3. LEXICON: the language's encyclopaedia and database

3.1 STORING KNOWLEDGE OF LANGUAGE

3.1.1 Lexicalism: atoms and molecules

The computational model of language presented here is knowledge-driven. It lives on the explication of grammatical combinatorics. The knowledge needed is both analytical as well as synthetic, both detailed and global, concrete and abstract, heading for the rules as well as for the exceptions. This knowledge has to be explicated and stored somewhere, and be readily accessible for parsing and generation. The explication and the storage had therefore better be efficient, with respect to operation as well as maintenance and manipulations like change and extension – learning, basically.

Accessibility of grammatical knowledge presupposes that the knowledge is retrievable on demand. And this requires a transparent storage policy. A language's immediate building blocks, the atoms of form and meaning, also known as *words*, *morphemes* or *phrases*, are the easiest addresses to handle, to retrieve and to order information. Let us call them word complexes, for the time being. There is little doubt that they are the basics of human awareness when it comes to language, as convincingly argued in Levelt (1989) and not really challenged since. These word complexes are also the best anchors for storing grammatical knowledge in a computational system. The reservoir of words presents itself as the immediate database for knowledge of language.

This point of view is quite common, but is no good unless we have an idea about what the building blocks of a language really are. This amounts to

the question at what level of language form and meaning meet. Clearly, pure forms are below that level, and texts are above it. This leaves us with morphemes, words, phrases and sentences as candidates. Sentences fail for the purpose of storing information, as they form an infinite set – information stored in an infinite database is hard to retrieve. As for morphemes, categorial grammarians in particular have argued, with some force, that the meaningful combinatorics of word formation and sentence formation are quintessentially equal (Moortgat 1988, Hoeksema 1984, and – to some extent – Wheeler 1981). This, however, is misguided, Aronoff (2008) claims: '... what happens inside lexemes is qualitatively different from what happens outside them'. He sticks with Chomsky's hypothesis on the need to lexicalize nominalizations: the semantics of morphological operations are idiosyncratic, and morphological combinatorics do not match syntactical operations at phrase level (Chomsky 1970, Rozwadowska 2006) – it must be admitted, though, that generative syntax is more abundant than the categorial algebras of Moortgat and Hoeksema. This *lexicalist hypothesis* accounts for the difference between word grammar and sentence grammar. Sentence grammar – the one that has to provide propositional interpretation – cannot steer word grammar in the following sense: if some properties of words or phrases are asked for, the sentence grammar lacks the expressive power to tell the word grammar how to provide or check these properties. We want the sentence grammar to be restrictive somehow, with less expressive power than unrestricted rewriting systems, for reasons of computational complexity. The word grammar, however, can easily be shown to operate in an unrestricted mode: it erases, it adds, it reshuffles and overrules, by necessity, in a way that we would never allow sentential combinatorics to operate. Of course we want to derive both a passive participle and its adjectival instances from some stem form of a transitive verb, but the information coming with the stem and feeding into these derivations has to be manipulated: the combinatorial properties of the participle and the derived adjectives differ essentially from the properties of other verb forms. To put it more bluntly, word grammar cannot be based on unification – it is based on destruction and reconstruction. This delicate process is what this chapter is about. Its results are complex symbols stored in a lexical database, available to the parser and the generator and disclosed by fast and precise search algorithms. These complex symbols are molecules to the syntactical and semantic chemistry of parsing and generation: they are self-contained, well-defined and unique clusters of properties and attributes relevant to sentence grammar. They are *signs* (Morrill 1994) in the sense that they combine syntactical and semantic information that is

conservatively exploited by the sentence grammar: it feeds into unification, but is not altered by serving as such.

The lexicon explicitly specifies all the combinatorial and semantic peculiarities of a language. If a certain verb form comes with a specialized meaning when combined with a certain preposition, there will be a complex symbol defining exactly this state of affairs. If that combination has several syntactic instantiations, there will be a complex symbol for each of these instances. If a certain preposition in combination with a certain type of noun gives rise to a specialized meaning on which the noun's determiner operates, there will be a complex symbol or a family of complex symbols holding that particular combination. If a certain combination has combinatorial restrictions not common to its kind, there will be a complex symbol that expresses these restrictions. The lexicon is constructive and exhaustive: it defines and delivers all phrasal molecules. At the same time, the lexicon itself is constructed, generated by a bunch of involved algorithms that account for inheritance and data economy. These algorithms, the word or phrasal grammar, distribute, add and change all kinds of grammatical specifications. They create graphs the unification of which is the final target of sentence grammar. Here is an example of such a graph: the first person singular main clause template of the separable particle-preposition-verb combination *laat+toe+tot* 'admit (someone) to (something)'; it is just one of the numerous templates representing the verb form *laat* 'make, do' in the lexicon. All values that are dependent on other values in the template and thus occur more than once are in bold face. Some mainly administrative features are left out.

(368) *lexical template for* laat ... toe *'admit'*

```
ID:A+B
HEAD:CONCEPT:admit
     PHON:laat
     SLF:admit
     SYNSEM:ETYPE:event
            FLEX:fin
            NUMBER:sing
            PERSON:1
            TENSEOP:at-pres
            VTYPE:bi_trans
PHON:C
PHONDATA:lijnop(laat,A+B,[arg(right(-1),0,D),arg(left(11),
         wh,E),arg(right(-2),6,F), arg(right(0),8,G)],C)
SLF:{{[H&(B+I)#J, K&(B+L)#M, N&(B+O)#P,
    Q@some^R^and(and(quant(R,some), admit~[R], event~[R],
    entails1(R,incr)), Q, entails(R,incr))&(A+B)#S],[],[]},
    and(and(and(theme_of~[S,J],agent_of~[S,P],
    goal_of~[S,M]), attime(S,T)), tense(S,pres))}
SYNSEM:CAT:s
       EXTTH:agent_of~[A+B,P]
       PREDTYPE:nonerg
       SUBQMODE:U
       TENSE:tensed
TYPE:s\0~[np^wh#B+O]/0~[pp^6#B+I,np^0#B+L,part^8#B+V]

ARG:   ID:B+I
       HEAD:PHON:tot
       PHON:F
       SLF:H
       SYNSEM:CAT:pp
              OBJ:indirobject_of(A+B)
              THETA:theme_of

ARG:   ID:B+L
       PHON:D
       SLF:K
       SYNSEM:CASE:obliq
              CAT:np
              OBJ:dirobject_of(A+B)
              THETA:goal_of

ARG:   ID:B+O
       PHON:E
       SLF:N
       SYNSEM:CASE:nom
              CAT:np
              FOCUS:focus
              NUMBER:sing
              OBJ:subject_of(A+B)
              PERSON:1
              QMODE:U
              SUBCAT:pron
              THETA:agent_of

ARG:   ID:B+V
       HEAD:PHON:toe
       PHON:G
       SYNSEM:CAT:part
```

The graph's representation is just for convenience: it might as well be represented by an unordered list of fully-specified paths, all headed by a node TOP (see section 3.3.1), or by a figure with labelled edges or vertices.

This graph is generated from two input data: a basic graph for intransitive particle verbs and some lemma-like specifications for the verbal complex *toe+laten+tot* 'admit', among which, the concept. The combinatorial category requires the subject to be left-dislocated and the particle to occur at the right-hand side of the verb. The head of the constituent is the verb form – 2nd person singular. The tense is specified as part of the overall Stored Logical Form. The subject argument is obligatory: its type is part of the category and its yet uninstantiated phonology ARG:PHON:E is specified in the value of PHONDATA, the label heading the linearization data. The concept it represents HEAD:CONCEPT:admit stems from the input and is integrated in the more extensive cluster of lambda terms under SLF. This ordering of lambda terms holds the full semantic architecture for the (sentential) constituent emanating from this graph. The particle does not contribute to the semantics: no semantic value for the particle *toe* would make it to the constituent level, as it has not been integrated in the value for SLF at top level. The same holds for the obligatory and selected preposition *tot* 'to'. Both the particle and the prepositional object as well as the subject are specified in the graph, as values of an ARG feature, and these specifications are open to unification with other graphs. All ARG labels have a unique ID value, for identification, linking them to the ID at top level, to be instantiated by unification, again. Under these labels, all constraints imposed on unification candidates can be specified. Above, the subject is required to be first person and a focusable pronoun. In the same vein, the subject could have been required to be animate, for example. This is a way to state that the subject of *toe+laten+tot* is animated by its occurring as such, not necessary by lexical specification: unification is not blocked by one-sided values.

It may be clear from this example that the differences between our lexical format and the practices in Head-Driven Phrase Structure Grammar (HPSG) are rather limited. They will be discussed in the next section. The base line is that in lexical entries – derived lemmas, templates, lexical graphs, complex symbols – like (368), all grammatical knowledge converges. It is the storage and the source of grammatical wisdom. With this ambition, the proper balance in the lexicon can be defined: that the lexical graphs be self-contained grammatical objects that do not need any other source of information or data in order to contribute to the formation and interpretation of sentences. This view implies that all inheritance relations are compiled into the singular lexical graphs: two graphs are lexically related if and only if they share some values, and there is no other relationship between the graphs of *sings* and

sing than there is between the graphs of *sings* and *walks*: they share certain paths. Moreover, the graphs contain whatever lexical-semantic features play a role in determining semantic structure, like the *qualia* and the patterns of type coercion discussed in Pustejovski (1993). Each lexical graph is an object in its own right. That mutual independence is an important precondition for the kind of fast retrieval for the sake of parsing and generation that we will introduce in section 3.5.

3.1.2 Delilah and HPSG

Head-Driven Phrase Structure Grammar, introduced in Pollard and Sag (1987), has become the most influential grammatical framework for computational linguistics; there were times when the framework seemed to be the shortcut for grammar-based computational linguistics. Sag (2003) has it spring off from an alternative view on generative grammar. This alternative to derivational generative grammar is constraint-based, not transformational, and strongly lexicalist. This last feature in particular requires the Delilah system to be compared to the HPSG approach, which underlies high-standard parsing systems like LINGO, Alpino and Verbmobil, all available on the internet. The Delilah system has been developed in the light of HPSG.
At the theory's home site http://hpsg.stanford.edu we find a neat summary of HPSG's main ideas. Here is the list, with each idea being shortly described (from the site) and compared to Delilah.

HPSG is a constraint-based, lexicalist approach to grammatical theory that seeks to model human languages as systems of constraints. Typed feature structures play a central role in this modelling. Some of the leading ideas of current work in HPSG are the following:

Strict Lexicalism
Word structure and phrase structure are governed by partly independent principles. Words and phrases are two kinds (subtypes) of sign.

In Delilah, words and phrases are 'created', organized, manipulated and combined by principles which are independent of the (rules of) syntax. Words and phrases are indistinguishable in the Delilah lexicon, however. All constraints are imposed and effected in the same manner, whether the constraint is of a phonological, syntactic or semantic nature.

Concrete, surface-oriented structures

'Abstract' structures (e.g. empty categories and functional projections) are avoided wherever possible, in favour of 'minimal' grammatical structures.

DELILAH's lexicon consists exclusively of phonologically non-null material. It may, however, introduce meaningful structures not explicitly licensed by syntactic units. These structures are not introduced apart from the 'real' lexicon, though.

Geometric prediction

The hierarchical organization of linguistic information plays a significant role in predicting the impossibility of certain kinds of linguistic phenomena.

DELILAH's lexicon is not hierarchically organized in the sense that some objects depend, or live, on others. DELILAH's signs are directed graphs, and the only hierarchy in the lexicon originates from this directionality. Grammatical processes are not governed by the hierarchy between lexical objects. The graph's geometry is not exploited for combinatorial or theoretical issues.

Locality of selection

According the theory of valence articulated in Pollard and Sag (1994), lexical heads select only for the synsem *objects (a kind of syntactico-semantic complex) of their complements, subjects, or specifiers. It follows that category selection, role assignment, case assignment, head agreement and semantic selection all obey a particular kind of locality determined by valence selection features. This is a kind of geometric prediction.*

In DELILAH, selection is local in the sense that all parameters have to be part of one single lexical data structure – a template – when selection is activated. Since information can only be added by a conservative form of unification, locality of relevant information is maintained through derivation. Selection is not restricted to particular objects, though.

A distinction among types of agreement

Agreement phenomena have been classified by Pollard and Sag (1994) as syntactic concord, anaphoric agreement, or pragmatic agreement. Their theory of indices predicts, inter alia, the absence of case agreement in anaphoric agreement. (...)

This distinction is not exploited in DELILAH. Concord is handled by local unification. Anaphors are resolved by additional post-unification mechanisms. Pragmatic agreement is not within DELILAH's scope.

Local encoding of unbounded dependencies

Filler-gap phenomena and other long-distance dependencies are treated not via grammatical transformations, but rather in terms of certain feature specifications that are present throughout the 'path' from filler to gap. This feature-based theory in essence predicts the existence of grammatical phenomena sensitive to such specifications, i.e. phenomena that occur only within the domain of unbounded dependency constructions.

In DELILAH too, unbounded dependencies are based on lexical specifications, namely in the lexical categories. Restrictions on the path are accounted for in the multi-modal combinatory categorial grammar that steers the unification.

Lexical cross-classification

Within HPSG, words are rich in information. Lexical information is not simply listed, however; rather it is organized in terms of multiple inheritance hierarchies and lexical rules that allow complex properties of words to be derived from the logic of the lexicon. Current research is developing extensions of hierarchical lexicons that allow lexical rules to be eliminated and linking patterns to be derived in a general fashion from semantic properties.

DELILAH's lexicon is created by a complex set of rules implementing weak forms of inheritance and generalization over lexical classes. The final lexicon, however, is flat, in that every entry is completely independently represented and unfolded. The hierarchical organization of the data structures themselves, and their graph-like format in particular, is exploited for dynamic classification in parsing and generation. The HPSG approach is certainly anchored in artificial intelligence (Sag 2003: 303 ff). The DELILAH lexicon does not embody any claim on organization of information beyond its retrieval mechanisms.

Hierarchical cross-classification of grammatical constructions

There are new proposals within HPSG to model constructions, as well as signs, in terms of feature structures. This allows constructions to be analyzed via multiple inheritance hierarchies. This in turn provides a way of modeling the fact that constructions cluster into groups with a 'family resemblance' that corresponds to a constraint on a common super-type. This strain of HPSG has thus coalesced with one conception of 'Construction Grammar'.

The DELILAH lexicon contains all instantiations of all constructions, salve coverage. All lexical templates are built in the same way, by a complex set of procedures. These rules respect common ground for the sake of efficiency, but there is no built-in hierarchy in the resulting lexicon. The family resemblance between templates is exclusively established by dynamic classification, but any resemblance can be retrieved that way. The morphological paradigm of a

verb, for example, is no different from the class of second person finite forms of transitive verbs with a right-hand subject. The way the lexicon is unfolded does not bear on its internal structure, which is plainly flat.

Obliqueness-based binding theory
Generalizations about constraints on the binding of referentially dependent elements are stated in terms of relative obliqueness (o-command), rather than configurational superiority (c-command).

The DELILAH binding algorithms use both obliqueness and configurational information. These operate post-derivationally on fully specified grammatical graphs.

Linearization theory
Current work in HPSG is exploring modes of serialization which are not based on the model of traditional phrase structure grammar (where sentences are word strings defined derivatively in terms of phrase structure). This has implications for the treatment of discontinuous constituency, allowing even the introduction of levels of linear syntactic organization that are to some extent dissociated from the combinatorial relationships among the items serialized.

DELILAH's multimodal categorial grammar is designed for dealing with discontinuity. In principle, it is capable of handling any form of discontinuous linearization that can be expressed by reference to syntactic categories. As explained in chapter 1, it is, nevertheless, quite restrictive and fairly rigid.

In general, HPSG focuses on the organization of the lexicon and the signs. It seeks restrictiveness and linguistic relevance in this domain. DELILAH has a more liberal attitude towards the structure of signs. It seeks restrictiveness in the rigid, multimodal combinatorial categorial grammar – categorial list grammar – introduced in chapter 1. The less constrained view on the lexicon is mainly motivated by the huge and yet unexplored variation in multiword expressions or extended lexical units (Sag *et al.* 2002, Poß and Van der Wouden 2005, Poß 2010). They appear in several of the following sections.

3.1.3 Words and worlds: a lexicon is not a dictionary

For computational linguistics with semantic ambitions, the most intriguing relation to be explored is the distinction between the lexicon and the encyclopaedia. For those who assume that knowledge of language must be absorbed from modeling and inspecting huge corpora, the distinction is hard. Language

is definitely not about itself. Texts feeding into automated lexical storages are about something other than language. Information derived from those texts is information about how the worlds are caught in phrases by people using a certain language. One may assume that this *is* semantics. In our view, it is not. Semantics is about the preconditions under which language can be used to capture the worlds. *What* is actually claimed, and how often, and by whom is beyond the language's architecture. The language system is neither defined nor affected by someone claiming something.

To derive a lexicon on the basis of texts acting as encyclopaedias, then, is doomed to yield an excerpt of those encyclopaedias. Though intriguing as a window on the use of language in a certain community, these excerpts cannot define the language, for the simple reason that what is going to be said or could be said but is yet unrevealed is as much language as what has actually been said. Language is the capacity of being meaningful, and the lexicon is the place where this potency is maintained and vitalized, not buried or mummified. In exactly this sense, the Semantic Web enterprise (Berners-Lee, Hendler and Lasilla 2001, Shadbolt, Hall and Berners-Lee 2006) differs essentially from natural-language semantics. Existing knowledge of all kinds is specified there as a defining characteristic of objects of all kinds. In natural-language lexicons, however, only the opportunity to refer can be sketched, not its 'extension' on any index.

