLF into a coherent formula and it compiles onto each variable the additional semantic information resulting from this conversion.
The first step involves β-reduction of implicitly typed lambda terms, while explicitly specifying scope dependencies between semantic operators like quantifiers, modalities and negation. This conversion is sensitive to scopal variation of operators, to the semantic nature of operators and to structural restrictions on scope imposed by islands or other intervention effects. In this step, a good deal of a semantic theory can be effectuated, for example with respect to islands, scope and negation. This step yields a more or less standard logical form. All quantifiers, though, are described rather than interpreted as operators (as *e.g.* in (295) and (313)), to allow for quantificational variety.

In the second step, the information spelled out by the conversion is specified at each occurrence of each variable by a compilation protocol that turns the semantic operators superfluous. After applying this protocol, each variable is locally sugared with information as to

- its entailment property,
- the quantificational regime it is bound to,
- the variables its instantiation is dependent upon.

The entailment property indicates whether the predicate the variable is an argument of allows for upward, downward or no entailment with respect to the variable. The specification of 'governing' variables indicates whether a

variable is referentially independent or has to be valuated by a choice function; the notion of variables y_i governing a variable x thus amounts to the introduction of a choice function $f(y_1,..,y_n)$ for x, claiming: given values for y_i there is a way to determine a value for x such that the proposition resulting from these valuations is true.

The information encoded on the variables is compiled from an intermediate representation where lambda terms are fully converted and scopes are specified. The index with respect to entailment – *up* and *down* – specifies the local monotony properties of the binding quantifier in the relevant domain, according to its definition as a general quantifier. For simple, lexical determiners this is straightforward, as these properties are lexically defined by the theory of generalized quantifiers (in Barwise and Cooper 1981, and Zwarts 1986, for example). For complex determiners like *at least n but not more than m* entailment properties have to be computed from the composition; in many of these cases no monotony can be detected, however. This calculus is discussed in Reckman (2009).

The index with respect to the binding quantifier can be complex, as the quantifier itself is complex. Yet, complex quantifiers too should be classified in such a way that *e.g.* existential impact of the quantifier can be derived immediately. Note that the entailment property of the variable *y* varies with the domain of its quantifier: in the restriction of the universal quantifier, the variable bound by it allows for downward entailment in the nuclear scope it gives rise to upward entailment. Referential dependency does not vary with domains.

In our system, FLF is derived from a purely semantic, fully-specified logical form ALF (cf. section 2.6.4). Neither this logical form nor its derivative FLF contains any language-specific information. Therefore, if two sentences mean the same, their logical forms must be equivalent – which is undecidable – and their FLFs will comply. Paraphrasing an example of Shieber (1993), the following five sentences share one particular canonical and normalized semantic representation, in so far as they share a meaning:

(348) Clapton led Derek and the Dominos
Derek and the Dominos was led by Clapton
The leader of Derek and the Dominos was Clapton
Clapton yazzℜr ĩ Derek ɔd the Dominos
Clapton yămôs amuzăr ɔn Derek ɔd the Dominos

According to the requirements above, their FLFs, when fed into appropriate generators for English and Tuareg respectively, should give rise to these, and maybe even more, sentences. Note that looking at logical form in this way implies that it is underspecified in comparison to natural-language sentences inducing it. This underspecification is inevitable, even when the logical form itself is fully specified from a logical point of view: sentences 1 to 5 are neither equal nor equivalent, but their logical forms are.

As a matter of fact, Flat Logical Form is an *index* on ALF. The FLF is derived by taking every single predicate term at ALF and indexing every variable in that term for the way it is bound, for polarity and for the variables on which its valuation depends. Here are two examples.

(349) Every man invites a woman that he does not know

standard first-order representation:

```
∀x. [M(x) ⇒ ∃y. [W(y) & I(x,y) & ¬K(x,y)]
```

logical form ALF:

```
∃e1 ∃e2 ∃e3 ∃e4 ∀x. [[state(e1, M) & th1( e1,x)] ⇒ ∃y.
[state(e2, W) & th2(e2, y) & state(e3, K) & ¬[th3( e3, x) &
th4(e3, y)] & event(e4, I) & th5( e4, x) & th6(e3, y)]]
```

index FLF:

```
{ state(e1+↓+some+[], M), th1(e1+↓+some+[], x+↓+all+[]),
  state(e2+↑+some+[], W), th2(e2+↑+some+[], y+↑+some+[x]),
  state(e3+↓+some+[], K), th3(e3+↓+some+[], x+↓+no+[]),
  th4(e3+↓+some+[], y+↓+no+[x]), event(e4+↑+some+[],I),
  th5(e4+↑+some+[],x+↑+all+[]), th6(e4+↑+some+[],y+↑+some+[x])}
```