The lexicon, as we envisage it, can and must contain all constraints and frames that help to guarantee a sentence's interpretability in whatever model. The lexicon does not necessarily contain factual or contingent properties of objects covered by a concept. Of course they can be added *ad libitum*, but every addition will constrain the general applicability of the lexicon, and improve its applicability within a certain model.

World knowledge – the encyclopaedia – has been widely invoked in computational linguistics to assure common-sense inference. In particular, it has been argued that natural-language processing systems should have access to information about named entities and normality in order to avoid recognizing or producing anomalies of the type

(369) The boy that a MIG painted green had been pregnant with almost every iron song before any donkey.

It is almost standard to notice, though, that this kind of sentence is a perfectly well- formed and interpretable sentence. Because of that well-formedness and interpretability, one may judge that my present world is not a model for that sentence, or accept it as a way of describing the present state of affairs.

Semantically, however, either of them is a fine result – one of the possible outcomes of an evaluation – and neither evaluation points to a problem of language. One may observe that her present interpretation of being pregnant is hard to comply with painted boys having that property or songs being iron. It is not the lexicon that has to be specific about these *qualia*: it cannot be so, unless it is restricted in advance to my little world. Note, however, that these *de re* specifications are essentially different from assuming that a predicate, when applied positively to an argument, imposes some attributes to the argument. The qualia imposed by combinatory syntax, should be either disjoined from all other lexical specifications or should not be checked under unification, *e.g.* by labeling them differently. Those selectional features alone can never be a reason to reject a sentence. Assuming graph unification, however, it is straightforward to have a predicate impose certain qualia on its arguments, no matter whether these qualia are realistic or not. But most certainly, a free language generator producing (369) is doing fine.

The lexicon is not the place where normality is defined. The lexicon does not tell us what the actual world looks like – it is not a genuine encyclopaedia. The lexicon is the place where virtual semantic space is created – the virtual semantic space we live by, to use a simple metaphor. The lexicon does not tell us how it is outside language. At best, it tells us something about language use and the human imagination – not about truth outside the gates of Eden. The lexicon does not tell us that pregnancy is reserved to female mammals.

For all kinds of purposes and applications, restrictions on the semantic space can be imposed on the lexicon, by brute force or by statistical modeling of normality. There are two basic problems for anomaly regimes, however, deserving attention. First, we have the finiteness problem. If sentence (370) is anomalous, each of the sentences in (371) is too. Yet, it seems impossible to sum up all equivalents to the relevant predicates.

(370) The male regretted being pregnant.

(371) The individual whose chromosome structure was definitely not equivalent to those of most individuals that may become pregnant regretted being so.
The regular procreator of her dearly beloved oldest daughter regretted being pregnant.
The male regretted being in a state in which till then only women had turned out to be.

Secondly, polarity can be used to declare or even define anomaly:

(372) Until now, almost no male has been known to be pregnant.

This sentence is a mere this-worldly fact, and it is certainly not more anomalous than *hot snow does not exist*. Consequently, judgments on anomaly are post-interpretational and, therefore, not part of any interpretative device proper.
If normality is not a genuine feature of the lexicon, knowledge of the world is not a feature of the lexicon either.

3.1.4 Lexical chemistry: cooking the graphs

In the preceding sections, the lexicon was depicted as a database, containing all the linguistically relevant information. It was claimed that the process of constructing this database is knowledge-driven: whatever there is to be known about individual lexical items has to be made explicit, stated and stored in a general, retrievable way. In combination, the requirements of explicitness and retrievability lead us to a distributed lexical format, in which no piece of information is hierarchically ordered above another. Phonological, morphological, syntactical and semantic information must be available simultaneously and must be randomly accessible. Though the data are not unordered (see section 3.5) they do not dwell in an a priori system. The lexicon consists of *complex symbols* and each complex symbol belongs to all classes of symbols defined by a feature value it carries at the end of its paths; the graph in (368) is as much related to other graphs with a path `SYNSEM:CAT:s` as it is related to other graphs with a path `HEAD:PHON:laat` or with a path `HEAD:SLF:admit`. Consequently, the lexicon consists of some enumeration of complex symbols, each of which is a completely independent object. The system for enumeration is random in principle, and can be chosen on purely pragmatic grounds (see sections 3.4 and 3.5).
As a matter of fact, the same holds for the way in which the lexicon is generated. The DELILAH lexicon is produced by an extremely complicated set of rules that combine 'classical' lemma data with canonical graphs and rules defining the lemma's offspring. To put it bluntly: all the graphs for all the morphological forms of the verb *laten* 'let' are created by applying the rule machinery to a particular lemma data structure and one or more 'pre-defined' canonical graphs, *e.g.* for intransitive verbs. Whether two intransitive verbal paradigms live on the same canonical graph is only a matter of efficiency, not of principle. At the end of the day, the lexicon is as flat as a pancake.
This strategy brings about the option that *inheritance* of features and values is no longer grammatically relevant. Inheritance is a major lexical concept in HPSG (*cf.* Sag 2003; ch. 15). It regulates the spread of features and values throughout the lexicon, and is part of the economy and architecture of the

grammar. If the lexicon is completely spelled out, however, in each and every painful detail, the lexicon's history or the way it came into being does not carry additional information. Even the classical notion of a lemma becomes obsolete at the level of the lexical database. Whether or not this history is subject to computational, lexicological or grammatical principles is for the record, but not relevant for parsing or generation. It certainly makes sense to scrutinize the process of lexicon formation in order to find regularities there. However, this creates an expert system for lexicography rather than a module of formal grammar. The semantics of constructions, in our view, renders any hard-boiled principalism redundant. The enormous variety of ill-known collocational effects overshadows, for the time being, the regimenting effect of pre-established inheritance hierarchies. As a matter of fact, in many constructions hardly anything inherits according to a grammatical pattern. For example, in the Dutch *way*-construction an intransitive verb – any unaccusative intransitive verb – combines with a reflexive pronoun, a *way*-type noun phrase and directional pp to create a causative *move* event where the meaning of the intransitive verbal head only occurs as the cause, and neither the *way* NP – lexically fixed – nor the reflexive – a dependent anaphor – adds to the meaning (Poß 2010). Similar observations can be made about a variant of the construction without *way*. The members of the construction are underlined.

(373) Geen enkele bankier had zich een weg naar de Raad van Bestuur kunnen golfen
No banker had himself a way to the Board of Directors been-able play-golf
'No banker could have succeeded in moving into the Board of directors by playing golf'

(374) Geen enkele bankier heeft zich de Raad van Bestuur weten binnen te golfen
No banker has himself the Board of Directors been-able inside to play-golf
'No banker was able to move into the Board of Directors by playing golf'

The constructions themselves are rare in any corpus, but the classes of constructions are manifold (Grégoire 2010). Their variety and lexical construal are discussed in section 3.4.4. The point here is that sound opportunism in cooking graphs appears to be called for, rather than esoteric principles of design. The main principle seems to be that for each construction, only one lexical element needs an additional entry in the lexicon. This element can be called the *head* of the construction: it pays the price and becomes ambiguous. The other elements of the construction are simply selected, in whatever form they occur elsewhere.

3.2 MODES OF LEXICAL KNOWLEDGE

The lexicon stores knowledge of language – all knowledge needed for processing. As such, the lexicon stores knowledge in all classical domains of grammar: phonology morphology, syntax, semantics. Moreover, it may contain relevant information on phonetics and information structure, and even pragmatics – though we consider pragmatic properties of sentences to be beyond computability. Furthermore, lexicons can be used as stores for contexts – remarkable or frequent phrases other than constructions – as is done in the better-written lexicons, or information on means of use. There is a natural boundary to the storage: relevance for processing. This keeps the encyclopaedia out, for example. It allows for ontological hierarchies to be adopted by the lexicon to the extent that they play a role in an application, or are addressed in inference.

3.2.1 Phonological form: the one-dimensional grammar

In its present state, the DELILAH lexicon contains little phonological information. Of course, a fully-fledged lexicon must contain information as to the phonological structure of words and phrases. This information is widely available, but sound is hard to process on a general platform. Phonology is the lexicon's white spot, but its absence is not dramatic for DELILAH's modeling purposes. As a consequence, the elementary encoding in our system is orthographical.

This being the case, however, phonological form is built by templates linearizing lexical units. It is the surface realization of the syntactic structure. Every template contains information on the linear ordering of its lexical components. The linearization is not completely arranged by unification: rather, unification instantiates the variables in a program that is executed post-derivationally. Below are the phonologically relevant lines from (368).

(375) *phonological network in template of* laat ... toe '*admit*'

```
ID:A+B
HEAD:PHON:laat
PHON:C
PHONDATA:lijnop(laat,A+B,[arg(right(-1),0,D),
          arg(left(11),wh,E),arg(right(-2),6,F),
          arg(right(0),8,G)],C)
TYPE:s\0~[np^wh#B+O]/0~[pp^6#B+I,np^0#B+L,part^8#B+V]
...
ARG: ( ID:B+I
       HEAD:PHON:tot
       PHON:F
            ... )
ARG: ( ID:B+L
       PHON:D
            ... )
ARG: ( ID:B+O
       PHON:E
            ... )
ARG: ( ID:B+V
       HEAD:PHON:toe
       PHON:G
            ... )
```

The feature PHONDATA is valuated by a formula `lijnop/4`. The predicate operates on the orthographical representation of the phrase's head ('laat'), the template's index and a list of specifications on the four arguments, and returns its value ('C') to the attribute PHON. The specifications of the arguments provide the relevant data: the mode of composition for each of the arguments, its own 'phonology' – yet uninstantiated – and an indication of at which side of the head and at which relative distance to the head the argument occurs. All this information on the linearization of arguments is derived from the TYPE value when the template is constructed.

Even though the phonological value of an argument's head can be specified as part of the selective construction – as is the case twice in (375), the argument's phonological form itself can be complex. When instantiated, an argument comes with its own PHONDATA, to be executed in order to establish the argument's phonological form. Because of this recursive aspect and because in the lexicon, the patterns of discontinuity are largely unpredictable, lin-

earization cannot catch up with unification in an agenda-driven generation procedure. Therefore, `lijnop/4` can only be executed when all relevant linearization data are available: after the derivation has been completed. The procedure itself is described in chapter 1 as the operation `makestring/3`.

As a side effect, the independence of this linearization procedure offers an opportunity to check the coherence between the recognizing and the generating grammars. In an ideal world, the linearization result should be equal to the input string. If not, the linearization procedure must be assumed to have bugs.

3.2.2 Morphology: the combinatoric guide

The lexicon is word-based, rather than morpheme-based. This is a choice of system. DELILAH does not (yet) entertain a morphology, a word grammar. It specifies, though, systematic morphological variation in order to produce correct labels for complex symbols. These forms are produced by a set of rules operating on the orthography of a 'main' word and producing all the forms that can be derived. For verbs, for example, all finite and infinite forms, including adjectivally and adverbially used ones, are generated. Each of them is represented orthographically and marked for its role in the verbal paradigm: its templates will be generated on the basis of that role.

For the verb *verbranden*, 'to burn' the following list of forms will be produced automatically – not every form, though, is acceptable to every taxpayer.

(376) verbrand: *1person+singular+pres / imperative+singular / 2person+singular+pres+postverbalsubject / perfective participle / passive participle / adjective*
verbrandt: *3p+sg+pres / 2p+sg+pres+preverbalsubject / imp+pl*
verbranden: *pl+pres / infinitive / noun*
verbrandde: *sg+past*
verbrandden: *pl+past*
verbrandend: *pres part*
verbrandende: *adjective / noun*
verbrandenden: *noun*
verbrande: *adjective / noun*

For each of its labels, each form can be associated with multiple templates, differing from each other in combinatoric properties. They happen to share the concept *burn*, however. Consequently, a mono-concept lemma like *verbranden* may end up with sixty or eighty different templates in its transitive

branch. In its unergative reading, the verb will produce a similar number of templates. In principle, all these templates are different objects. Accidental similarities caused by situational convergence of different rules are filtered. As a consequence, every template is a unique attribute-value matrix (avm). Lexical redundancy is accounted for by the fact that across the lexicon all verb forms with a certain label are built into complex symbols by a single rule cluster. Instead of defining, *e.g.*, the affix *-t* in isolation as a present tense morpheme, the lexicon will treat all verb forms showing this affix with the same '2-and-3p-sg-pres' module, just as a different module accounts for all participles.

Other categories with morphological variation are instantiated in a comparable way. The rules producing the forms are complex but not deep. They had better be fed by phonological rather than by orthographical input – that is an option. The rules producing the templates of the distinct forms, on the other hand, carry and reflect much of the syntactic subtleties of Dutch. In particular, all variation in order – including systematic inversion – is encoded by the lexical rules in all kinds of attributes, like TYPE, PHONDATA, and ARG:SYNSEM, and as such is distributed over a template. The lexical rules embody the generalization, up to and including exceptions and constructional particularities. In a finite way, they must express all that can be known about the concatenation of forms in a particular language. Moreover, these rules express that knowledge in a very explicit way. Their manipulation of templates – they are real transformations (see section 3.4) – is complex but transparent and reconstructable, and every piece of knowledge is returned as a particular value to an attribute.

In this sense, the morphological paradigms represent the backbone of the combinatoric engine; they index the syntax.

3.2.3 Syntax: the unification agenda

3.2.3.1 Categories as data structures

Strictly speaking, DELILAH's categories encode three types of information:

(377) (a) for each phrase or constituent, its proper parts
(b) for each part, its position relative to the head
(c) for each part, the conditions under which its category can be composed

The way in which this information is derived and used is the subject of chapter 1. Here we consider only how categorial information appears in the lexical template.
The rules of lexical construal associate every complex symbol with exactly one category. The category of a template is completely determined by other attributes. This information is retrieved on construal and assembled in the data structure that becomes the value of TYPE:

(378) (a) Head\ LFlag~[TopLeft^Mode_i,..] / RFlag~[TopRight^Mode_j, ..]
(b) Head\ 0~[TopLeft^Mode_i, ..] / 0~[TopRight^Mode_j, ..]

Each of the two lists is finite, with a low cardinality, and may be empty. The values for Head, TopLeft and TopRight come from a restrictive set of categorial literals, among which are *s*, *np* and *vp*. The values for Mode are indices, again, from a restricted set. The sets of categorial literals and of modes are disjoint. LFlag and RFlag are dynamic indices, defined by the rules of grammar, but in the lexicon their value is always 0; (378)b reflects the lexical state of a category.

As a matter of fact, the composition modes Mode_i referred to in (377)c have little impact on the template's structure. The modes occur as part of the TYPE value where they are introduced as indices on argument types. They are addressed by the rules of syntax – both in parsing and generation – but are neutral with respect to unification; they are imposed by the constituent but not specified in the argument's sign. Consequently, they are not used as constraints on unification. Their role is merely to regulate the sequence of unifications that make up a sentence's analysis.

The constituent structure itself is reflected in the structure of the template. For each proper part, the template specifies a sub-graph as a value of an ARG(x) attribute where x is a unique and identifying index – this index is represented only when convenient. The sub-graph identifies the constraints imposed on the argument phrases by the mother construction, among which is a specification of the argument's type occurring as a literal in the matrix category. The relevant part of template (368) is represented below, with the complex indices on the ARGs, the mode attributes ('flags') and the direction/distance attribute indicated; the latter were left out of the representation in (368).

(379) *syntactical network in template* laat ... toe '*admit*'

```
ID:A+B
...
TYPE:s\0~[np^wh#B+O]/0~[pp^6#B+I,np^0#B+L,part^8#B+V]
...
ARG: [ ID:B+I
       SYNSEM:[... CAT:pp ... DIR:right(1)...FLAG:6] ... ]

ARG: [ ID:B+L
       SYNSEM:[.. CAT:np .. CASE:obliq .. DIR:right(5) .. FLAG:0] ]

ARG: [ ID:B+O
       SYNSEM:[... CAT:np ... DIR:left(11) ...FLAG:wh] ... ]

ARG: [ ID:B+V
       SYNSEM:[... CAT:part ...DIR:right(10)...FLAG:8] ... ]
```

These argument sub-graphs are the proper target of unification, to the extent that the unification of two templates is defined as an essentially antisymmetric operation.

(380) Two templates T_1 and T_2 unify *iff* there is a sub-template ARG(x):T_3 of T_1, and T_2 and T_3 are compatible.

Whether or not two templates are selected for unification is decided by rules of grammar which are also antisymmetric by nature. Thus, unification is the main process of natural language grammar, according to Kayne (1994).

3.2.3.2 Features and values

If we assume that unification of templates is basically antisymmetric, an argument's *attribute-value matrix* in a lexical template generally specifies two regimes: it states which feature-value pairs the mother construction imposes on the argument and it states which values the argument, when instantiated, is supposed to deliver to the construction values. All and only specifications to these effects are needed in the sub-avm. The nature of the value indicates its function: if it is a variable, the feature is delivered, and if it is a constant, the feature is imposed. Delivered values will also occur outside the sub-avm. As an example, take the lexical template for the simple intransitive *werkt* 'works' with just one (subject) argument specified. In that sub-avm, every feature is

indicated for being imposed (↓) or delivering (↑); a few redundant bookkeeping features have been left out.

(381) *lexical template of* werkt *'works' with mood of features specified*

```
ID:A+B
HEAD:CONCEPT:work
     PHON:werkt
     SLF:work
     SYNSEM:ETYPE:event
            FLEX:fin
            NUMBER:sing
            PERSON:3
            TENSEOP:at-pres
            VTYPE:nonacc
PHON:C
PHONDATA:lijnop(werkt,A+B,[arg(right(-10),0,D)],C)
SLF:{{[E&(B+F)#G,H@some^I^and(and(quant(I,some),work~[I],
    event~[I], entails1(I,incr)), H,
    entails(I,incr))&(A+B)#J],[], []},
    and(and(agent_of~[J,G], attime(J,K)), tense(J,pres))}
SYNSEM:CAT:s
       EXTTH:agent_of~[A+B,G]
       PREDTYPE:nonerg
       SUBQMODE:L
       TENSE:tensed
TYPE:s\0~[]/0~[np^0#B+F]
ARG:ID:B+F
    PHON:D ↑
    SLF:E ↑
    SYNSEM:CASE:nom ↓
           CAT:np ↓
           NUMBER:sing ↓
           OBJ:subject_of(A+B) ↓
           PERSON:3 ↓
           QMODE:L ↑
           THETA:agent_of ↓
```

The interaction of imposed and delivered features indicates the minimum and maximum of the class of features that have to be specified in the lexicon. A template does not need more specification than what can be imposed on it, and a template must not be less specific than what it is supposed to deliver. As an example: for an NP-template to be unified with (381) it does not need to be specified for the imposed features, but it must come with values for the features to be delivered. As a consequence, an NP-template needs to have only phonological content (PHON) and underspecified semantics (SLF) to qualify as an argument.

Knowledge of syntax amounts to specifying agreement in a proper balance of imposed and delivering features. Moreover, it assumes an antisymmetrical relationship between the construction itself – imposing values – and the constitutive parts – delivering values. The lexicon is phrasal, and organized by constructions.

3.2.4 Semantics: ultimate knowledge of language

3.2.4.1 Use of Concepts

Talking about the meaning of words, no *dictum* is more relevant than Frege's famous arithmetical statement:

(382) Nur im Zusammenhange eines Satzes bedeuten die Wörter etwas
Only in the frame of a sentence words mean something
(G. Frege. *Grundlagen der Arithmetik* (§ 62))

The proper translation may lead us into debates on in- en extensionality beyond the scope of this work. The quintessence, however, is that the denotation of words and phrases lives on the propositions in which they are framed, and not the other way round. All there is to the meaning of words in isolation, *e.g.* in a lexicon, is abstract and non-derivable. In a Carnapian approach to intensionality, words in isolation at best represent a function that delivers a referent when the word is *used* (in the framework of a sentence) to refer in a particular world. Unfortunately, human beings are excellent in recognizing reference and in converging on reference, but extremely poor in checking, not to speak of computing the Carnapian function underlying that referentiality. That is, we do not have the slightest problem in understanding your sentence

(383) That man is drinking a beer

even without checking what you mean by *man, drink* or *beer* and without having to agree with you on these intensionals. In fact, we understand your sentence, even if we turn out to disagree strongly on any concept used. People hardly ever check each other's concepts. And if they do, there will be a debate or a miscommunication.

Intensional functions are hard to define, for two reasons: we need other undefined intensional functions, and we do not know whether our function is correct – if we had a criterion here – and what it covers. Lexicographers tend to

compensate for that by shifting meaning from the function definition to the labels of relations between undefined concepts: *synonymy, antinomy*, *hyperonomy* and so on. The problem is, however, that these relations are extensional. A *chair* may be a piece of *furniture* in many situations but it is not so by definition – there is no intrinsic linguistic relationship between the two concepts, just as there is no intrinsic linguistic relationship between the concepts *whale* and *mammal, drink* and *eat*, *love* and *hate*, *city* and *capital*; the only relationship between the pairs is that in your particular world some phenomena may be both or neither or just one of them. Or, to put it in other words, the junctions in (384) are neither ungrammatical nor un-interpretable; but we must assume that the brain behind the sentences does not take roses to be flowers.

(384) John brought her roses and flowers.
John will bring her roses or flowers.

Neither is the next sentence false by definition:

(385) This rose is not a flower.

Yet, most lexicons will state that a rose is a flower – and there is a certain tendency among the living ones to see it that way. Generally, what is a rose will be referred to as a flower, but that is a statement of extension. It belongs in the encyclopaedia of the normal world conceived by a headmaster, but it does not express knowledge of language beyond its truthful use in the accidental world we live in.

Even though we may assume that anyone using *rose* or *love* applies some intensional function, there is no good reason to assume that we can *define* that function intensionally, as shared knowledge of language users: there are quite a few nouns we use felicitously without knowing their encyclopaedic status. Categorizing them conceptually would not express knowledge of language, not even knowledge of language use, but knowledge of this-wordly normality.

The intensional gap we can observe in language use may even be considered to be the engine of reference. When we say something in a certain situation, we refer to a state of affairs that has never been referred to before. We do so by using words from a finite set. If their intension were to be fixed, we might miss the novelty of the concept. This, of course, is the message of Lakoff and Johnson (1980): meaning shifts along cognitive patterns like *love is war*. The shift is not random, but its peculiarities are unpredictable.

Frege's statement (382) raises the question how words can get meaning amidst a complex network of interdependent phrases. The answer here is that structure conveys the message, and that structure is induced by functional elements. The structural elements in (383) are *that, is* and *a*. They are not even concepts: their meaning is fixed, but far from trivial and not easy to reconstruct. Yet, for functional terms, a lexical meaning can be established, as Montague (1972) showed.

For all other terms, however, all that can be said is that they represent some concept. This concept is beyond rigid formalization. This is not meant to disqualify heroic approaches to explain and capture lexical variation like Mel'čuk's *explanatory combinatorial lexicon* (*e.g.* Mel'čuk and Zholkovsky 1984), Sowa's conceptual *graphs* (Sowa 1984), Pustejowski's *type coercion* (Pustejowski 1993), Gärdenfors' *conceptual spaces* (Gärdenfors 2000) and many other efforts by semanticists, lexicologists and lexicographers through the ages. The point is that neither the properties of concepts *an sich* nor the relations between concepts are computable in any interesting sense. The source of this intractability is that the semantic properties of words or phrases are undecidable. For every non-trivial property P that is claimed to be semantically relevant to a word W, each of the following statements is true:

- we cannot tell whether P is relevant to W in all (relevant) contexts in which W is used meaningfully;
- we cannot tell whether there are other properties Q that might be semantically relevant to W in some context in which W is meaningfully used;
- we cannot tell whether such properties P and Q are independent;
- we cannot tell whether P is a characteristic property of W.

And '*we cannot tell*' means that there is no empirical practice to validate P, in exactly the sense in which Chierchia and McConnell-Ginet (2000: ch. 1) refer to the *empirical domain of semantics*: converging judgements of language users. We may gather converging judgements on lexical meaning in specified contexts. But an old observation is that any noun N that is claimed to be characterized by a property P may be used meaningfully in a sentence which amounts to *Some N is not P*. Therefore, even a converging judgement on P for N in a context neither limits nor characterizes N. What we can do, though, is measure its use and qualify its 'normality' or default-value based on this. Even for this purpose, however, we may run into circular traffic: at a certain moment some semantic grounding must be supplied, of the same nature that we are after. In this respect, it is wise to realize that vagueness, meaning transition and metaphori-

cal meaning is not exceptional; rather, it is the rule. As for vagueness, it must be stressed that not only predicates for which some metric exists are vague, but all predicates are: verbs, nouns, adverbs, and adjectives. The paradoxes discussed and analyzed in Van Deemter (2009) are perfectly general.

As a matter of fact, Chaffin and Herrmann (1988) argue correctly that the sets of relations with which conceptual networks can be built themselves require independent empirical justification – that is different from empirical validation, which was denied to lexical semantics mentioned earlier. We do not believe that it is the computational linguists' task to try and define these networks, although one may exploit computational tools to describe the way they are used. For all sorts of applications in specific domains or for specific purposes, the elementary concept may be substituted by more elaborated objects, like ontologies or lemmas from an encyclopaedia. Alternatively, concepts themselves can be seen as a molecular structure of semantic atoms, in the spirit of Wierzbicka's *Natural Semantic Metalanguage* (*e.g.* Goddard 2002). Such an enterprise is also beyond computational modelling of natural language. It amounts to implementing a theory of lexical meaning. In the same vein, Ebeling's (1978) complex theory of signs and semantic features may appeal to structuralist linguists, but his following dictum may prevent computational lexicographers from trying to implement it with finite resources: *even the most subtle semantic aspect can be described as a feature of something* (Ebeling 1978: 109).

The *concept*-field in DELILAH's lexicon is, as a consequence of this reflection, valuated with a link, a label or a hook, rather than by content. The lexicon is semantically and conceptually open.
Note, however, that synonymy is a decision not on concepts, but on language. Two phrases may mean the same, even when that meaning is not trivially given. This *intensional synonymy* is rather the exception, as it seems. At least, it is not wise to mix ontologies and meaning.

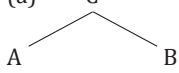
3.2.4.2 Types and lambdas

The *Empirical Domain of Semantics*, which Chierchia and McConnell-Ginet (2000) refer to, comprises those semantic properties that relate to propositions and, more generally, the way in which words and phrases combine into verificational statements. It is only natural that the semantic categories in this domain are closely connected to syntax. We have to assume that shared interpretations – another way to introduce empiricism – are induced by overt and covert syntax, to the extent that the judgements converge: if interpretations

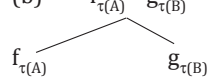
are stable and shared, they cannot be accidents. Compositionality is the name of the game: the meaning of a phrase is a function of its construction and the meaning of its parts. In order to model compositionality properly, however, the construction of a phrase, the semantic contributions of its parts and the way meanings combine must be made explicit. The ingredients for the compositional bakery are stored and labelled in the lexicon: categories, types and functions or, better, types relating categories to functions. If they have not been lexically declared, they will not appear anywhere.

Type theory, when it is applied to natural-language analysis, provides exactly the coherence of form and meaning that underlies the empirical dimension of natural- language semantics. Types index functions, and functions we need in order to construct the semantic network prompted by a sentence. The basic pattern is given below. If two syntactical units of categories A and B combine and produce a unit of category C, then C *means* the composition of two functions $f_{\tau(A)}$ and $g_{\tau(B)}$. Here t(X) is the type corresponding to X and composition passes into application if the type of the secondary function in the composition is the domain of the primary function: $f_{\tau(A)} \circ g_{\tau(B)} := f_{\tau(A)} (g_{\tau(B)})$ iff $\tau(A) = <\tau(B), \sigma>$. Moreover, given the types of A and B, only one of the two compositions can be valid: $f_{\tau(A)} \circ g_{\tau(B)}$ is defined iff $\tau(B) = <\alpha,\beta>$ and $\tau(A) = <\beta,\sigma>$ and $\alpha \neq \sigma$. Thus, the types maintain – if not embody – the fundamental antisymmetry of syntax (Kayne 1994); see also chapter 1.

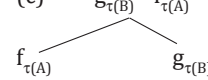
(386) (a)



(b)



(c)



Every template in the lexicon containing a combinatorial category comes with a semantic type, but the relation between categories and types is not functional: categories are not exclusive for types, nor types for categories. Therefore, the type of a phrase is defined and decided in the lexicon. A phrase is of a certain type only if the phrases' template gives rise to that type.

Yet, semantic typing in the DELILAH lexicon is implicit rather than explicit. Two reasons justify this obscurantism:

- not every syntactic argument is reflected in the meaning of its phrase;
- the syntax is rigid (see chapter 1), but the frozen syntactical combinatorics do not necessarily reflect the associated composition.

The first argument for implicit typing relates to the abundance of phrases the syntax of which does not reflect their semantic structure. To put it clearly: the

kick in *to kick the bucket* heads a transitive construction, but the corresponding function is not applied to the meaning of *the bucket*. Explicit typing would have to be overruled explicitly, which makes no sense.

The second argument is of a systematic nature. Functions are modelled by sets, as a function from A-objects to B-objects is a homomorphism from set A into set B. So, functions themselves form sets, since they can be arguments to functions typed otherwise. From algebra we know that each member of a set can be identified by its unique ultra-filter: the set of subsets containing that particular individual. That ultra-filter is a function again, of a predictable type: if the individual is of type α, its ultra-filter on the domain of objects of type α is of type <<αt>t>, for *t* typing the boolean values. In the Lambek-calculus of type-logical grammar (Moortgat 1988), this is an instance of a general theorem called *type raising* or *type lifting*.

For semantic purposes, but not for syntactic combinatorics, it can be useful to exploit the ultra-filter, rather than the original function. Montague (1972) exploited higher-order functions with flexible combinatorics to deal with scope variation. In the DELILAH grammar, the higher order functions are exploited to prepare constituent meanings for the scoping algorithm (see chapter 2). Nominal objects in particular are syntactically 'eaten' by their non-nominal licensers, like verbs, but semantically the objects scope over the licensor's meaning in order to allow for scope variation between its arguments. More bluntly, a subject is a syntactic argument to its verb, but semantically the division of labour is reversed: the subject is the primary type in a composition with the verb. This incongruency is deep and essential: an eventive verb may assign a theta-role to its subject, syntactically, but the subject's algebra determines whether (or not) some individual bears the relation expressed by the theta-role to that event: *no woman* holds the agent position in (387) but no woman was the agent of song singing if the sentence is true.

(387) No woman sang a song

In order to solve the paradox, generative syntactic theory assumes that the subject phrase does not carry the theta role, but heads a *chain* of positions of which the theta role's position is the tail. Assuming the lack of correspondence between syntactical and semantic structure to be real, nevertheless, the normal categorization and typing of a transitive verb and its arguments runs as follows.

(388) *syntactic categories*

finite transitive verb:	s\np/np	or	t\e/e
subject:	np	or	e
object:	np	or	e

semantic types

transitive verb:	<e<et>>	λ-term: $\lambda x_e.\lambda y_e.R_{eet}(x_e,y_e)$
subject:	<<et>t>	λ-term: $\lambda P_{et}.\ \wp_{ett}(P_{et})$
object:	<<et>t>	λ-term: $\lambda P_{et}.\ \mathscr{R}_{ett}(P_{et})$

Clearly, the arguments' types are *lifted* in the sense explained above, in order to allow for them to be composed with the verb. Lifting here is not deductive and procedural, as in standard categorial grammar (*e.g.* Moortgat 1988), but lexical and declarative and, thus, not recursive.

In our system, the higher-order functions are in the store of the lower function, and the arguments are in the store of the head, as explained in chapter 2. Consequently, the lexical template(s) of a construction's head specify

- its syntactic arguments, in a path with label `ARG(X+Y+Z):...`
- their syntactic categories, in the path `ARG(X+Y+Z):SYNSEM:CAT:...`
- their λ-term (generally as a variable), in the path `ARG(X+Y+Z):SLF:...`
- for every argument's λ-term:
    * its position in the stores at Stored Logical Form, *i.e.* a position in the value in the path `SLF:{{[...] ...}, ...}`
    * the variable in the body of SLF to which the λ-term is converted.

Below are the relevant lines from template (368).

(389) *syntactic-semantic network of template* laat ... toe *'admit'*

```
ID:A+B
...
SLF:{{[H&(B+I)#J, K&(B+L)#M, N&(B+O)#P,
    Q@some^R^and(and(quant(R,some),admit~[R],event~[R],
    entails1(R,incr)),Q,entails(R,incr))&(A+B)#S],[],[]},
    and(and(and(theme_of~[S,J],agent_of~[S,P],goal_of~[S,M]),
    attime(S,T)),tense(S,pres))}
...
TYPE:s\0~[np^wh#B+O]/0~[pp^6#B+I,np^0#B+L,part^8#B+V]

ARG: [ ID:B+I
       SLF:H
       SYNSEM:CAT:pp
       ... ]

ARG: [ ID:B+L
       SLF:K
       SYNSEM:CAT:np
       ... ]

ARG: [ ID:B+O
       SLF:N
       SYNSEM:CAT:np
       ... ]

ARG: [ ID:B+V
       HEAD:PHON:toe
       SLF:nosem
       SYNSEM:CAT:part ]
```

The relevant chunks of knowledge are in bold. The top SLF contains the following information. The λ-term that interprets the agentive argument internally converts to – or: is bound to bind – the variable P in the body. The λ-term that interprets the thematic *np*-argument internally converts to the variable J in the body. The semantic contribution introduced by the *pp*-phrase and likely to be equal to the meaning of the complement of the (semantically neglectable) preposition, internally converts to the variable M. No semantic contribution of the particle is envisaged at top level. Moreover, the event quantifier itself is made explicit and stored, binding the variable S. This quantifier is introduced by the phrase itself, as an argument to its finite head, the body, which specifies the semantic space inherent in the verb's finiteness; from a morphological point of view, the verb is taken to be an argument of the inflection.