(350) Few monkeys dare to fly

standard – no first-order standard

logical form ALF:

```
∃e1 ∃e2 ∃e3 Few(x). [state(e1, M) & th1( e1, x) & state(e2, D)
& th2(e2, x) & iy. [property(y) & th3(e2, y) & th5(y, e3) &
event(e3, F) & th4(e3,x)]]
```

index FLF:

```
{state(e1+↓+some+[], M), th1(e1+↓+some+[], x+↕+few+[]),
 state(e2+↑+some+[], D), th2(e2+↑+some+[], x+↕+few+[]),
 property(y+↕+the+[e2]), th3(e2+↑+some+[], y+↕+the+[]),
 event(e3+↑+some+[y], F), th4(e3+↑+some+[y], x+↕+few+[]),
 th5(y+↕+the+[e2], e3+↑+some+[y])}
```

Of course, the construction of ALF involves numerous decisions on the semantics of natural language, many of which are not consolidated (cf. Cooper *et al.* 1996). These decisions, like introducing intensional domains or assigning wide scope to state and event quantifiers, are not at issue here: whatever

clause occurs in ALF is indexed at FLF. The index is computed along with LF, and can be exploited for computing entailment between sentences and sentence generation.

The exploitability of the FLF index on logical form resides in its permutability. Given an FLF, one can select the subset of clauses that share a particular variable. Let us call this subset a *net.* For each net, a normal form can be constructed, *e.g*. by some ordering of the predicates involved. Here is a complete network for the FLF of (350):

(351)

e1-net: `< state(e1+↓+some+[], M), th1(e1+↓+some+[], x+↓+all+[]) >`
e2-net: `< state(e2+↑+some+[], W), th2(e2+↑+some+[], y+↑+some+[x])>`
e3-net: `< state(e3+↑+some+[], K), th3(e3+↑+some+[], x+↑+no+[]),`
`th4(e3+↑+some+[], y+↑+no+[x]) >`
e4-net: `< event(e4+↑+some+[], I), th5(e4+↑+some+[], x+↑+all+[]),`
`th6(e4+↑+some+[], y+↑+some+[x]) >`
x-net: `< th1(e1+↓+some+[], x+↓+all+[]), th3(e3+↑+some+[], x+↑+no+[]),`
`th5(e4+↑+some+[], x+↑+all+[]) >`
y-net: `< th2(e2+↑+some+[], y+↑+some+[x]),`
`th4(e3+↑+some+[], y+↑+no+[x]), th6(e4+↑+some+[], y+↑+some+[x])>`

The effort to construct the network of the FLF index is proportional to the number of variables in the LF. The network is the anchor for computation of entailment and for generation. It seems a characteristic property of natural-language Logical Form, reflecting the semantic coherence of sentences, that in its FLF every variable defining a net occurs in at least one other net. It is worthwhile to point to this form of connectivity by way of conjecture.

(352) *Connectivity conjecture for FLF*
Given the LF of a well-formed and meaningful sentence S, in the network that represents its FLF, every indexed variable that defines a net occurs as an indexed variable in at least one other net.

2.6.5.2 Application: inference

The notion of entailment for unrestricted natural language is far from being logically validated. From a processing point of view, entailment is problematic on both the logical and the semantic sides. As for the semantics, many – if not most – constructions of natural language lack any consolidated interpretation. As for the logic, even first order representations of natural language are too rich to be left to theorem-provers. In addition, Shieber has pointed out that equivalency for first-order logic, being undecidable, controlled generation of (unrestricted) natural language from pure first-order logic con-

straints is not feasible (Shieber 1993). This means that reasoning in natural language by entailment, using first-order logic, cannot induce provably correct verbalization of the outcome.
It is not just a trend that these days textual entailment is generally computed by shallow means – not by full grammatical representations. The RTE challenges show that relatively shallow analysis, combined with intelligent search and/or learning strategies, may cover a fair amount of 'common sense' entailment, as was pointed out by *e.g.* Chambers *et al.* (2007, Tatu and Boldovan (2007) and Bobrow *et al.* (2007). Moreover, deepening the grammatical analysis does not necessarily improve the result – the argument here is by Bos and Merkert (2006). Yet, the constructional nature of unrestricted natural language calls for deep processing: the lexicon for processing is bound to be phrasal, to contain complex lambda terms and to be submitted to subtle syntactic combinatorics in order to maintain phrasal semantic integrity. The phrasal lexicon is the source of all semantic wisdom, but in order to exploit it in processing a lot of grammar is required. Cooper *et al.* (1996) argue convincingly that several tasks for natural-language processing require structural semantic processing – deep structural semantic processing.

FLF is an operator-free conjunction of disjunctions. Each conjunct indexes a part of the ALF. All logical information is specified in every small clause as indices on variable occurrences. This representation intends to facilitate inference. To decide whether a certain inference is possible, it suffices to linearly inspect an FLF and, for each conjunct, to locally decide whether or not it gives rise to (part of) the hypothesis, *i.e.* the sentence to be inferred.

Here, we propose a means of exploiting deeply processed and fully-specified logical form for the sake of computing entailment for unrestricted natural language and for generation. The representations used can be computed, and are taken to enhance semantic inference between sentences and to feed into controlled generation (see section 2.7).

Our strategy is to approach the modal notion *S entails T* for Dutch sentences S and T by inspecting their respective FLF networks. We define a notion of *l-coverage* for FLF networks and suggest that l-coverage of FLFs is a sufficient condition for entailment.

(353) For Dutch sentences S and T, S *entails* T if FLF(S) *l-covers* FLF(T).

In order to compare the FLFs, we must assume that their respective networks can be compared *salve* alphabetic variance. This is far from trivial, but we will not pursue this here, since this problem is more general; we simply assume that an educated guess as to the line-up of variables in the two FLFs can be made.
Under this assumption, we first define the notion of *i-coverage* for individual FLF-clauses φ and ψ. Below is a tentative inventory of its instances; each instance is illustrated by a natural-language entailment relation in which that instance of *i-coverage* is intended to be the decisive licensing factor – the phrases involved are in small caps. In this definition we take <φ,ψ> to be a *subnet* – the clauses share a variable by definition. σ stands for two-place relations between an indexed variable and a conceptual constant, like *state* and *event*; τ abbreviates all relations between indexed variables, like thematic roles. Furthermore, we assume that between concepts and predicative constants a subsumption order may be defined, by meaning postulates and/or by ontologies such that X↓ ⊑ X↑ means that X↓ is at least as specific as X↑. Moreover, lists of governing variables are supposed to be in alphabetical variance when checked; this is indicated by the notation F/F'. Finally, an *intensional net* is a net which is headed by an intensional predicate like *proposition/1* or *property/1* and the defining variable of which is dependent on some other variable.

(354) I-coverage

(a) *identity:*

φ *i-covers* ψ

if φ and ψ are alphabetic variants

(b) *strengthening*:

φ[F] *i-covers* ψ[F']

if φ and ψ are alphabetic variants except for the list of parameters F and F', and F-F' does not contain variables that define an intensional net in the FLF of the covering clause and F'-F does not contain variables that define a net in the FLF of the covered clause

(These men kissed A WOMAN *entails* SOME WOMAN is kissed;
He said that A MAN died *not-entails* A MAN died;
SOME WOMAN kissed him *not-entails* These men were kissed by A WOMAN)

(c) *upward coverage:*

<σ(A,C),τ(A, x+↑+Q+F)> *i-covers* <σ(A,C↑), τ(A, y+↑+Q+F')>

if C↑ is defined by subsumption

(Every candidate is GERMAN *entails* Every candidate is EUROPEAN)

(d) *downward coverage:*

<σ(A,C), τ(A, x+↓+Q+F)> *i-covers* <σ(A,C↓), τ(A, y+↓+Q+F')>

if C↓ is defined by subsumption

(At most three candidates are EUROPEAN *entails* At most three candidates are GERMAN)

(e) *downward extension:*
<σ(A,C), τ(A, x+↓+Q+F)> *i-covers*
<σ(A,C), τ(A, y+↓+Q+F'), τ'(A, y+↑+Q'+[ ]) >
if τ' does not occur in the covering net
(No woman gave flowers *entails* No woman gave flowers to me)

(f) *upward reduction:*
<σ(A,C), τ(A, x+↑+Q+F)> *i-covers* σ(A,C)
(John ate an apple *entails* John ate)

(g) *existential impact:*
τ(A, x+M+Q+F) *i-covers* τ(A, y+M+some+F')
unless Q = *no*
(That student read a book *entails* Some student read a book
Few students read a book *entails* Some student read a book)

(h) *upward concept:*
σ(e1+↑+Q+F, C) *i-covers* σ(e2+↑+Q+F', C↑)
if C↑ is defined by subsumption
(She is a girl *entails* she is a woman)

(i) *downward concept:*
σ(e1+↓+Q+F, C) *i-covers* σ(e2+↓+Q+F', C↓)
if C↓ is defined by subsumption
(At most three students were ill *entails* At most three students had the flu)

(j) *upward relation:*
τ(A, x+↑+Q+F) *i-covers* τ↑(A, y+↑+Q+F')
if τ↑ is defined by subsumption
(This man hit the woman entails the man did something to the woman)

(k) *downward relation:*
τ(A, x+↓+Q+F) *i-covers* τ↓(A, y+↑+Q+F')
if τ↓ is defined by subsumption
(No men did anything to this woman entails No men hit this woman)

(l) *distribution:*
<φ, ψ> *i-covers* <φ', ψ'> iff φ *i-covers* φ' and ψ *i-covers* ψ'

This list of elementary coverings is rather tentative, but the ambition will be clear: the relationship between FLFs can be constructed from more elementary relations at the level of nets and subnets. In linguistic terms, it means that, by using the FLF index, we are trying to reduce the relation between (meanings of) sentences to a relation between (meanings of) phrases, notwithstanding the complex logical and syntactical interaction between these phrases. On this basis, we can define a notion *n-coverage* for the relationship between nets.