The implicit typing of the λ-terms in (389) is expressed in the following way: the type of the meanings K and N above is that of an ultra-filter on the individuals to be assigned to the variables M and P, respectively. Implicitly, M and P are of type *e*, marking the semantic arguments of the event *admit*. As a consequence, both K and N are of type *<<et>t>*. In the same vein, the body of the top SLF – italicized in (389) – represents a quadruple abstraction over the variables M, P, J and S, the last one being internally 'bound': the body is of type *<e<e<e<e,t>>>>*. Lambda abstraction is not made explicit in the representations, since lambda conversion cannot be executed properly under Prolog (Pereira and Shieber 1987). In standard notation, however, template (389) would look like this, with essential types specified.

(390) *syntactic-semantic network of template* laat ... toe *'admit'* (normalized)

```
ID:A+B
...
SLF:{{[     λH.ℋ(H) > J,
            λZ.℘(Z) > M,
            λR.ℛ(R) > P,
            λQ_t.∃R_e.admit~[R] & event~[R] & entails1(R,incr))& Q
            & entails(R,incr)) > S],[],[]},
            λS_e.λJ_e.λP_e.λM_e.(theme_of~[S,J] & agent_of~[S,P]
            goal_of~[S,M]) & attime(S,T))& tense(S,pres))_t
    }
...
TYPE:s\0~[np^wh#B+O]/0~[pp^6#B+I,np^0#B+L,part^8#B+V]

ARG: ⎛ ID:B+I
     ⎜ SLF:H
     ⎜ SYNSEM:CAT:pp
     ⎝ ...

ARG: ⎛ ID:B+L
     ⎜ SLF:λZ_<et>.℘_<ett>(Z)
     ⎜ SYNSEM:CAT:np
     ⎝ ...

ARG: ⎛ ID:B+O
     ⎜ SLF:λR_<et>.ℛ_<ett>(R)
     ⎜ SYNSEM:CAT:np
     ⎝ ...

ARG: ⎛ ID:B+V
     ⎜ HEAD:PHON:toe
     ⎜ SLF:nosem
     ⎜ SYNSEM:CAT:part
     ⎝ ...
```

The relevant composition protocol supporting the necessary β-conversions, which in the DELILAH system is applied post-derivationally (see chapter 2), is as follows, for appropriate terms of type <<αβ>β> and <α^nβ>, respectively:

(391) $\lambda P.\wp(P)_{<\alpha\beta\beta>} \quad \circ \quad \lambda x_1. \ldots \lambda x_n.\ R(x_1, \ldots x_n)_{<\alpha^n\beta>} \quad \Rightarrow$
$\lambda x_1. \ldots \lambda x_{n-1}.\ \wp_{<\alpha\beta\beta>}\ (\lambda x_n.\ R(x_1, \ldots x_n)_{<\alpha\beta>})_{<\alpha^{n-1}\beta>}.$

Thus, the DELILAH lexicon provides well-typed but underspecified semantic structures, to be instantiated by unification. These structures carry the final propositional content. In this sense, knowledge of the meaning of a sentence is established lexically, by means of combinatory semantic categories.

3.2.5 Information Structure: to the limits of decidability

According to Frege's *dictum* in the preceding section, the sentence is the main theatre of meaning: effective meanings in natural language are propositions; whenever an expression means something, it is interpreted as a proposition. Propositions are analyzed, from the very dawn of semantics, in terms of truth and validity. This point of view has proven extremely fruitful. It brought us to syllogistic logic, intensionality and generalized quantifiers, to mention just a few landmarks. Yet, sentences in natural language – the prototypical propositions – connect to other sentences in a much more complicated way than by mere juxtaposition. First of all, the context of a sentence determines and delimits all kinds of model indices like time, reference, modality and focus. Second, it may also provide the proper semantic contribution of phrases, as in the case of tense and anaphora. Still, it is wise to realize that the question whether or not a sentence fits into a certain context may be undecidable, for the simple reason that a context cannot exclude any continuation. Not every continuation may be felicitous, but a continuation of a context is only *uninterpretable* if the continuation is syntactically elliptical, and the context does not provide a proper *filler*.

(392) *context*
John is walking the dog
continuation
And me a book, sometimes. (*)

In all other cases, the propositional interpretation of a continuation may not be a proper update to the context from a cognitive point of view, but we can tell so only after (partial) interpretation. Contextual anomaly presupposes inter-

pretation. Whether or not a particular sentence updates the context properly, is beyond computation, though. Interpretation would require the set of continuations to be enumerable. The consequence is that it makes no sense to enrich the lexicon with information for selecting or testing proper continuation. As far as we can see, informational 'ungrammaticality' at text level – inappropriateness – cannot be defined in any interesting, *i.e.* finite manner.
Information structure that can be encoded, however, is the grammatical substrate to focus. It comes in two forms: certain (relative) positions have focus by default, and certain phonological units are too weak to occur in those positions. For example, in Dutch the position at the left periphery of the finite constituent – SPEC-CP in generative terms – is in focus when not occupied (or blocked) by a subject (Zwart 1993). Non-subject constituents occurring in that peripheral position can be marked as being focalized. In the DELILAH categorial grammar, this position is detected. Combinatory categories are specified in the lexical templates. As a consequence, focus is predicted lexically for non-subject constituents in left-peripheral position. At the same time, phonologically weak pronouns must be prevented from occurring in that position. They are lexically marked as non-focal. So, they do not unify in that position, in parsing or in generation.

Yet, the amount of information structure to be captured explicitly in the lexicon is small. It is not nearly enough to cover the needs of discourse analysis. For example, at the level of the lexicon, a general decision whether a certain anaphor is bounded within the sentence is out of reach. By now, there are many strategies for detecting antecedents and ranking the candidates (for an overview, see Mitkov *et al.* 2001), but none of these relies on lexical information.

3.3 UNIFICATION: POWERING GRAMMAR CONSERVATIVELY

3.3.1 Procedures and specifications

A template – the lexical data structure we exploit – is basically a list of properties of phrases that are interpreted as constraints on their combinatorial use. When we state that a phrase is of category *np*, this statement restricts its occurrence to those positions which require or allow for an np. When a

construction imposes a category to one of its proper parts, this statement restricts the choice of candidates for that proper part. In our system, these constraints are put to work by a process called *unification*. Sag (2003: 56) has an interesting footnote on the difference between constraints and unification:

> *Theories of the sort we describe in this book are sometimes called 'unification based' but this term is misleading. Unification is a method (i.e. a procedure) for solving sets of identity constraints. But it is the constraints themselves that constitute the theory, not a procedure we might use with them. Hence, we will refer to the theory of grammar we develop, and the class of related theories, as 'constraint-based', rather than 'unification-based'*.

The almost classic division between representations and processes seems to be at stake here. The author chooses to have the representations prevail in characterizing the approach. From our point of view, however, the process too is an important part of the message. As a matter of fact, we choose and present the constraints in such a way that they can be resolved by unification, by *graph unification*, to be precise. We consider unification to be the name of the game played in combinatorial grammar. It embodies an important logical or algebraic device: negation or complementation. The logic of feature structures leaves no room for 'normal' negation or complementation. In the logic for feature systems as presented in Kasper and Rounds (1990), classical negation does not occur. The main reason for this is that a feature structure is – at best – a partial description, by definition. To negate, in a classical way, either the occurrence of a feature or a value to it amounts to expressing complete knowledge of the object that the feature structure describes. Complete knowledge, though, is not something any linguist should pretend to have. When, for example, we specify that *thing* is valued 'inanimate' for the feature *animacy*, we claim that there are occurrences of the word where it has or expresses that value. It is an existential statement. To use negation here would be universal: under no circumstances is the word 'animate' as for *animacy*. We can hardly claim to have access to this kind of certainty, as the word can occur in infinitely many contexts. Universal statements about properties of words or phrases in all possible contexts are beyond computation or experience anyway. Those statements would be essentialist or intensional.
Rounds (1997) argues that Carpenter (1992) deals with negative valuation by typing feature structures. Carpenter, however, must assume that the feature structures themselves are organized and can be typed – an assumption apt for HPSG as in Sag (2003), but not for our purposes. In our view, the constraints subject to unification do not impose a theory of language; in a certain

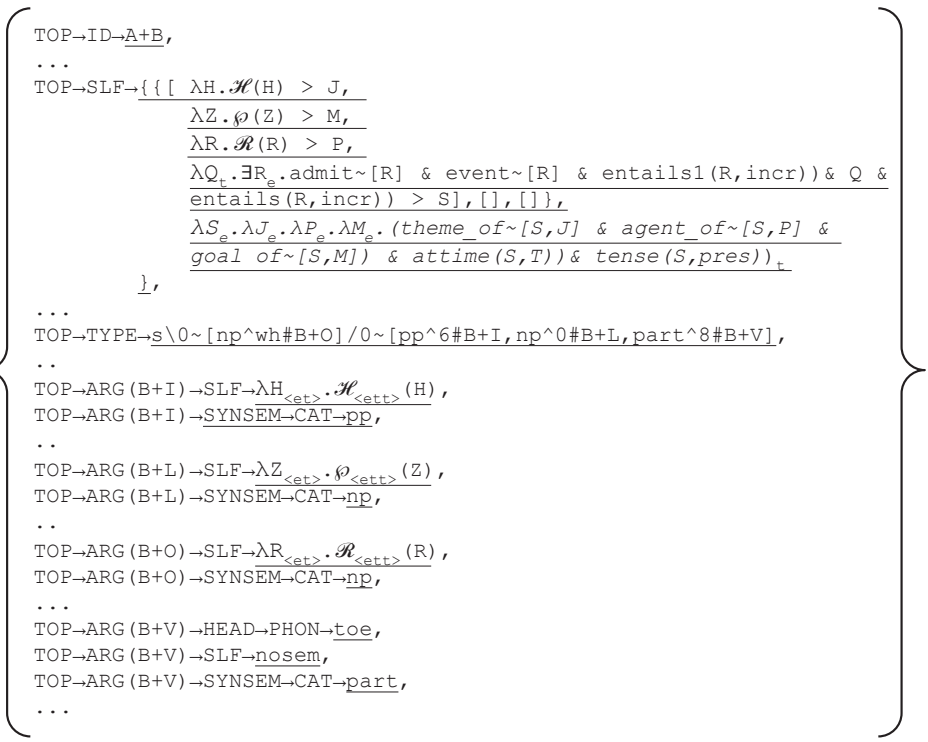
sense, our constraints are tools, rather than principles. They are just indices: handles to get things ordered properly.
When no complementation is available in the matter, the complementation must be in the procedure, in order to acquire a decent logic for our grammar. Unification offers this Boolean procedure. If two 'positive' objects meet under unification, their lack of being complemented is lifted by the openness of the question of subsumption: can the one be subsumed under the other? That is, we cannot tell whether or not a certain value or a certain feature must or cannot be assigned to a certain structure, but we cán tell whether or not a certain structure is compatible with another as it is. To put the matter in yet another perspective: unification does to feature structure what – in Frege's famous words – the proposition does to words: it creates a meaningful environment. Outside the gates of unification, a feature structure does not mean a thing.

This said, the operation mode of unification – basically a very fundamental mathematical procedure – must be defined. Firstly, we define a weakly connected directed acyclic graph by assuming a set of labels L for the vertices connected by directed edges such that:

(393) *Connected graph*
- (a) every vertex is labelled by exactly one label
- (b) there is exactly one vertex without incoming edge (*top: rootedness)*
- (c) there is a class of vertices without outgoing edges *(bottom)*
- (d) two vertices are connected by one edge at most
- (e) every bottom is labelled by a term

Clearly, a weakly connected acyclic graph can be represented as a finite set of paths from the top vertex to a bottom, where a path is a tree with a labelled top and a labelled bottom and in between nodes with exactly one in- and exactly one out-going edge. The weakly connected acyclic graph is equivalent to a forest of path-like labelled trees. This justifies our representation as a template, with every bottom spelled out even when occurring in more than one path. In this vein, template (390) has an idempotent representation as a set of paths, partially given below; bottoms are underlined.

(394) *representation graph of* laat ... toe *'admit' as a set of paths*



In our grammar, unification is an antisymmetric process: in every case of unification, a graph is projected on a well-defined weakly connected directed acyclic sub-graph labelled *arg(ument)* of another graph, and this process is not reversible. In (394), the paths starting with `TOP:ARG(B+I)` determine such a subgraph: the subgraph *governed* by that path. For the unification test itself, however, there is no hierarchy between the graphs involved. The definition of unification of feature value graphs, in terms of path unification, follows below. A path is a finite, directed sequence of labels, starting at the top node and ending with a value term; it is indicated as P→α, where α is the value term at a bottom vertex. The symbol '=' means 'equal to', while '⊔' stands for 'unifies with'.

(395) *Unification G1∪G2 of connected graphs G1 and G2*

(a) the unification G1∪G2 of G1 and G2 consists of all paths P→α⊔β for P→α in G1 and P→β in G2, where α⊔β exists, and of all paths R→σ in G1 or G2 for which no path R→τ exists in the other set.

(b) the unification G1∪G2 of G1 and G2 fails when there exist paths P→α in G1 and P→β in G2, but α⊔β does not exist.

(c) $\alpha \sqcup \beta$ exists iff either α is a variable or $\alpha = \beta$, or $\alpha = f(a)$ and $\beta = f(b)$, and $a \sqcup b$ exists
(d) if α is a variable, $\alpha \sqcup \beta = \beta$.
(e) if $\alpha = \beta$, $\alpha \sqcup \beta = \alpha$.

Even though our unification is steered in an antisymmetrical manner by the grammar, the resulting graph is subsumed by both of the *operanda*: the resulting feature structure $G1 \uplus G2$ is at least as informative as each of the original ones (Rounds 1997): it subsumes the originals. Since every combinatory process comes with unification of complex symbols, failure of unification amounts to local ungrammaticality, or denial of the hypothesis that two phrases combine.

Though unification is computationally expensive (but see Kešelj and Cercone (2002) for efficient techniques), the handling of (lexical) graphs themselves is liberal and relaxed. For example, one can add an edge to a lexical graph without any overall consequences. It is not necessary to add that edge to other graphs in the same class. In the same vein, edges can be removed without any overall consequences. Changing lexical graphs will have impact on only particular unifications. Since 'our' unification is conservative in that it neither destroys nor adds information on the fly and is governed by categories, the impact of extending or delimiting a lexical graph or a class of graphs can be predicted by inspection of the lexicon. As will be explained in section 3.5, this is easily done. Moreover, redundancy in the lexical specifications can be checked: a path $P \rightarrow X$ in a graph G of category C is redundant if no lexical graph has an argument subgraph of category C in which path P is specified.
Thus, control over the sets of lexical graphs is assured. Unification is effective, and therefore it is the backbone of every ambitious grammar automation. In this respect, it is important to observe that in DELILAH unification for parsing and generation is applied in a conservative and non-destructive way. Paths in a graph are not destroyed by unification for parsing or generation purposes. Unification leaves grammatical control to the lexicon, where the graphs are built and stored. Every single grammatical feature is guided from the lexicon; nothing happens outside its gates. The conservativity of unification is the anchor of grammatical lexicalism: for all information to be stored in the lexicon in a sensible way, we must make sure that no information is added during grammar application and no information is destroyed. We will see, though, that in *constructing* the lexicon unification – almost by necessity – must be applied destructively. In this sense, the mode of unification (conservative *vs.* destructive) is characteristic for the mode of the system: dynamic on-line

operations come with conservative unification; off-line the lexical database is built destructively.

3.3.2 Problems with re-entrance

Consider the two graphs in (396), assuming that the graph in (a) must unify its subgraph G with the graph in (b); the arrows indicate lexically imposed constraints.

(396) (a) G1: [.... [$_G$ slf: Y ↓, cat:xp ↓, ...[...slf:Y ↓ ...]] ...]
(b) G2: [.....slf: f(X), ...cat:xp, ...[...slf:X ...]]

Clearly, both X and f(X) will unify with Y, leading to an irresolvable instantiation loop: Y = X and Y = f(X), so X = f(X) = f(f(f..f(X)..)), *ad infinitum*. Though the problem is general for our form of *re-entrance* – variable linking within the same graph (see *e.g.* Bouma 1993) – we concentrate on the feature for the value for the Stored Logical Form. This feature typically has logically complex values (see section 3.2.4.2 and chapter 2).
The situation in (396) is not marginal: recurrence of a variable meaning at a higher level as in (b) is standard, and identification of lower and higher meanings as in (a) is typical for templates where the syntactical head does not contribute to meaning, *e.g.* lexically selected PPs or infinitival constructions with a purely functional complementizer. Moreover, we allow embedded constraints without restriction, as in the (a) template, to account for extended lexical units with specialized meanings. Still, we can argue that the additional requirement of co-categorization in (396) makes the clash unlikely; recall that every unifiable graph is categorized by definition, and therefore co-categorization is a precondition to unification. This being the case, the two templates in (396) show essentially different semantic dependencies within the same syntactical category. In normal combinatorial business, this is unlikely, since it runs counter to compositionality. But we must allow for zero-semantic phrases of any category, in which case nothing can be excluded. So, for the sake of argument, let us assume that (396)(a) requires a normal instantiation by a graph of category XP, and a non-compositional graph of that category is offered for unification. Now, the instantiation problem seems inevitable upon checking unifiability.

There are a few ways out. First, we may interpret the occurrence of the loop as failed unification. This is certainly in accordance with the remark on compositionality made above. Second, we can build in an *occurrence check* in these

cases, which is costly in terms of computational complexity. Third, we could impose the restriction on lexical templates that embedding of semantic conditions is not allowed – a kind of *head feature principle* as in HPSG (Sag 2003); it renders at least one of the two structures in (396) lexically illegal. This line contradicts our liberal attitude with respect to feature structures, to block as few collocational effects as possible. Fourth, we could not allow for 'eta-conversion' in unification – the unification of two variables – but instead delay unification until at least one variable is instantiated (*blockin*g; see Bouma and Van Noord 1994). But then we submit the syntax to unification, whereas we want the syntax to steer unification.
Reviewing the possibilities, we prefer the first option. The kind of clash in (396) is interpreted as failed unification, since the (semantic) networks in the two graphs differ essentially. However, we do not have to choose between options at all, because experiments do not demonstrate the problem. As a consequence, unification can be defined as elegantly as (395).

3.4 THE MAKING OF THE LEXICON

3.4.1 Organizing lexical knowledge: no lemmas – economy vs flexibility

The main characteristic of our lexicon is its flatness. It is a huge pancake, without external hierarchy. In particular, our lexicon is not organized by lemmas. Every word and every phrase occurs in all its individual glory without being subsumed under other levels of organization. For example, every finite form of a verb occurs in the lexicon with as many different graphs as it has combinatorial or semantic variants. This family of graphs is unordered, and so is the even larger class of graphs linked to other forms of that verb. The price to pay for this simple final structure is a complex set of rules for cooking the pancake. In this chapter, we describe the problems and the solutions of this *off-line* module.

Much of our knowledge of language is invested and exploited in the way the lexicon is generated. Although in the operational lexicon for parsing and generation lemma structure is not preserved, knowledge about lexical dependen-

cies, familiarity, generalizations and inheritance is compiled into the underlying generative system. The particular organization of this system, however, does not express principled knowledge of language. As a relatively simple example, consider the class of noun flections in Dutch. In general, a Dutch noun has a standard and a diminutive form, in both a singular and a plural edition. Also in general, we do not semantically distinguish between plural and singular forms – for the argument see Cremers (2002) – but we do semantically distinguish standard from diminutive. Plurals and singulars differ categorially: plurals in Dutch may be full noun phrases, singulars can be so only if they are non-countable. Not every diminutive derives its meaning from the standard form. Many plural forms are semantically indistinguishable from the singular. On the other hand, bare plurals – being NPs – must be differentiated from plural nouns – complements to quantifiers or determiners – and moreover, may have both a generic and an indefinite interpretation. Consequently, in our lexicon, the plural form *mannen* 'men' will come with at least three distinct templates, as listed below: a template as a noun with predicative semantics, a template as an NP with generic semantics and a template as an NP with indefinite semantics. In the latter two, a phonologically empty argument carries the noun meaning. This argument sub-graph is not specified for type, and therefore not open to syntax-driven unification. The crucial differential values are italicized.

(397) *lexical template of noun* mannen *'men'*

```
ID:A+B
HEAD:CONCEPT:man
     PHON:mannen
     SLF:man
PHON:C
PHONDATA:lijnop(mannen,A+B,[],C)
SLF:{{[],[],[]},D@man~[D]}
SYNSEM:AGGR:count
       CAT:n
       GENDER:nneut
       NUMBER:plur
       REFMODE:nontime
       SEX:male
TYPE:n\0~[]/0~[]
```

(398) *lexical template of generic plural np* mannen *'men'*

```
ID:A+B
HEAD:CONCEPT:man
     PHON:mannen
     SLF:man
PHON:C
PHONDATA:lijnop(mannen,A+B,[],C)
SLF:{{[{{[],[],[]},D@man~[D]}$E&(B+99)#F],[],[]},
    G@some^E^and(quant(E,gen), and(F,entails1(E,decr)),
    and(G,entails(E,incr)))}
SYNSEM:AGGR:count
       CAT:np
       FUNCR:incr
       GENDER:nneut
       NUMBER:plur
       QMODE:def
       REFMODE:nontime
       SEX:male
       SUBCAT:noun
TYPE:np\0~[]/0~[]
ARG:ID:B+99
    SLF:{{[],[],[]},D@man~[D]}
```

(399) *lexical template of indefinite plural np* mannen *'men'*

```
ID:A+B
HEAD:CONCEPT:man
     PHON:mannen
     SLF:man
PHON:C
PHONDATA:lijnop(mannen,A+B,[],C)
SLF:{{[{{[],[],[]},D@man~[D]}$E&(B+F)#G],[],[]},
    H@some^E^and(quant(E,some),and(G,entails1(E,incr)),
    and(H,entails(E,incr)))}
SYNSEM:AGGR:count
       CAT:np
       FREEARGS:no
       FUNCR:incr
       GENDER:nneut
       NUMBER:plur
       QMODE:indef
       REFMODE:nontime
       SEX:male
TYPE:np\0~[]/0~[]
ARG:ID:B+F
    SLF:{{[],[],[]},D@man~[D]}
```

Of course, these lemmas are produced by some mechanism, with some generality and some specificity. Below are a few scenarios for generating the

nominal graphs. In the *top-down* procedure, inheritance can be maximalized: independently of any particular noun, there is a general template for nouns, which is modified by a generation function in order to produce a dedicated graph for a particular instance of a particular noun. In the *bottom-up* procedure, no inheritance frame is pre-defined: we have a function that produces dedicated templates. This function, however, must embody the general properties of nouns in a dynamic manner. Between bottom and top – dubbed 'left corner' – there are several pre-defined templates, but they do not necessarily have any interesting common kernel that characterizes the category.

(400) *Top-down*
Define one single noun graph G_N. Determine for each noun N_m all its instances I_n. Determine per I_j the set of particular features F_k. Apply for each N_i, for each I_j and for each F_l a function σ from graphs, nouns, instances and feature sets to graphs: $G_{N, i, j, l} = \sigma(G_{N,} N_i, I_j, F_l)$.

(401) *'Left-corner'*
Define a set of noun graphs $\{G_p | G_p$ is a graph of category *n*$\}$. Determine for each noun N_m all its instances I_n. Per I_j, determine the set of particular features F_k. Apply for each N_i, for each I_j and for each F_l a function σ from graphs, nouns, instances and feature sets to graphs: $G_{p, i, j, l} = \sigma(G_{p,} N_i, I_j, F_l)$.

(402) *Bottom-up*
Determine for each noun N_m all its instances I_n. Per I_j, determine the set of particular features F_k. Apply for each N_i, for each I_j and for each F_l a function τ from nouns, instances and feature sets to graphs: $G_{i, j, l} = \tau(N_i, I_j, F_l)$.

The three strategies vary with respect to the integrity of lexical data structures. Under strategy (402) all data structures – lexical graphs – are created from scratch. Once created, they remain unaffected; even under unification, no information is destroyed. The functions that create the dedicated graphs contain functionally complete knowledge of the grammar of nouns.
Under the *top-down* strategy (400) the general template may have default values for certain features that can be overruled by the dedicated constructor functions. One can even conjecture that for a 'mother template' to be an interesting focus of nominal aspects, it must contain specifications that not every noun in each of its instances will maintain – default values are nothing more than that (cf. Bouma 1993). To state that the typical noun has a certain feature value is to state that some particular nouns may differ. Or, in different terms, the set of feature values that all nouns share is uninterestingly small. Therefore, the *top-down* strategy is either trivial, starting from an almost empty template, or destructive, as the template will contain information that is not

valid for all nouns. For the *top-down* strategy to be effective, it must allow the destroying of pre-specified paths in the top template.
The intermediate strategy seeks a balance between information destruction and sensible inheritance. It is really a matter of economy because both maintainability and linguistic transparency are at stake. If all information can be discarded in the course of the lexicon's construction, we may lose track of the level of grammatical knowledge invested in the pre-defined templates. If little or no information is shared between graphs of the same class, maintenance (adaptation, debugging, extension, ...) of the lexicon will get tough. Moreover, the construction functions for the dedicated graphs are among the most complex ones in the whole processing system (see below). Because they control vital parts of the attribute-value matrices and operate at the crossroads of generality and specialty, they too must be kept maintainable and adjustable. In the intermediate strategy of (401), the number of graphs underlying the generation of the lexicon is a matter of pragmatics, not of principle.
In the case of nouns, there is little grammar to help us to decide between either a basic template for singular count nouns and another for singular abstracts, or just one for both. In other cases, economy is more helpful. Every intransitive verb V in Dutch participates in a structure *zich een ongeluk V*, 'oneself an accident V' meaning 'to V very intensively'. Even without your calculator it is evident that creating one basic frame in which the intransitive verb is inserted and in which the crucial semantic dependencies are pre-established is much more effective than building every dedicated graph with this frame from scratch. Yet, both strategies will do the job.

The basic components of our lexicon can be summarized as follows:

(403) (a) a finite set of basic templates or graphs
(b) for each word a list of particular feature values with respect to one or more basic templates
(c) a set of rules, applying (b) to (a).

Actually, the particular specifications of a word can also be constructed as graphs. Ultimately, then, the sets (403)(a) and (b) may coincide – the lexicon is practically handmade, in that case. But descriptive economy may come in. A finite verb form, for example, lives on both properties of the individual form and on properties of the finiteness construction. It would be redundant to mix these two constructions over and over again in new unique graphs. Therefore, we had better take either a verb template to be modified with finiteness or a basic finite template to be enriched with verbal specifications – the nature of this choice will be discussed below. Set

(403)(c) contains the rules for performing these modifications. These apply a destructive form of unification called *adaptation*:

(404) *Adaptation*
The adaptation of a (basic) graph B to a specific graph C is the connected graph B[C] that unifies the largest subgraph B' of B that unifies with C, and C. That is: B[C] = B'⊎C.

Typically, B does not subsume the resulting graph B[C]: if B does not equal B', B contains information that is 'overwritten' in B[C]. Moreover, the specific graph C is typically dynamic, in that it is not stored and applied by general adaptation but it is imposed on the basic graph in a rule-like fashion: adaptation of one graph to another is a very precarious process, with all kinds of procedural implicatures too subtle to leave to a one-size-fits-all process. The use of rules to overrule or overwrite default specifications establishes the operational antisymmetry of the adaptation procedure.

Adaptation is the *off-line* constructor of the pancake lexicon – the flat compiled-out lexicon where every item is an object equal to all others. Moreover, adaptation offers a metric for lexical redundancy: the number of paths that are actually overwritten in constructing the lexicon – the overall sum of the differences between B and B' in definition (404) – can be compared to the overall number of specifications in the specific graphs C, *i.e.* the union of set (403)(b). The metric may operationalize the notion of *default value*.

3.4.2 General and special: everything as a graph

From the point of view of complex symbols or signs, it is not easy to decide whether a finite verb primarily represents finiteness or its verbal semantics. As a matter of fact, in the grammar of the Germanic languages, for example, the double nature of the finite verb is a major source of syntactical complexity: the structural position for finite inflection and the position of the verb are not necessarily adjacent. Morphologically, in these languages the verb provides the free morpheme, and finiteness is suffixed. But the morphological difference is combinatorially quite uninteresting: the bound morpheme is the functional element and it determines the main syntactical position of the word.
In a flat lexicon like the one we pursue, where every single word occurs in its own right, there is little compelling reason to derive the finite verb from its verbal root rather than deriving it from the finite basic construction, for example. There is a practical reason for choosing the de-verbal option, how-

ever: the verbal root is not predictable, whereas the bound morpheme comes from a closed class, and the irregularities in the verbal paradigms are easier to connect to the verbal roots than to the finite morphemes. Moreover, the finite form may inherit its basic valence and its thematic structure from the verbal root, so that not all syntactical features are determined by its finiteness.
Therefore, we derive finite forms by adapting a verbal basic template to a set of constraints inducing finiteness. The procedure is adaptation and not just unification, because at least the combinatorial category must be updated in the transition from infinite to finite signs: at least the category value is destructively adapted.
In a certain sense, this manoeuvre is upside down, as we apply general rules – imposing the templates of finiteness – to the more or less special attributes of a particular verbal template. The unprincipled trade-off that characterizes the in-between approach to lexical derivation shows up in (401): the special information contained in the verbal root's graph is overruled by more general, finite 'defaults'.
The same kind of decision we made with respect to de-verbal adjectival and nominal forms applies to the participles. First, a participle is encoded or 'signed' as a verbal derivative, and then its adjectival and nominal instances are produced by complex instances of adaptation: the process is inevitably destructive.

Below you will find the prerequisites of production for a standard verbal paradigm.

(405) (a) there is a *lemma* specifying
- a non-empty graph of characteristic attribute-values for that verb
- the address of one or more basic templates to be adapted by these characteristic values
- a (possibly empty) list of deviant morphonological instances of the verb

(b) there is *a set of retrievable basic templates*, elements of which the lemma can refer to
(c) there is *a set of rules* producing the morphonological paradigm of the verb
(d) there is a *set of classes of rules* each of which
- specifies a characteristic attribute-value matrix for every particular instance of the verbal paradigm
- adapts each of the templates referred to in the lemma, after adaptation to the lemma characteristics, to the attribute-value matrices for the particular instance

(e) there is a *set of classes of rules* each of which
- specifies a characteristic attribute-value matrix for a particular de-verbal instance
- adapts a particular matrix of a particular instance of the verb to this characteristic attribute-value matrices

This scheme of prerequisites does not fully reflect the actual organization of the lexical module, but it identifies the main agents in the production of a verbal paradigm in the present DELILAH architecture; similar schemes can be drawn up for nouns, adjectives and other categorial paradigms. The agents in the process could even be designed differently: the 'functional' matrices applied by the rules could just as well be independent initial templates like the ones mentioned in (b). This would lead to the following architecture.

(406) (a) for each lemma L introducing verbal paradigm V, specifying a paradigm-specific template TP
(b) for each template TV to which L refers
(c) for each morphonological instance M of V
(d) for each template TM specifying a particular function of M
(e) there is a function σ(TP, TV, TM, T) adapting TP to TV and TM to a final template T

Thus, we have a fully constructional approach, in which every grammatical specification is a retrievable template, and in which only one rule is called for, namely adaptation. There are templates not only for transitive verbs, but also for second person singular present finite forms with inversed subject, passive participles occurring as predicative adjuncts, nominalized generic plurals of present participles, and so on. This may be a desirable architecture of the lexical component: every grammatical object is a template, subject to adaptation. Unfortunately, however, the functions σ of (406)(e) may have to vary largely with these functional templates TV. When applying destructive modes of unification, it is unlikely – to say the least – that you will be able to use a one-size-fits-all adaptation strategy. Rather, every adaptation will require its own provisos, in particular with respect to the combinatory categories. Thus, under set-up (406) we still stick with a – possibly large – number of construction-specific adaptation rules: a set that may hardly be distinguishable from the present complexes which are referred to in (405)(d) and (e), to be discussed below.

3.4.3 The rules that make the language

Adaptation of graphs, templates or matrices is a powerful *off-line* process. It is almost too strong to apply *on-line*, as its destructive nature might be a threat to compositionality. For example, it does not impose a principled antisymmetrical relationship between the graphs involved, as is the case with unification in the way it is applied in DELILAH. As a constructional tool in building the lexicon, however, it is very useful. It provides compactness and generality,

and its finite product can be submitted to all kinds of well-formedness checks. Yet, adaptation itself is too coarse to be applied without additional steering. Just as syntax guides unification, adaptation is controlled by rules that specify constraints on the merge of graphs per category and subcategory. Since these rules compile knowledge of the structure of language, we will sketch their operation in this section by scrutinizing the making of the full paradigm of a simple transitive verb, *slaan* 'to beat'.

The root of the paradigm is manually specified as an instance of relation `lemma/5`.

(407)
```
lemma(
        slaan,          /* naming some root form of the paradigm */
        verb,           /* identifying the rule set to which the lemma is submitted */
        [transitive_verb, transitive_verb_with_small_clause],
                        /* listing the basic templates for the paradigm */
        [head:phon:slaan, head:concept:beat, head:sem:beat,
        head:synsem:etype:event, arg(10):synsem:theta:agent_of,
        arg(1):synsem:theta:theme_of],
                        /* specifying the particular paths for this paradigm with
                        respect to the basic templates listed in the third argument */
        [pressing12:sla, pressing23:slaat, pastsing:sloeg,
        pastplur:sloegen, participle:geslagen] ).
                        /* specifying (orthographically) 'irregular' (strong) forms of
                        the paradigm */
```

The lemma exclusively contains otherwise non-predictable properties of the *slaan* paradigm: it may occur in a sentence with a resultative small clause; it is an event rather than a state; its subject is an agent and its object a theme; moreover it entertains a few particular finite and infinite forms. Even at this elementary level, other decisions could have been made: the small-clause extension could be generalized to a default property of all transitive verbs; events may have agents as subjects by default; the forms might be predictable from tracing *slaan* to Indo-European verb classes. Clearly, every alternative choice would be reflected in the basic templates. Moreover, `lemma/5`'s third argument consists of a template in the form of a number of paths, to which the verbal templates named `transitive_verb` and `transitive_verb_with_small_clause` are adapted.