(355) A net N is *n-covered* by a net M if every clause in N is *i-covered* by M.

The *i-coverage* almost has to be functional in this definition: in the list of pairs of i-covering and i-covered subnets, no subnet can occur in an i-covering posi-

tion more than once, with the possible exception of applications of *downward extension* (354) (e).
This leads to the following notion of *l-coverage* between FLFs.

(356) *L-COVERAGE*
An FLF $\Phi = < \varphi_1, \ldots \varphi_k >$ *l-covers* an FLF $\Psi = < \psi_1, \ldots \psi_m >$ iff
for every network $N_\psi = <\psi^1 \ldots \psi^n>$ in Ψ there is exactly one (possibly empty) network $N_\varphi = <\varphi^1 \cdots \varphi^m>$ that *n-covers* it.

In order to back up conjecture (353), which introduces l-coverage as a sufficient condition for entailment, we argue that we essentially treat the FLF index as a conjunction of independent clauses. FLF abstracts away from any non-commutative connective at LF, in particular, from the material implication. Although FLF-indexing itself is non-reversible, one observation is relevant: conjunction entails implication if we do not allow for *ex falso* interpretation of implication in the combinatory semantics of natural languages. Seuren (2006) argues that the cognitive-pragmatic construal of natural language does not leave space for zero-valued antecedents. Following his lead, we assume that phrasal semantics does not induce *ex falso* interpretation – the universal quantifier has existential impact. Under this assumption, the 'conjunctive' logic of FLF as presented above is more restrictive than the logic of ALF, which goes proxy for the relation between sentences. In that case, conjecture (353) seems a fair and cautious approximation of entailment in natural language.

2.7 GENERATING FROM LOGIC

2.7.1 Generation as translation

In the preceding section, DELILAH was explained as computing – apart from a semi-standard full representation ALF – two related but procedural and formally very distinct levels of semantic representation: underspecified, compositional SLF and fully-specified, para-compositional FLF. SLF is produced as part of the graph unification, which is the kernel of the parsing and generation procedures. Therefore, its construction does not complicate the derivation. Yet, on SLF, measures can be defined that express essential semantic properties of the structure and can be exploited for selection of readings and

reduction of ambiguity. Developing the best and most telling measurements in this respect is an empirical matter, but it is clear that hard-boiled criteria can be found and applied in reducing ambiguity exactly where it resides: in the semantic structure of a sentence. Of course, not all problems of selecting readings can be handled at SLF. Pure global polysemy cannot be decided upon by inspecting SLF. But the delicate balance between levels of lexical aggregation can most certainly be tracked at this level.

SLF seems a good level for generation to take off because of its sensitivity to constituent structure. We aim at a procedure to generate sentences the SLF of which gives rise to an FLF that entertains a strict logical relationship to another FLF, the generation constraint. That is, the procedure starts with analyzing an FLF, proceeds with guessing a syntactical structure on the basis of that information, then spells out its SLF and compares the result with the original FLF. In this sense, the labour division between SLF and FLF may contribute to purely meaning-driven translation, as argued in Alshawi (1991) and Copestake *et al.* (2005). Rosetta (1994) made clear that machine translation on a semantic base – this is *not* a pleonasm nowadays – flourishes by strict and controllable compositionality. FLF approaches the representation of meaning in the spirit of Minimal Recursion Semantics (MRS) as implemented in the English Resource Grammar (Flickinger *et al*. 2000, Copestake *et al.* 2005). The main point of convergence is that MRS and FLF present full semantics while avoiding syntactical complexity of the logical form. There are some differences, however. Since FLF is derived from SLF, all constraints on the interaction of semantic operators – like weak island conditions – are integrated in this derivation. The construal of FLF thus embodies an explicit theory on the syntax-semantics interface. Furthermore, FLF encodes all scopal and inferential information directly onto the logical form, rendering inference strictly local, as was pointed out in section 2.6.5.2.

In the following sections, we present a system for generating natural language that acts as a semantic translation automaton. Both the source and the target are FLFs; the precise semantic relationship between these two FLFs can be computed, and yields the criterion for the generation's success. In between, Delilah's generator explores the syntactic-semantic interface, deriving well-formed sentences with SLFs to be converted into FLF. Schematically:

(357) *Generation procedure*
- (a) input: FLF InputFLF
- (b) transform InputFLF into a set of lexical constraints Constraints
- (c) generate a sentence Sentence that obeys Constraints
- (d) transform Sentence's SLF into OutputFLF
- (e) check whether OutputFLF entails InputFLF

(f) if (e) is negative, repeat (c)-(e); else: done;
(g) output: Sentence, *translating* InputFLF