Because *slaan* is addressed as a verb, the construction of its paradigm is submitted to the class of verb-specific rules. This class is controlled by a single predicate `lemmalist/2`, which simply takes the lemma and produces a set of final templates, the full paradigm. The paradigm exists only in the

output of this predicate. As soon as the output set is adapted in the lexicon, the paradigm is lost.
`Lemmalist/2` performs the following operations:

(408) (a) it computes all different word forms in the paradigm *(sla, slaat, sloeg, sloegen, slaand, slaande, geslagen)*;
(b) for each template T in the lemma, it creates a basic instantiation BT by adapting it to the particular values of *slaan*;
(c) for every single finite and infinite form, it calls a class of rules adapting the basic instantiations BT into final templates;
(d) it calls a class of rules adapting some non-finite final templates into all de-verbal adjectival and nominal final templates.

The morphological component is not essential to our system, but economical. Although inflection is regular in general, the component deals with some mean properties of Dutch orthography. Its main function is reducing the number of forms that have to be listed in the (hand-made) lemmas.
The rules creating the finite forms are among the most complex of the whole system. In our combinatorial grammar, finite forms introduce the sentential category (see chapter 1). Therefore, all order variation at the top level of a sentence, including processes like left dislocation and inversion, has to be accounted for in the finite categories – it must be specified somewhere anyway. Moreover, Dutch is a verb-second language, which implies that main sentences and embedded sentences have different word orders, different combinatorial categories and, therefore, different finite 'heads'. Furthermore, main sentences come in three different semantic types. In addition, indefinite subjects may come with 'spooky' *er*. Although this number is partly an artefact of the grammar formalism we chose, a simple finite instance of a transitive verb like first person singular present *sla* is the phonological content of at least twenty-two final templates. All these final templates are adaptations of the basic templates, imposed and controlled by the rules for finiteness. They determine the sentential type, the combinatorial category, the order of arguments, their modes and their side effects. To follow the different stages, consider first the basic template for simple transitive verbs, with many variables to indicate the internal network.

(409) *basic template transitive verbs*

```
ID:Top+ID
HEAD:PHON:_X
     SLF:Main
     SYNSEM:ETYPE:Etype
            FLEX:infin
            VTYPE:transacc
SLF:{{[SemS&(ID+ID1)#A,
       SemO&(ID+ID2)#B,
       EStructure@some^E^and(quant(E, some), Main~[E],
       Etype~[E], entails1( E, incr),
       and(EStructure, entails(E,incr))) &(Top+ID)#EV], [],[]},
       Time@and(and(Stheta~[EV,A], Otheta~[EV,B]),
       attime(EV, Time)) }
SYNSEM:CAT:vp
       EVENTVAR:EV
       EXTTH:Stheta~[Top+ID, A]
       PREDTYPE:nonerg
       TENSE:untensed
ARG:  ID:ID+ID1
      PHON:_Subj
      SYNSEM:OBJ:subject_of(Top+ID)
             THETA:Stheta
      SLF:SemS
      ..

ARG:  ID:ID+ID2
      PHON:_Obj
      SYNSEM:CASE:obliq
             CAT:np
             DIR:left(1)
             FLAG:0
             OBJ:dirobject_of(Top+ID)
             THETA:Otheta
      SLF:SemO
      ..
```

All terms starting with a capital are variables. The values for the cases of subject and object are fixed by default. The target type of the basic template is set to *VP* and its tense to *untensed*, but that is fairly arbitrary.

One of the many final templates generated by the finite rules is the following, representing the first person singular present of a simple transitive verb in main (yes/no) questions. Its format is slightly different from the basic template's format – the former is generated, the latter is handmade. Paths and/or values changed (not instantiated) or added are italicized.

(410) *lexical template* sla *'beat' 1st person sg pres, main question*

```
ID:A+B
HEAD:CONCEPT:beat
     PHON:sla
     SLF:beat
     SYNSEM:ETYPE:event
            FLEX:fin
            NUMBER:sing
            PERSON:1
            TENSEOP:at-pres
            VTYPE:transacc
PHON:C
PHONDATA:lijnop(sla,A+B,[arg(right(-10),0,D),
         arg(right(-1),0,E)],C)
SLF:{{[F&(B+G)#H, I&(B+J)#K,
      L@some^M^and(quant(M,some),beat~[M],event~[M],
      entails1(M,incr),and(L,entails(M,incr)))
      &(A+B)#N],[],[]},
      quest$$and(and(and(agent_of~[N,H],
      theme_of~[N,K]),attime(N,O)),tense(N,pres))}
SYNSEM:CAT:q
       EVENTVAR:N
       EXTTH:agent_of~[A+B,H]
       PREDTYPE:nonerg
       SUBQMODE:P
       TENSE:tensed
TYPE:q\0~[]/0~[np^0#B+G,np^0#B+J]

ARG:  ID:B+G
      PHON:D
      SLF:F
      SYNSEM:CASE:nom
             CAT:np
             NUMBER:sing
             OBJ:subject_of(A+B)
             PERSON:1
             QMODE:P
             SUBCAT:pron
             THETA:agent_of

ARG:  ID:B+J
      PHON:E
      SLF:I
      SYNSEM:CASE:obliq
             CAT:np
             DIR:left(1)
             FLAG:0
             OBJ:dirobject_of(A+B)
             THETA:theme_of
```

One can see, for example, that the top-level semantics SLF is instantiated, whereas agreement features of the subject are added. Most combinatorial specifications are added or introduced by adaptation. The combinatorial TYPE and the PHONDATA (the linearization information) are absent in the basic template (409): they can only be added as the last step towards finalization. It is important to note that (409) does not subsume (410) but that (410) is an adaptation of (409) to (a) the lemma-specifications of (407) and (b) the following template for the first-person-singular-present-transitive-main question:

(411) *basic template 1st person sg pres main question*

```
ID:A+B
HEAD:SLF:MAIN
     SYNSEM:FLEX:fin
            NUMBER:sing
            PERSON:1
            TENSEOP:at_pres
PHON:C
PHONDATA:lijnop(Main,A+B,[arg(right(-10),0,D)],C)
SLF:{{[S&(B+F)#G,H@some^I^and(and(quant(I,some),
    event~[I, Main], entails1(I,incr)), H, entails(I,incr))
    &(A+B)#J], [], []}, quest$$and(and(Extth~[J,G],
    attime(J,K)), tense(J,at_pres))}
SYNSEM:CAT:q
       EVENTVAR:N
       EXTTH:Extth~[A+B, G]
       SUBQMODE:P
       TENSE:tensed
ARG:   ID:B+F
       PHON:D
       SLF:S
       SYNSEM:CASE:nom
              CAT:np
              OBJ:subject_of(A+B)
              PERSON:1
              QMODE:P
              SUBCAT:pron
              THETA:Extth
```

That is, the rule of finiteness creating the final template (410) may also be seen as one of the many instances of the adaptation relation σ/4 referred to in (406)(e).

Lemmalist/2 also produces all relevant de-verbal templates, according to (408)(d), apart from the many finite and infinite templates in the *slaan* paradigm. Among others, it feeds the templates establishing the passive participle

geslagen 'beaten' to a class of rules that create final templates for each of the following combinatorial varieties of the participle:

(412) de *geslagen* munt 'the stroken coin' *attributive adjective*
de *achthoekig geslagen* munt 'the octagonal stroken coin'
attributive adjective with resultative small clause
de *door de staat geslagen* munten 'the by the state stroken coins'
attributive adjective with agentive pp
de *door de staat achthoekig geslagen* munten
'the by the state octagonal stroken coins'
attributive adjective with agentive pp and resultative small clause
een *geslagene* 'a beaten (one)' *singular noun*
alle *geslagene* 'all beaten (ones)' *plural noun*
alle *geslagenen* 'all beaten (ones)' *plural noun for human beings*
de munt bleek eenmaal geslagen pijnlijk te ontwaarden
'the coin appeared once stroken to devaluate painfully'
...

In a lexicalist approach, all these forms must be made available and retrievable. In the DELILAH set-up, they result from a complex, partly destructive, adaptation of verbal and combinatorial or functional templates. This choice is, again, pragmatic. For constructions like nominalizations – the classical case for lexicalism in generative grammar – we have chosen to introduce them as such, rather than to derive them by adaptation from verbal templates; the choice is reasoned in Reckman (2009). Adaptation may become too complex or too intrinsic to be viable.

The two sets of rule classes in (405)(d) and (e) differ from each other in that the (d) classes take pre-final graphs and deliver final graphs, where the (e) classes take a final graph and adapt it into another final graph. The (d) classes, for example, produce the final finite templates out of raw material. The (e) classes produce final adjectival and nominal templates out of fully-specified participle templates, which remain part of the final pancake lexicon. The distinction, again, is pragmatic rather than theoretical. In fact, no aspect of the set-up (405) is mandatory: the single defining feature of our lexicon is its pancake outcome. All inheritance hierarchy and all differences between general and particular specifications are exclusively exploited in the off-line generation of the lexicon, not in its surface structure. At the end of the day, all graphs in the lexicon are equal and independent of each other.

3.4.4 The constructive lexicon

The lexicon – of any natural language – abounds in *extended lexical units* (elus): phrases with specialized meanings or combinatorics. And if everything is a graph, so is an extended lexical unit. Poß (2010) extensively discusses the variation in syntactical, morphological and of course semantic respect. Here we discuss the DELILAH implementation of the so-called *way*-construction in Dutch, as an illustration of the handling of complex elus. An instance mentioned before is repeated here, now with mandatory parts of the construction italicized:

(413) Geen enkele bankier had *zich een weg naar* de Raad van Bestuur kunnen *golfen*
No banker had himself a way to the Board of Directors been-able play-golf
'No banker could have succeeded in moving into the Board of Directors by playing golf'

We assume that Dutch has a verbal pattern of the following kind:

(414) subject + intransitive verb + reflexive pronoun + NP with 'way'-like nominal head and an attributive directional PP

Its meaning can be summarized as:

(415) *Subject* is *verb*-ing in order to arrive at *the complement of the preposition in the directional PP*

The main linguistic characteristics of the construction are these:

(416) (a) every intransitive eventive verb is eligible;
(b) the construction is causative: the verb itself is not providing the main event of the construction; rather, the verb is instrumental;
(c) although the verb is intransitive, a reflexive anaphor occurs in an object-type position; its semantic contribution may be restricted to stressing the causative nature of the construction;
(d) the head of the *way*-NP is lexically fixed but semantically irrelevant; it may just contribute to the causative interpretation; its proper semantics do not play a role;
(e) the complement of the preposition is lexically open, just like the subject; it provides an essential parameter of the interpretation;
(f) the determiner of the *way*-NP coincides with the top-level event marking; it is a singular indefinite, and it may carry the possible negation: it is either *a* or *no*.

These constructions have to be accounted for in a semantic lexicon of Dutch. In general, elus require choosing one of the words involved as a hook. As our lexicon will contain only fully-specified and independent objects, in the DELILAH system this choice concerns only the adaptation of templates, not their final ordering or embedding in the lexicon. In the matter of the *way*-construction, the construal is fairly easy: because the syntactic head is the verb – its inflection determines the combinatorial potential of the whole – the best choice is to introduce the constructions as lemmas of every candidate verb. To this end, our lexicon contains the template (417), comparable to the general verbal template for transitivity (409); its main semantic specifications are underlined.

All intransitive verbs, *i.e.* their lemma-specifications, are adapted to this template, in addition to the other basic templates to which they are linked. The verbal rules discussed in the preceding section also apply to the outcome of this adaptation, yielding the full spectrum of finite and infinite forms. As a consequence, the pancake lexicon will also contain, among many other instances of the verbal paradigms for *lachen* 'to laugh', the form (418) – with the same network in bold as in (417), and where DIR and FLAG values of the arguments were used to compose the syntactical TYPE.

Template (418) represents the *way*-construction with *lachen* as it occurs in embedded sentences, in finite past singular form, as in:

(419) Wij wisten niet dat de bisschop *zich een weg naar het Vaticaan lachte*
'We did not know that the bishop laughed himself a way into the Vatican'
We did not know that the bishop made it into the Vatican by laughing

The template occurs in the lexicon on a par with all other templates derived from *lachen*. Its presence there, as an independent and fully-equipped combinatorial object, is an immediate consequence of the lexicon's pancake 'architecture'. Moreover, many more constructions with verbal heads have to be compiled out – a clear overview of collocations per verb or per head is not available. In its final edition, then, the lexicon will contain many millions of such objects. Such a lexicon is viable only if it can be disclosed efficiently for parsing and generation tasks. The access to this type of lexicon is the theme of the next section.

(417) *basic template for Dutch* way*-construction*

```
ID:Top+ID
HEAD:PHON:_X
     SYNSEM:CONCEPT:Main
            ETYPE:Etype
            FLEX:infin
            SLF:Main
SLF: { { [ SemS&(ID+ID1)#A, SemPPNP&(ID3+ID4)#C,
           EStructure@some^E^and(quant(E, Quant), Main~[E],
           Etype~[E], entails1( E, FuncL),
           and( EStructure, entails(E, FuncR)))&
           (Top+ID)#EV],[],[]},
       Time@and( and( Stheta~[EV,A], some^EE^and(quant(EE,some),
       and(and(event(EE), move(EE)), theme_of(EE, A)),
       goal_of~[EE, C], entails1(EE, incr), cause(E, EE)),
       entails(EE, incr)), attime(EV,Time)) }
SYNSEM:CAT:vp
       EVENTVAR:EV
       EXTTH:Stheta~[Top+ID, A]
       PREDTYPE:nonerg
       TENSE:untensed
ARG:   ID:ID+ID1+10
       PHON:_Subj
       SLF:SemS
       SYNSEM:NUMBER:Number
              OBJ:subject_of(Top+ID)
              PERSON:Person
              THETA:Stheta
ARG:   ID:ID+ID2+1
       PHON:_Obj
       SLF:{_Stores0, _@Quant^_^_}
       SYNSEM:CASE:obliq
              CAT:np
              DIR:left(2)
              FLAG:0
              FUNCL:FuncL
              FUNCR:FuncR
              NUMBER:sing
              PERSON:3
              QMODE:indef
              THETA:theme_of
       ARG:   ID:ID2+_ID4+1
              HEAD:CONCEPT:road
              SYNSEM:CAT:n
                     NUMBER:sing
ARG:   ID:ID+_ID5+3
       PHON:_Reflex
       SLF:_RefSem
       SYNSEM:CASE:obliq
              CAT:np
              DIR:left(3)
              FLAG:0
              FOCUS:nonfocus
              NUMBER:Number
              PERSON:Person
              PRON:refl
              SUBCAT:pron
ARG:   ID:ID+ID3+2
       HEAD:CONCEPT:towards
       PHON:_PP
       SLF:_SemPP
       SYNSEM:CASE:obliq
              CAT:pp
              DIR:left(1)
              FLAG:0
              THETA:goal_of
       ARG:   ID:ID3+ID4+1
              SLF:SemPPNP
              SYNSEM:CAT:np
```

(418) *lexical template for* lachte *'laughed' as head of the* way*-construction*

```
ID:A+B
HEAD:CONCEPT:laugh
     PHON:lachte
     SLF:laugh
     SYNSEM:ETYPE:event
            FLEX:fin
            NUMBER:sing
            PERSON:C
            TENSEOP:at-past
            VTYPE:transacc
PHON:D
PHONDATA:lijnop(lachte,A+B,[arg(left(1),0,E),arg(left(3),0,F),
         arg(left(10),0,G),arg(left(11),wh,H)],D)
SLF:{{[I&(B+J)#K, L&(M+N)#O, P@some^Q^and(quant(Q,R),laugh~[Q],
       event~[Q], entails1(Q,S), and(P,entails(Q,T)))&(A+B)#U],[],[]},
       and(and(and(experiencer_of~[U,K],some^V^and(quant(V,some),
       and(and(event(V),move(V)),theme_of(V,K)),goal_of~[V,O],
       entails1(V,incr),cause(Q,V)),entails(V,incr)),attime(U,W)),
       tense(U,past))}
SYNSEM:CAT:s_vn
       EVENTVAR:U
       EXTTH:experiencer_of~[A+B,K]
       PREDTYPE:nonerg
       SUBQMODE:X
       TENSE:tensed
TYPE:s_vn\0~[pp^0#B+M, np^0#B+Z, np^0#B+Y, np^0#B+J]/0~[]

ARG:  ID:B+J
      PHON:G
      SLF:I
      SYNSEM:CASE:nom
             CAT:np
             NUMBER:sing
             OBJ:subject_of(A+B)
             PERSON:C
             QMODE:X
             THETA:experiencer_of

ARG:  ID:B+Y
      PHON:F
      SLF:B1
      SYNSEM:CASE:obliq
             CAT:np
             FOCUS:nonfocus
             NUMBER:sing
             PERSON:C
             PRON:refl
             SUBCAT:pron

ARG:  ID:B+Z
      PHON:H
      SLF:{C1,D1@R^E1^F1}
      SYNSEM:CASE:obliq
             CAT:np
             FOCUS:nonfocus
             FUNCL:S
             FUNCR:T
             NUMBER:sing
             PERSON:3
             QMODE:indef
             THETA:theme_of
      ARG:  ID:Z+G1
            HEAD:CONCEPT:road
            SYNSEM:CAT:n
                   NUMBER:sing

ARG:  ID:B+M
      HEAD:CONCEPT:towards
      PHON:E
      SLF:A1
      SYNSEM:CASE:obliq
             CAT:pp
             THETA:goal_of
      ARG:  ID:M+N
            SLF:L
            SYNSEM:CAT:np
```

3.4.5 The lexicon is local

In the development of grammatical theory, the idea that grammatical relations must be restrictive is fundamental. Not every observed dependency as such qualifies a grammatical construct. Generative grammar almost lives by restrictivity, although it started out by widening the concept of rule of grammar by introducing transformations. Yet, the quest for conditions on transformations is still a fine way to describe the grammar programme of the generative 'enterprise'. The most impressive results have been in the field of locality: no rule of grammar relates items over a random structure, or: if two elements are interdependent, it is always possible to finitely characterize the path between them. In early generative grammar, this resulted in the condition on transformations that essential variables not be involved.