This procedure is essentially non-deterministic. In stage (b), as much information as possible is gained from the full semantic representation InputFLF, but – quintessentially – this logical form contains too little information to determine the structure of a sentence representing it. Because the semantic constants in FLF are taken from lexical SLF – the spell-out does not add or delete any conceptual content – the acquisition of information from FLF amounts to a line-up of lexical units that can or must contribute to the sentence, with some pre-established relations between them. Starting from this line-up, at stage (c) a sentence is constructed according to the generation procedure of chapter 1. The generation procedure tries to accommodate at most and at least the concepts of the line-up. If the procedure comes up with a sentence that meets these requirements, its FLF is computed. We assume that OutputFLF is at least as specific as InputFLF because InputFLF inevitably underspecifies the features of its translation. Therefore, at stage (e), OutputFLF is checked to entail InputFLF. The outcome of this check may be negative, for reasons beyond predictability but having to do with the fundamental incongruence between form and meaning. If so, the generation procedure is so little deterministic that it may come up with another sentence meeting the lexical and conceptual requirements. We assume that in principle it is impossible to add reliable information to this backtracking manoeuvre as to the source of the proven incompatibility. If that were possible, the information could have been added before, making the generation essentially deterministic. But in that case, InputFLF would contain full information with respect to sentence construal. This runs counter to every semanticist's experience: the track from form to meaning is one-way. Going from form to meaning and back is taking a roundabout, with obligatory change of vehicle. Yet, we will argue that the grammars for parsing and generation are essentially and characteristically identical, to wit, the joint of the syntax of chapter 1, the semantics of this chapter and the lexicon of chapter 3.

2.7.2 From logical form to lexical line up

Entailment is a relation between natural-language sentences. That relationship can be judged by language users: natural-language semantics is an empirical art when founded on entailment (cf. Chierchia and McConnell-Ginet 2000). At the end of the day, it is not the relation between formal representations, but the informal relation between sentences, that can and must be tested. To lead

entailment and reasoning back to natural-language processing, we must be able to generate from those logic representations that are exploited for reasoning and inference. We now present a non-deterministic procedure to generate Dutch sentences with a predefined, fully-specified formal meaning; of course, the procedure is grafted on the DELILAH parser and generator. The input to the procedure is an FLF index, as presented above. This index formula contains semantic information only. The output is a well-formed Dutch sentence with a full grammatical representation, again providing a formula in LF and its index FLF. The main characteristics of the procedure are:

- the input constraint is not biased towards the syntax or the lexicon;
- the generation procedure is non-deterministic, but finite;
- the result can be validated logically: input and output semantics are formulas in the same language.

Here, we describe a method to relate FLF to the lexicon and the (categorial) grammar by exploiting the semantic nets induced by it, exemplified in (351). These networks are shown to be able to steer lexical selection and grammatical unification. *L-coverage* (356) is applied to validate the result.

The generator is hypothesis-driven: it tries to construct a well-formed and meaningful phrase of a given category, with a complete parse in the form of a unified graph representing a complex symbol. The generation procedure is strictly meaning-driven: it operates without any structural preconditions, unlike the logical form-driven generators described in Carroll *et al.* (1999) and White and Baldridge (2003). In these 'realizers' – as in the famous translation system of Rosetta (1994) – the input to the generation procedure contains essential pieces of structural information relevant to the output. Here, we propose a purely semantic way of constraining the realization, as our input constraint completely abstracts from syntax. Generation proceeds by selecting appropriate phrases from the lexicon after inspecting an agenda and by testing their unification. The generation is successful if the hypothesis can be checked, no item is left at the agenda and some non-empty structure has been created.

Basically, the algorithm tries to find templates and tries to unify them according to an agenda which is set by an initial hypothesis and updated by applying combinatory categorial rules. The agenda consists of two parts: *given*, corresponding with complex symbols already adopted, and *to_find*, corresponding to structures still to be checked. A successful unification of complex symbols according to the agenda is the genuine result of the procedure.

An FLF offered to the generator is 'chunked' into nets. As an example, the set of clauses in the FLF of a Dutch sentence *Elke vrouw probeerde te slapen* 'Each woman tried to sleep' is given here.

(358) { state(e1+↓+some+[], *woman*), th1(e1+↓+some+[], x+↓+every+[]),
tense(e2+↑+some+[], *past*), event(e2+↑+some+[], *try*),
th2(e2+↑+some+[], x+↑+every+[]), property(y+↕+the+[e2]),
th3(e2+↑+some+[], y+↕+the+[e2]), event(e3+↑+some+[y], *sleep*),
th4(e3+↑+some+[y], x+↑+every+[]),
th5(y+↕+the+[e2], e3+↑+some+[y]) }

For each variable all the clauses in which it occurs as an argument are selected. Here is the resulting list of variable-induced nets over (358).

(359)

e1-net: <state(e1+↓+some+[], *woman*), th1(e1+↓+some+[], x+↓+every+[])>
e2-net: <event(e2+↑+some+[], *try*), th2(e2+↑+some+[], x+↑+every+[]),
th3(e2+↑+some+[], y+↕+the+[e2]), tense(e2+↑+some+[], *past*)>
e3-net: <event(e3+↑+some+[y], *sleep*), th4(e3+↑+some+[y], x+↑+every+[]),
th5(y+↕+the+[e2], e3+↑+some+[y])>
x-net: <th1(e1+↓+some+[], x+↓+every+[]), th4(e3+↑+some+[y],
x+↑+every+[]), th2(e2+↑+some+[], x+↑+every+[])>
y-net: <property(y+↕+the+[e2]), th3(e2+↑+some+[], y+↕+the+[e2]),
th5(y+↕+the+[e2], e3+↑+some+[y])>

Each of the nets P[X] in (359) is taken to be a sequence of simultaneous conditions on the semantics of some lexical phrase. Together, the nets determine the lexical space for a generation process. Per net, all the candidates in their lexical quality of complex symbols are selected. That being done, the generator tries to produce a grammatical construct in which each net is represented exactly once. The agenda for this process is derived from the contingent syntactic properties of the candidates.

The conceptual agenda raised by the FLF-network strictly limits the freedom of the generator. Even when the available lexical space is extended by allowing purely functional additions, infinite looping is excluded under the cancellation agenda. On backtracking, the generator will produce all and only the sentences that live within the lexical space constructed by the input network and are reachable by the grammar.

Yet, this generation procedure from FLF is essentially non-deterministic in at least two senses:

- the (structure of the) FLF does not fixate the structure of the sentence, by definition of FLF;
- the output-FLF may not match the input-FLF, according to a certain semantic standard.

Like LF, FLF underdetermines not just the syntax, *e.g.* the word order, of sentences generated in this way. Because of this 'inverse underspecification' – LF

and FLF are not the outcome of a derivation but the spell-out of an underspecified unification result – the generation procedure cannot fixate all the characteristics of the produced semantics in the logical space in advance or *on the fly*. There are two sources for this flaw:

- FLF may itself contain less specifications (aspect, mood, tense, …) than any verbalization would introduce;
- inspection of the input-FLF cannot predict which logical dependencies between variables are blocked or improved by following a certain construction mode for the sentence.

The first aspect of generative underspecification is evident: one cannot be sure that an FLF contains all the information that a full sentence will produce by default. Natural language is meant to be more expressive than any logical representation. Complex symbols may introduce additional meanings to those mentioned in the conceptual agenda, *e.g.* by default specifications like tense on finite verbs. The concepts in the input are a subset of those in the output. Moreover, the input FLF, when constructed by reasoning, may not specify semantic dependencies that are inherent in sentential construal, like intensional embedding.

The second incongruence between input-FLF and output-FLFs is due to the form-driven nature of sentence meaning. Whether or not a certain operator can scope over another depends partly, if not mainly, on its embedding. The generation process may isolate an operator on a strong or weak island, by choosing certain phrases or certain syntactical patterns for the respective nets. For example, a quantifier embedded in a nominal construction (*to make the promise that ….*) has fewer scope options than a quantifier embedded in a non-nominal, but conceptually equivalent construction (*to promise that …*). In the same vein, intensional domains are not completely predictable from inspecting an FLF. Generally, weak and strong islands of any sort are induced by syntax, and the syntax is underspecified, by definition and inevitably. Consequently, the generation procedure cannot be enriched with an additional agenda controlling possible scopal dependencies. Scope can be checked or compared only *post hoc*.

Since FLF – and in fact, every purely semantic logical form – contains too little information to fully determine the generation procedure, generating from logic is a non-deterministic trial, by necessity. The outcome of the process can or must be checked against the input constraint. It is important to realize, however, that the input constraint and the output FLF can differ in a limited number of ways only. For example, the output may contain concepts that are not present in the input. But this situation occurs only if these concepts are

introduced by default, when applying certain complex symbols and if they passed the restrictions on unification imposed by the semantic networks. The output is far from being in free variation with the input.
As was argued above, it is not wise to check for strict identity or equivalence of input- and output-FLFs. Taking into account the considerations given above with respect to the 'inverse underspecification', we propose that the normal check would be as in

(360) Accept S with OutputFLF as a generated translation of InputFLF into Dutch iff OutputFLF *l-covers* InputFLF.

Informally, this means that the generated sentence is at least as specific as the input, or that a model for OutputFLF is also a model for InputFLF, but not necessarily the other way round. Again, it should be noted that the conceptual difference between InputFLF and OutputFLF will be very limited, restricting lexical resources to those induced by the semantic nets of InputFLF.