Sag (2003: ch. 16.6) repeats the message for HPSG in a very strict way: no construction has access to its daughter's daughters, and that is not just handy but fundamental. It is, for example, 'a striking fact about human languages' that they do not exhibit verbs that select for case in their verbal complement. So concepts of locality like this keep grammatical descriptions within the borders of theoretical relevance, restricting the notion *possible (grammar of a) language*.

Still, under a fundamental lexicalist approach to grammar, a less restrictive but more intrinsic notion of locality than Sag's access principle presents itself. If all grammatical relations were made explicit in the lexicon and the lexicon consisted of fully-specified combinatoric objects, all grammatical relations would be localized *because* they were lexical. Of course, we are free to make lexical graphs as complex as necessary to accommodate all dependencies, but complexity does not interfere with specificity. The only variables occurring in a lexical graph are terms at the end of a path. Every path is rooted, the graph is connected and directed, paths cannot be made recursive, and every graph is finitely traversable. In our system, the unit of locality is the lexical graph, in all its variety. That is, every interesting grammatical relationship can be specified within the borders of a single lexical graph. That, too, might be seen as a striking fact about human languages. This is what makes languages computable.

3.5 DISCLOSING THE LEXICON: OBJECT-ORIENTATION AND SPEED FOR SEMANTIC GENERATION

3.5.1 The enterprise

As was said before, DELILAH adheres to the 'gnostic' approach to computational linguistics: the view that explicit knowledge of language can be assembled, formalized and exploited. As the system focuses on semantics and the full specification of logical form, it must operate on a high level of grammatical and lexical fine-grainedness. As a consequence, DELILAH's lexicon is both detailed and large. It is generated on the basis of a restricted number of 'predefined', underspecified, generic templates, *e.g.* for transitive verbs or abstract nouns, representing minimal default graphs for all kinds of lexical types (see section 3.1.4). These templates can also be considered as constructions in the sense of Croft (2001). The templates formulate important generalizations about lexical relations, meanings and syntactical behaviour of phrases. This class of templates is flexible, and defined by practical and empirical considerations. A particular template is produced by specifying differences with respect to a more general template. In this sense, every template itself defines a lemma. A set of rules produces the lemma from the generic template(s), by inheritance, and the specified difference list. From this lemma, another set of rules produces graphs for all the marked instances – morphological or otherwise – of the lemma. The lexicon, a set of 'lexical entries', thus hinges on three components: a set of generic templates, a set of lemma difference specifications and a complex set of rules (see (403)). Data management, then, is done on a higher level than in a traditional database.

The lemmas or lexical entries are completely defined by HPSG-style feature-value specifications (Sag et al. 2003 and section 3.1.2). Lemmas are complex symbols, and can be represented by Attribute Value Matrices (avms), or Direct Acyclic Graphs (dags). Typically, they have a different number of features. A lemma may or may not specify a certain value for a certain feature. Besides atoms and numbers, values can be complex structures themselves, defining sub-graphs. A lemma will contain sub-graphs for semantically and/or syntactically related phrases. Unification, as defined in (395), will apply to these sub-graphs, that is, two graphs A and B unify whenever B unifies with a designated sub-graph of A, in which case A is called primary and B secondary. By definition, the primary graph constrains the secondary graph in every relevant aspect: morphologically, syntactically and semantically. This way,

the DELILAH lemma is a natural way of expressing collocational effects, from weak combinatory effects to rigid combinations. In fact, every lemma defines the domain for collocational effects. The lemma essentially separates constraints on sub-phrases of a structure from properties of the overall phrase. Inheritance, information sharing and co-indexing are specified by using the same variable as a value at different places in the graph. These places can be treated as pointers to the same memory location, rather than as identically valued contents. By definition, a graph is underspecified in the sense that not necessarily every possible or even actual value is lexically instantiated.

Section 3.4.2 describes the lexicon as a collection of explicitly defined, spelled-out, unrelated, and autonomous linguistic entities, containing all the necessary linguistic knowledge that is needed to properly use the word (form) in a language. By 'unrelated' we mean that the entities are stored independently of each other: there is no external hierarchy, and there are no imposed paradigms. By 'autonomous' we mean that entities, once retrieved, are operated independently of each other. A different approach is followed in Cornetto, which defines a combinatorial and relational, i.e. implicit, network on word level for Dutch (Vossen *et al.* 2007). In DELILAH such an information network, including, for example, collocations, has been 'compiled away', yielding real linguistic entities to start with, *e.g.* for generation purposes.

The following examples will give an idea of the lexicon's size and growth, and demonstrate the storage and access problem of a large computational lexicon. Adding the lemma for *hij* 'he' means adding 1 lexical entry, while adding the lemma for *gelopen* 'walked' (past. part.) means adding at least 19 entries; adding the lemma for *heeft* 'has' (3rd.pers. pres. sing. aux) means adding at least 133 entries; and adding the whole class of forms for the simple intransitive verb *verven* 'to paint' means adding 226 entries. Clearly, the fully written-out specification of lexical entries introduces an exponential storage factor. These figures are to be interpreted relatively and do not mean anything by themselves. They reflect the current draft state of the lexicon. Furthermore, for DELILAH's grammar-driven generation component, efficient access to the lexicon is crucial, because a word form should be produced only when its lexical specification matches certain constraints specified by the grammar and by the generation algorithm. It has been observed that searching and finding lexical entries is the main business of the generator. Therefore, efficient access methods are required for retrieval, with the ability to search and match complex lexical graphs and lexical constraints, which is hard. Finally, we developed our system in Prolog (Clocksin and Mellish 1984) for historical reasons. We implemented as many standards as possible, including ISO Prolog (Deransart et al. 1996; Clocksin and Mellish 2003), and refrained from using closed-

source libraries or third-party packages. This approach has yielded (almost) portable software. As a consequence, we are faced with the problem of implementing a fast and large-scale lexicon from a Prolog environment.

3.5.2 Two models

Lexicons may be modelled in different ways; here, we are focussing on computational models. The question whether linguistic information is stored centrally in the brain or rather in a distributed way is not addressed here. Neither is the question whether lexical entries are available in pre-compiled format, or get 'unfolded' on demand. A computational lexicon can be stored either in the internal, working memory of the computer, or in external, secondary memory, i.e. on a hard disk. Storing the lexicon in internal memory implies loading all lexical entries. Access to data structures in internal memory is usually very fast. Working memory, however, cannot be extended beyond a few gigabytes, which is not large enough for our purposes, while extension by virtual memory gives bad performance. The enormous proportions of the lexicon, let alone its foreseen expansion, and the limitations of current hardware rule out the option of storing it in internal memory. Storage of the lexicon externally does not pose a problem spacewise, but is slower and harder to access. As the lexicon is of a static nature, once it has been generated a full-featured database management system, including update facilities, is unnecessary. We can restrict ourselves to implementing a lexicon as a saved set of 'read-only' lexical entries, and provide efficient access methods for at least the most important features used by the generator, being the syntactical type, the semantic concept, and the word form.

The lexicon, built as a collection of lexical entries, can be regarded as a set of records in a database. There are a number of database models for database management. The Relational Model (Codd 1970) is based on two parts of mathematics: first-order predicate logic and the theory of relations. It is data-based and well-known from relational database management systems (RDBMS). As the programming language Prolog is based on the Relational Model, it seems straightforward to consider this model in more detail. Alternatively, the Object-Oriented Model (Meyer 1997) is investigated, because it was noted that a lexical entry in our system resembles the notion of 'object' in the OO Model. It can be called a 'linguistic object' in this model. A linguistic object stores data (lexical specification), and methods (procedures, *e.g.* for linearization). The OO Model is knowledge-based. Furthermore, the OO Model is often recommended when there is a need for high performance pro-

cessing of complex data, *e.g.* binary multimedia objects. The OO Model is well-known from graphical user interfaces ('point-and-click devices'), while object database management systems (ODBMS) are emerging.
Prolog – our programming vehicle – is itself relationally oriented rather than facilitating object orientation, but this feature did not influence the choice made. We assume that we can use the Prolog language as a powerful query to a relational Prolog database and apply Prolog's declarative semantics to the OO paradigm as well.

3.5.2.1 The Relational Model

The Relational Model was the first formal database model, solidly founded on well-understood mathematical principles and explained by Date (2003). It was invented in those days when computer memory was scarce and expensive. A relational database consists of a number of relations (often called tables), in which all data is stored. Each relation is a set of tuples that all contain the same attributes (the horizontal rows, often called records). A tuple is an unordered set of attribute values (the vertical columns, often called fields). An *n*-tuple is an unordered set of *n* attributes. An attribute is an ordered pair of an attribute name and a type name. Attributes are accessed by name, instead of by their position in the tuple. All of the attribute values (values in the same column) should be in the same domain, that is, they should be a valid value for the data type of the attribute, and they should obey the same constraints. Data types must be scalar, like an integer or a string, and cannot be compound, like a graph. Constraints provide a way of restricting the data that can be stored, either in tuples or in attributes. A relation is said to be *n*-ary iff it consists of a set of *n*-tuples. A special kind of constraint is a key. A key is an *m*-tuple of an *n*-ary relation, where $m < n$, which enforces the uniqueness of the combination of the *m* attribute values for each tuple. Key values are usually kept in an index table or hash table, which is stored in internal memory for fast access. By using keys, storage of duplicate data is prevented. The chance of duplicate data is further reduced by applying a set of normalization rules to the database structure. A 'relvar' (relational variable) is a named variable ranging over the set of tuples; as the result of a query, a subset of the set of tuples can be assigned to it, including the empty set.

The lexical entries of our lexicon are complex, non-atomic data structures. The Relational Model only allows atomic data types. It would be possible to pre-compile them into flat strings, which are atomic. Generally, however, flattening structured information is not a good idea when it comes to retrieval on the basis of some highly detailed substructure.

Lexical entries are directed acyclic graphs (dags), recursive data structures. Although it would be possible to pre-compile all feature-value paths of a dag to a number of tuples in different tables by means of a recursive procedure, it would be impossible to retrieve them since a relational database system does not provide for recursive processing (Hirao 1990). On the other hand, when we regard the lexicon as knowledge, and mark DELILAH as a knowledge-based system, we can linearize and store the graphs in a relational database, and use the powerful processing of recursion for inferences by Prolog. It is possible to successfully and easily store objects in a relational database (called 'Object-Relational mapping') by following a step-by-step procedure (Ambler 2000). A disadvantage is that quite a number of tables might be involved as graphs typically hold large numbers of features, while for semantic generation complete lexical specifications are required. Pre-compiling a lexical entry for a noun will typically yield a different number of tuples (in just as many tables) from pre-compiling a verbal entry. The top level, a main table, which is to represent complete lexical entries, has to span all the tuples transferring them into one large main tuple, in which each attribute represents a path, and where the attribute's value is either the value of the path or an ID that links to another table. This implies that there will be more than one top level, one table for each combination of attributes, and consequently more than one main table. As we do not want to impose a restriction on the internal dependencies of a graph, there is no restriction on the recursion depth of features. Consequently, the number of different main tables can be large in practice and infinite in theory. For practical purposes, e.g. retrieval by DELILAH's generator component, more than one main table means decreasing performance with orders of magnitude. Because of this, lexical entries cannot be regarded as single, homogeneous relations in the Relational Model; they cannot be retrieved as such.

Furthermore, lexical entries use variables, e.g. to co-index as yet unknown information between nodes in the graph. The Relational Model does not allow attribute values to be variables. A variable is not an atomic constant, and therefore not distinguishable from other values of the same attribute. As a consequence, an attribute that has a variable in its data domain cannot be indexed, which could lead to bad overall performance of the database. It would be possible to pre-compile variables into constants, and to de-compile them during retrieval. Calling meta-predicates, however, is generally rather time-consuming.

We conclude that the Relational Model, although appreciated for its economic storage, cannot efficiently accommodate (logic) variables, recursive features, and, consequently, the top level. Despite its efficient access, the Relational

Model does not meet high-performance demands on complex (recursive) data structures. The Relational Model is inappropriate for our purposes.

3.5.2.2 The Object-Oriented Model

In the Object-Oriented Model, information and control are represented in the form of interacting 'objects', as is well-known from the Object-Oriented Programming (OOP) paradigm. An object can be seen as a little information processor. It accepts commands (called 'messages') from other objects, processes data by executing procedures (called 'methods') that are stored in the object, and sends commands to other objects (called 'message passing') to be executed by those objects. It operates like a neuron in the brain, or a router in a computer network. By keeping data (properties) and procedures (operations) together in one local unit, an object holds all characteristics related to some concept. This implies that an object is a complex data structure. An object is independent of other objects; it has its own ID and its own role. These characteristics make an object attractive for representing 'behaviour'. OOP, then, is modelling a problem by distinguishing different (abstract) levels of objects (called 'classes' and 'subclasses', which are kept in a 'class hierarchy'), and defining their cooperation and interactions. A class defines the general characteristics of a concept in terms of the problem domain. An object is a particular instance of a class, from which it inherits all properties and methods, and to which it may add its own information, or overwrite inherited information. Inheritance may be seen as an 'is-a' relationship. Multiple inheritance means inheriting from more than one independent class, thereby combining properties and methods. Classes are the structuring elements (modules) in OOP, which hide the details of the code to be accessed by objects that stem from other classes.

Our linguistic objects are generated by deriving information from (one or more) generic classes of templates – 'constructions' – and by adding local information. After their creation, linguistic objects are independent of generic classes or other objects. This fits nicely in the OOP concept of objects that are constructed by a specialized 'constructor' method and by inheriting information from multiple classes. Our linguistic object, a complex, recursive graph, can easily be mapped onto an OOP object, which is a complex data structure. Shared variables, in fact, stand for a unification procedure, deferred until runtime. Encoding them by an OOP method is straightforward. Linguistic objects are independent information units, and, thus, uniquely identifiable, as are OOP objects. This makes an object, including all properties and methods, accessible by one ID, which is very important for efficient storage and access

by data-intensive processes such as semantic generation. ISO Prolog terms, being complex data types, are well equipped to represent OOP objects, including shared and singleton variables, recursion, and unique identification.
On the other hand, as linguistic objects are built from classes, and any combinatory difference is compiled out as a difference between objects, objects may differ from each other minimally, yielding a significant amount of overlap between objects, and introducing an exponential space factor. For example, the linguistic objects for the 2nd and 3rd person of a regular verb differ *only* in the person and phonological features.
Linguistic objects can adopt different states when they get involved in some linguistic process, like generation. When we talk about storing linguistic objects, we mean saving only their initial state. Objects whose states have been saved are called 'persistent'.
We conclude that the Object-Oriented Model provides a natural environment for representing linguistic objects. It does not suffer from the drawbacks of the Relational Model with respect to data modelling, and potentially facilitates fast access by unique identifiers. Its data redundancy is a small price to pay, given the considerable decrease in the price/performance ratio of hard disks each year. Future developments in runtime file (de-)compression techniques, as demonstrated on the level of the operating system, or the application, e.g. Java, might weaken this disadvantage. Prolog's term data type is suitable to represent linguistic data objects.

3.5.2.3 Object-Oriented Databases

An Object-Oriented database is a database which stores objects as created and modelled in OOP. Or, more strictly, an OO database system must satisfy two criteria: it should be a DBMS, and it should be an object-oriented system, i.e., to the extent possible, it should be consistent with the current crop of object-oriented programming languages (Atkinson *et al.* 1989).
OO databases arose after it was discovered that relational databases lacked high-performance processing on complex data structures like graphs. In the Relational Model, complex data, split and stored in several tables, have to be retrieved by searching these tables and combining pieces of data (called 'joining' in relational jargon), while in the Object-Oriented Model, a complete object is retrieved by its ID in one operation. In the Relational Model, the relational database is accessed by a declarative query language, typically SQL, which needs a procedural interpretation at runtime. The declarative query language permits general-purpose queries and transforms them into efficient retrieval procedures. In the Object-Oriented Model, lacking a standard query language, the OO database is accessed by means of a pre-compiled pointer

mechanism, which can be regarded as an optimized query answerer. In our application of a computational lexicon for semantic generation, we do need specialized queries, e.g. on syntactical type, semantic concept, and word form. Hence, an OO database is appropriate.

We come to the conclusion that, as our lexicon is static after it has been derived from generic templates, we do not need a full-blown object database management system (ODBMS) for persistent storage, but we can limit ourselves to implementing a simpler OO *lexicon*, containing fully-specified lexical entries, for retrieval only. We can represent objects by Prolog's native term data type, and manage them by Prolog's execution mechanism and pre-compiled pointers.