2.7.3 From lexical line-up to sentence: intertwining agendas

Given the lexical constraint of the previous section – basically: a restriction on the lexicon – the generator can start producing sentences within this restriction. The engine for this procedure is the categorial grammar and the *constructional agenda* that can be derived from its types (see chapter 1). For the sake of efficiency, we may add to the procedure a *semantic agenda*: a list of concepts that must occur exactly once in the sentence to be produced. Again for reasons of efficiency, it makes sense to combine the constructional and the conceptual agendas. The semantic agenda does not provide constructional information, though. Therefore, the generation procedure is essentially nondeterministic, if not of the *trial-and-error* breed. Yet, there is no objection to exploiting the network information as much as possible. It is, for example, advisable to retain the network itself as part of the semantic agenda, and put it to use in the following way. Suppose the constructional agenda requires a phrase with an *np* as a head to be instantiated and suppose that this part of the agenda is dictated by a phrase representing network *e2-net.* Before selecting a phrase to fulfil this task, check which variable in the actual network is related to that *np* and determine for that variable which semantic constraints it induces. Use these constraints to select a candidate phrase from the restricted lexicon. In the example it is clear that if the desired *np* is supposed to deliver the agent for the *try*-instantiation, the *x-net* immediately tells

you that this agent entertains universal quantification: a candidate for the *np*-phrase had better comply with this semantics.
The architecture for this intertwining of agendas is an empirical matter. Instead of trying to capture idle wisdom in words, let us look stepwise at the construction of a sentence according to DELILAH's present state.

Suppose we have an FLF like (361) with networks (362):

(361)
```
{ agens(e^i, x+↑+every+[])
  event(e^i, sing)
  atplace(e^i, y+↑+some+[])
  state( s^j, car)
  theme_of( s^j, y+↑+some+[])
  state( s^k, man )
  theme_of( s^k, x+↓+every+[] }
```

(362)
```
x-net:   <agens(e^i, x+↑+every+[]),
          theme_of(s^k, x+↓+every+[])>
y-net:   <atplace(e^i, y+↑+some+[]),
          theme_of(s^j, y+↑+some+[])>
e^i-net: <agens(e^i, x+↑+every+[]),
          event(e^i, sing),
          atplace(e^i, y+↑+some+[])>
s^j-net: <state(s^j, car),
          theme_of(s^j, y+↑+some+[])>
s^k-net: <state(s^k, man) &
          theme_of(s^k, x+↓+every+[])>
```

The *x-net* does not contain a non-functional concept. The set of lexical items induced by that net, LEX(x-net), therefore consists only of quantificational elements representing the functional concept *every*, among which of course are determiners like *elke* 'every' and *alle* 'all'. They are stored in the *ad hoc* lexicon, referring to variable x. The same holds *mutatis mutandis* for LEX(y-net). LEX(e^i-net) is centered around the concept *sleep* and contains at least those lexical instances of this concept that introduce both events and agents; it does not contain the nominal versions of *sleep* as they will not meet these requirements. LEX(s^j-net) is centered around the concept *car* and selects those lexical entries of this concept that introduce at least a state. Similarly, LEX(s^k-net) contains at least the nominal instances of the concept *man*. All retrieved lexical items are indexed for the variable of their network.
Each LEX-net is constructed according to a particular selection algorithm scrutinizing the particulars of the network. These algorithms – the *selectors* – are at the heart of the procedure: they must be liberal enough to allow every lexical item that might be needed for the production, and restrictive enough

to keep the generation within the semantic borders of the input FLF. The construction of these algorithms is feasible: every network is finite and all small clauses in the networks are headed by standardized predicates from a finite set. Yet, the selectors are language specific and in need of experimental validation. They represent grammatical intelligence.
So, the dedicated lexicon for the generation procedure is the union of these five sets. This lexicon is enriched with two more sets:

- the set of all lexical items that are specifically requested by one or more of the entries in the union, in particular lexically-selected 'words' in multi-word expressions or constructions;
- some selection of functional elements that may be necessary to bridge the underspecification gap between logical form and full sentence, in particular auxiliaries and functional markers without operational content, like Dutch *er*.

These sets can be constructed pre-derivationally, or addressed during generation, 'on demand'. It is important to note that in both cases the resulting lexicon for the generation procedure is a severe restriction on the full lexicon. After the dedicated lexicon has been established, generation takes off in the very same way as described in section 1.9. On the basis of a categorial hypothesis – *let there be an S* – agendas are created, executed and checked until a grammatical string complying with the hypothesis arises. The lexical space for the full generation procedure is reduced to the union of the LEX(X-net) sets and the two sets mentioned above.

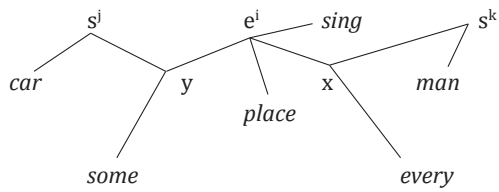
Of course, it makes sense to load the agenda in advance with the restriction that from each LEX(X-net) exactly one item is chosen for the ultimate production. Moreover, the relationship between the networks can be used as an agenda for the construction of the sentence. The relevant observation here is that for an FLF to be represented by a sentence its networks have to be connected in the following sense.

(363) FLF is *connected* iff every net in it contains at least one clause that also occurs in another net of the FLF.

Connectedness means, among other things, that the networks do not constitute a partition of FLF. If an FLF is not connected, the formula lacks semantic coherence. Sentences can be constructed only for coherent parts of the formula.
Now suppose that an FLF is connected, as (362) proves FLF (361) to be. The coherence of the FLF can be expressed in a connected graph, linking variables

and constants if they co-occur in at least one clause. Here is such a graph for (361)/ (362):

(364)



Every non-constant is a vertex supporting at least two edges. This graph can be used to steer the generation agenda*, e.g.* by starting at the densest vertex and following a depth-first strategy. In this way, the intrinsic combinatorial agenda according to which the generator operates and the semantic *desiderata* can be combined into one goal-directed procedure.

Restrictions like these on the generation procedure improve efficiency but do not disturb its essential non-determinism. In fact, finding the proper balance between lexical restrictions and the application of the grammatical agenda is an experimental rather than a theoretical matter. The generation procedure has its own syntactical backbone strong enough to accommodate additional lexical or semantic constraints. It seems beyond our grammatical reach, however, to predict the semantic fine structure of a sentence, in particular with respect to matters of scope and referential dependencies. Only a fully-generated sentence can therefore be checked for compatibility with the input.

2.7.4 Testing logical form by entailment

A sentence generated according to the protocols is bound to be grammatical and to comply with the lexical restrictions. There is no guarantee, however, that its Flat Logical Form entertains the intended semantic relationship with the input condition. In order to check whether it does, the derivationally produced SLF has to spell out as a family of FLFs. According to section 2.6.5, this family amounts to a conjunction of disjunctions. Since we assume that the input condition generally under-specifies the particulars of the sentence-to-be-produced, the produced Flat Logical Form is at least as specific as the input condition. Therefore, the intended semantic relation must be: *the produced logical form entails the input*: PrLF ⊰ IP. On the other hand, we do not want the produced FLF to contain content or information not covered by the input. The protocols of the previous sections are designed to prevent that. As a matter of fact: we take a sentence that is less specific than its semantic condition to be

a better instantiation of that condition than a sentence that is more specific. In both the following triples, the condition is more adequately and more correctly represented by the entailed than by the entailing 'translation'.

(365)	*condition*:	John hits Bill
	entailed/acceptable translation:	Bill was hit.
	entailing/over-specified:	Bill was hit by John with a stick
(366)	*condition:*	John did not hit Bill
	entailed/acceptable translation:	Bill was not hit by John with a stick
	entailing/over-specified:	Bill was not hit

Therefore, one could also require the produced form to be at most as specific as the input: *the input entails the produced logical form* or: IP ≺ PrLF. Consequently, we may want the two logical forms to be equivalent: IP ⊣⊨ PrLF. But the requirement of simple equivalence ignores the asymmetry of the directional argumentation for entailment: the input cannot be required to specify everything that a full sentence in a particular language brings along, and the sentence cannot be kept from coming with intrinsic essentials like tense and aspect which may be absent in IP, in particular when the input is not language-borne.
The analytic nature of FLF and the simplicity of its inference engine offer a solution to the problem. To see how, let us first be specific about the requirements of the produced logical form:

- no clause in PrLF introduces non-functional concepts absent in IP;
- every clause in IP is uniquely reflected in the PrLF – a bijection should exist between IP and a subset of PrLF;
- no clause in the PrLF is more specific than its reflection in IP: at clause level, the produced FLF is entailed *per clause*.

To meet these requirements simultaneously, PrLF minus the image of IP has to be submitted to severe restrictions, both with respect to the variables in the remnant clauses as well as with respect to the concepts. These restrictions are language-dependent, influenced for example by finite morphology. In Dutch, for example, PrLF cannot contain states or events outside the image of IP but it may contain tense specifications in that segment.
This yields the following definition of a successful generation procedure.

(367) Given semantic condition IP, PrLF is a correct representation of IP *iff*
(a) IP' is a conjunctive subset of PrLF such that for every clause φ in IP' there is exactly one clause ψ in IP and vice versa for which ψ ≺ φ
(b) PrLF - IP' contains only clauses that obey predefined semantic restrictions.

By the construal of Flat Logical Form, this constraint can be checked linearly. If the check is positive, the produced sentence is one of the kind we were after. If not, simple backtracking will produce another sentence.

By combining the connected graphs of type (364) and the type-driven agenda of the generator, a generation will almost inevitably produce sentences which comply with the semantic infrastructure expressed by the graph. In particular, one may expect that the input graph is an alphabetical variant of a subgraph of the one constructed on the output- FLF. This is a presupposition of condition (367). Scopal dependencies, however, cannot be arranged by manipulating the agenda. Scopal dependencies follow from the structure as a whole; they are determined under a top-down regime, whereas the constructive agendas operate piecewise and bottom-up. Scope is reconstructed post-derivationally, in the transition from SLF to FLF and ALF, rather than being constructed on-line. Its validity with respect to the input can only be tested and not predicted. Generally, clauses with independent variables entail clauses with referentially charged variables: if there is a woman that every man loves, it must be the case that every man loves one or other woman. Consequently, following (367), if a clause in the input-FLF specifies variable dependency, the output LF must specify an equivalent dependency. Again, scheme (354) can do the job.