3.5.2.4 An example

We illustrate the operation of the generator by the following simplified example. In categorial grammar, a category consists of a head, and zero or more arguments to its left and/or right side. For generation purposes, the category can be regarded as an agenda. The generation algorithm keeps heads already produced in an unordered list, and arguments still to be produced in a stack. It handles them by inserting and deleting elements from an arbitrary position or from the top of the stack. It starts with a random semantic concept, and finds one of its realizations in the lexicon. The head of its category is inserted in the list. The arguments are shifted onto the stack, in reverse order. Each argument is produced either by reduction of some head in the list, or by reduction of a new category to be found in the lexicon. The topmost argument of the stack is replaced by the arguments of the new category, or removed completely, when it does not have arguments of its own. When the argument stack is empty, there are two possibilities. When the heads list does not consist of exactly one of the sentential categories s ('sentence') or q ('query'), the lexicon is consulted for a category that has the non-sentential category as an argument. Its head is inserted in the heads list, while its arguments are shifted onto the argument stack. If not, the algorithm stops. Categories, being complex symbols, are unified with each reduction step. The following example demonstrates the procedure, in line with (224).

1. search for random concept; find *betekenissen* 'meanings', cat=n; heads={n}, args={}
2. search for random entry with argument n; find *diepe* 'deep', cat=np/n
3. reduce arg n with head n in step 1; insert head np; heads={np}, args={}
4. search for random entry with arg np; find *ontdekt* 'discovers', cat=s\np1/np2
5. reduce arg np2 with head np in step 3; insert head s; shift arg np1; heads={s}, args={np1}

6. search for random entry with head np; find *die* 'that', cat=np/n
7. reduce arg np1 with head np in step 6; shift arg n; heads={s}, args={n}
8. search for random entry with head n; find *Nederlander* 'Dutchman', cat=n
9. reduce arg n with head n in step 8; heads={s}, args={}

Linearization information, kept in the complex symbols, is applied, yielding the final sentence *die Nederlander ontdekt diepe betekenissen* 'that Dutchman discovers deep meanings'.

3.5.3 Methods

We will describe methods that store our linguistic objects as objects in an OO lexicon, and methods, some borrowed from the Relational Model, that retrieve these objects efficiently and fast. The methods have been implemented in ISO Prolog. They should obey the Resource Principle, which is stated as: "Deploy working memory when performance is the key factor, and deploy external memory when storage is the main aspect". This principle is a practical phrasing of the insights that working memory will never be large enough to hold our current and planned number of linguistic objects, and that working memory is needed for real computation tasks, like semantic generation, and of the facts that working memory is faster than external memory, and that external disk space is abundant and cheap. We refrain from implementing interfaces to third-party products, e.g. the (semi-commercial) Berkeley DB library for external storage of terms (SICS 2014), or the Objects Package in SICStus Prolog (SICS 2014), because we prefer 'light', compatible, and manageable interfaces that are portable to other platforms.

3.5.3.1 Direct access and index tables

The 'Edinburgh' Prolog standard (Clocksin and Mellish 1984) offers sequential read and write access to files. On average, searching a record, given its number, out of N records will take time that is proportional to N/2. Although this is still efficient in complexity terms, for big N, however, searching will take an unacceptable amount of time. In ISO Prolog, the concept of a file has been improved on, and been replaced by the concept of a 'stream'. A stream is a file with random access, which means that each byte in the file can be located in constant time.
An index table is a table relating unique, simplex values to the positions, or context, where these values can be found. In our case, this translates to a table per feature that relates each feature's value to all occurrences of objects con-

taining that particular feature with that particular value. Internally, Prolog uses 'first-argument indexing' to locate the correct clause, given its first argument, which must be constant. This technique is not part of ISO Prolog, but it is regarded as an essential facility for interpreting Prolog programs efficiently, to be compared with the 'array' data type in other programming languages. It is implemented in every professional Prolog system on the market. An index table can be simply implemented by a collection of clauses, where each clause's first argument encodes the feature's values.
A fast lexicon for semantic generation hinges on the stream concept and indexing techniques.

3.5.3.2 Caching

Internal (working) memory can be accessed much faster than external memory. Applications often need the same data again and again. This has led to the development of cache memories. A disk cache is a storage mechanism in working memory that keeps the most recently read data plus the data of adjoining sectors in a buffer. As soon as the application asks for some data, the buffer is consulted first, saving access time to the hard disk. When the required data does not fit in the buffer, an extra disk access is necessary. To exploit disk caching, the data must be ordered in a way that corresponds to a relevant search criterion. Disk caching might be implemented at the application level or at the operating system level, or both. Caching at the application level, as advocated by Ceri *et al.* (1989), runs counter to the Resource Principle.

3.5.3.3 Hashing and compression

Hashing (Knuth 1973) is a method that converts an arbitrary, complex value to a simplex one (the 'hash' or 'key') by applying a hash function to it. Typically, hashing converts to an integer, because it is the most economical data type, and the easiest for a computer program to use. A hash value enables the use of an index table. Searching for an object in a collection of N objects, given an unhashed value of some constraint, will take O(N) comparison operations. When the value is hashed, and the objects are indexed on the constraint by applying one and the same hash function onto their values, searching an arbitrary object only takes O(1) time, assuming that the index table can be directly accessed. This is easy to implement in Prolog systems that can do first-argument indexing.
As each piece of data ever to be searched for is fixed, the lexicon is a collection of 'static search sets'. For such sets, a 'perfect hash function' (PHF) can be designed that is a function that will never assign the same hash value to

different data structures. However, depending on the complexity of the data structure, the hash can exceed the range of the integer data type on some computer platforms. A 'near perfect hash function' takes the integer range into account, but allows for duplicates ('collisions'). Duplicates increase space and time complexity. A 64-bit platform extends the integer range by several orders of magnitude compared to a 32-bit platform, enabling a PHF to be used, yielding zero collisions.

Number grouping is a compression technique which replaces a set of adjacent integers by a range starting with the lowest number and ending with the highest number. The space complexity for a range is constant instead of linear for a set of numbers. The time complexity for determining the subset of two ranges is constant, while it is linear for intersecting ordered sets.

3.5.4 Creating and accessing the object lexicon

The object lexicon, including retrieval methods, is generated by an off-line process which is not subjected to the Resource Principle. During generation, the objects are checked for well-formedness and validity. We will not describe the creation of the linguistic objects here; see section 3.4. Once they have been created, they are linearized into a series of bytes, and stored as Prolog terms in an output stream. We might store them in XML format as well, to make the lexicon application-independent.

A persistent object can be seen as a variable-length 'record', a concept from the Relational Model. The implicit 'record number' acts as the object's ID. The physical address of the object's first byte is kept in an external access table. The object lexicon and the access table have the same – large – number of entries. Storing them externally is in agreement with the Resource Principle. The access table holds records of a fixed size. As a consequence, it can be addressed by a function in working memory mapping an object with ID=N to the physical address of the Nth entry in the access table. The information to be found there in its turn refers to the physical location of the Nth object in the object stream. Thus, objects are retrieved by ID via an indirect addressing scheme in constant time, at the cost of accessing the disk one more time, of one function in working memory, and of one auxiliary access table in external memory. Prolog's built-in read predicate loads the objects as native Prolog terms.

Efficient data structures are the basis for efficient algorithms. Therefore, the objects need to be ordered by some criterion. A parser would benefit from an ordering on the phonological field for lexical look-up and tagging purposes, and would exploit a disk cache by accessing *all* objects with the same sur-

face form using a buffered read operation. A generator would benefit from an ordering on the most frequently used deep constraint for handling the agenda. As our semantic generator picks only *one* carefully selected object at a time, physical ordering does not seem to be beneficial. However, physical ordering by a criterion corresponds to a very compact index table for that criterion, because each entry can be represented by a range. Working memory is saved this way and intersection operations are sped up. It turns out that a physical ordering on syntactic type will prove helpful to the semantic generator. This exemplifies that a lexicon to be deployed both for a parser and for a generator needs to meet differing requirements. As the class of syntactical types is much smaller than the class of surface forms, a physical ordering on the former will yield fewer but bigger ranges, and as a consequence, a more compact index table than an ordering on the latter, saving more memory, in line with the Resource Principle.

3.5.4.1 Creating and accessing index tables for concept, type and phonology

For semantic generation, we require maximum performance on the retrieval of linguistic objects, specified by constraints on the features for semantic concept, syntactical type, and word form. For each linguistic object, its concept, type, and phonological features are stored in three auxiliary index tables in working memory. If an entry does not exist, it is created and added to the index table, together with the object's ID. If an entry already exists, only the ID is added. Number grouping is applied to the ordered (sub)sets of IDs as far as possible. Each combinatory type is hashed into a unique string instead of into an integer, because, theoretically, types are unlimited in size, which may yield too many collisions. Each word form and each semantic concept is hashed to a (non-unique) integer by applying a near-perfect hash function to their letters. The hash value is kept as small as possible by taking the letter frequency in Dutch into account (https://onzetaal.nl/taaladvies/advies/letterfrequentie-in-het-nederlands). This method is a variant of Huffman coding (Huffman 1952). Provisions are made for handling collisions which result from lexical ambiguities. In general, retrieving all IDs of objects that satisfy either the concept, type, or phonology constraint takes $O(1)$ time, when first-argument indexing is applied by the Prolog system to the respective index tables, and zero disk accesses. The index tables have a space complexity that is a linear function with a small factor – due to number grouping – of the number of templates. The index table on the feature that is used for ordering the objects has a space complexity that is a linear function of the number of

the feature's values. Three index tables are kept in working memory for fast access, in accordance with the Resource Principle.

3.5.4.2 Creating and accessing a meta-index table

Additionally, the semantic generator may select objects that are specified by constraints on other features. We do not demand maximum performance on queries of this kind. For each linguistic object, each value of each feature ever to be retrieved (not being concept, type or phonology) is stored, with its full path, in one auxiliary index table in working memory. This 'meta-index table' spans more than one feature. The IDs of all objects that specify the same value for some feature are grouped by number and stored in a metadata table, which is a stream. Additionally, the set of IDs of objects that do not specify any value for the feature is calculated and stored. The storage address for each feature-value pair is kept in the meta-index table. Retrieval of all IDs of objects that match one of these constraints takes $O(1)$ time, when first-argument indexing is applied by the Prolog system, and one extra disk access. The space complexity of the meta-index table is a linear function of the number of unique feature-value combinations, occupying only a fraction of the working memory, consistent with the Resource Principle.

3.5.4.3 General retrieval

Retrieval of linguistic objects is performed by executing a search task, specified by one or more constraints. When the search task is a graph – typically, an argument subgraph of some object involved in the semantic generation process – it is flattened into a series of constraints (paths). Each constraint is looked up in the appropriate index table. We distinguish between strict and liberal constraints, which demand objects to match the constraint explicitly, or allow objects that are unspecified for the constraint, respectively. Liberal constraints follow from graphs that allow under-specification. If a strict constraint is a restriction on semantic concept, syntactical type, or word form, the set of IDs of the objects satisfying the constraint is found immediately in the respective index tables. For other features, including liberal constraints, the set of IDs is found after issuing an extra disk access to the metadata table. In all cases, a constraint is replaced by a set of ID numbers corresponding to objects that are not inconsistent with the constraint. The objects that satisfy all constraints are identified by IDs that result from intersecting all sets of IDs. As the sets of IDs are ordered, determining their intersection is a linear function of the size of the biggest set. The function factor can be very small when ranges are involved, because only the edges of a range need to be inspected.

In general, however, the ranges get fragmented after a few intersection operations. Only the IDs in the resultant set of IDs may be accessed and loaded. The semantic generator randomly picks one of the IDs. Disk access is performed only for this ID, in order to locate and retrieve a complete linguistic object for further processing.
By applying directly accessible pre-compiled access and index tables, searching is reduced to looking up, giving a great performance boost; this contrasts with Prolog's own depth-first backtracking search algorithm, which is inefficient. In summary, finding one object, given its record number, takes $O(1)$ time and one disk access. Finding the set of IDs of objects satisfying a constraint on concept, type, or phonology only, takes $O(1)$ time and zero disk accesses. Finding the set of IDs of objects satisfying some other constraint takes $O(1)$ time and one disk access. Finding the set of IDs of objects that satisfy N arbitrary constraints takes $O(N)$ time and zero disk accesses.

3.5.5 Perspectives

The sum of all external index and external auxiliary tables is at present under 5 % of the total (externally stored) lexicon, measured in bytes. In the future we will explore how our way of organizing access to the stored lexicon can be maintained when the lexicon increases in orders of magnitude. It is expected that Prolog's internal management of huge numbers of index clauses is challenged with respect to access times. We leave that technical matter to compiler builders. More importantly, the lexicon, although it may take very large proportions, always stays finite. As the indexing tables are derived from the full-blown lexicon, there is always the possibility of having sufficient working memory available for accommodating them. Further, we want to investigate to what extent cognitive insights are reflected by this organization of the lexicon – a question which would not have been raised if we had applied standard techniques or third-party products.

3.6 THE LEXICON WHILE PARSING

In the DELILAH system, parsing is a process with three distinct ordered stages. Firstly, a chart parser or a backtrack parser determines the optimal analysis of the sentence in compliance with the categorial syntax. This is the combinatorial stage. Secondly, unification is exploited to combine graphs following the pattern of the syntactical combinatorics. This is the unification stage. Thirdly, the stored logical form is compiled out: the semantic stage.
The lexicon as a database is called for in the first and the second stages. In the combinatorial stage, syntactical categories are collected and tested for each word. These categories are specified in every lexical graph. For the combinatorics to work properly it is not mandatory but it is certainly efficient to maintain a register of combinatorial categories per word form, as an index to the lexicon. Note that the relationship between combinatorial categories and lexical graphs is not bijective: distinct graphs may share word forms and categories. In the first stage, the categorial index is the only emanation of the lexicon that is required. One or more syntactical analyses are chosen and these are transferred to the second stage.

In the second stage, every selected analysis is taken as a guide to unification. In bottom-up mode, appropriate graphs are taken from the lexicon and tested for unification. The lexical graphs are selected by their phonological and syntactical/combinatorial properties only. Successful unifications are stored in parallel, if necessary. At this stage, it is mandatory to keep track of argument subtemplates which are candidates for unification at certain points in the derivation. A transitive verb, for example, has two nominal arguments, and subject and object had better not be confused in unification. For this purpose, argumentative subtemplates (marked ARG) are uniquely indexed, and this index is reflected in the combinatoric category. For example, in (418) the index of the subject-argument reenters in the combinatoric category and in the Stored Logical Form; the index is highlighted in the following excerpt:

(420) *re-entering*

```
ID:A+B
...
SLF:{{[I&(B+J)#K,
       L&(M+N)#O,
       P@some^Q^and(quant(Q,R),laugh~[Q],event~[Q],
       entails1(Q,S),and(P,entails(Q,T)))&(A+B)#U],[],[]},
     and(and(and(experiencer_of~[U,K],
     some^V^and(quant(V,some, and(and(event(V),move(V)),
     theme_of(V,K)),goal_of~[V,O],entails1(V,incr),cause(Q,V)),
     entails(V,incr)),attime(U,W)),tense(U,past))}
...
TYPE:s_vn\0~[pp^0#B+M, np^0#B+Z, np^0#B+Y,np^0#B+J]/0~[]

ARG: ID:B+J
        PHON:G
        SLF:I
        SYNSEM:CASE:nom
               CAT:np
               NUMBER:sing
               OBJ:subject_of(A+B)
               PERSON:C
               QMODE:X
               THETA:experiencer_of
```

The indices are variables in lexical templates, but become instantiated under unification. They are pairs of natural numbers N1+N2: N1 is the second member of the higher index, N2 the first index of embedded arguments. Hence, chains of instantiated indices represent argument structure for the sake of anaphor resolution. In this way, the bookkeeping device of indexing argumentative subtemplates provides additional structural information.

When parsing, access to the lexicon is in real time. Every graph that may be useful is identified and found instantaneously, as described in the preceding section. The lexicon is supposed to be complete and indexed when parsing starts. As soon as a parse is selected on syntactical grounds, the relevant templates from the lexicon are retrieved and ordered according to the derivation of the chosen parse. Each template is selected on the basis of its phonological head – the value HEAD:PHON in the template – and its combinatorial category – the value of TYPE. These values are fixed in every lexical template. The phonological head corresponds exactly to the word form occurring in the sentence that is being parsed. The category corresponds exactly to the combinatory category that is applied for that word by the syntax in the present deriva-

tion. Below is the parse scheme of a simple three-word sentence. The relevant combinatory categories at word level are in bold face.

(421) Parsing *Jeroen wil werken* 'Jeroen wants to work'

(a) for all words in the sentence, retrieve their syntactic categories and make a chart

(b) choose the optimal parse(s) from the chart:

```
1-3+s\wh~[]/1~[]  jeroen wil werken
      1-1+np\0~[]/0~[]  jeroen
      2-3+s\0~[np^wh]/1~[]  wil werken
            2-2+s\0~[np^wh]/0~[vp^6]  wil
            3-3+vp\0~[]/0~[]  werken
```

(c) retrieve all lexical templates

```
[ ... HEAD:PHON:werken... TYPE:vp\0~[]/0~[] ...]
[ ... HEAD:PHON:wil... TYPE:s\0~[np^wh]/0~[vp^6] ...]
[ ... HEAD:PHON:jeroen... TYPE:np\0~[]/0~[] ... ]
```

(d) try to unify, according to the parse tree(s) of stage (b)

1. every retrieved *werken* template with every retrieved *wil* template
2. every unification resulting from step 1 with every retrieved *jeroen* template

(e) select the optimal unification result

Many of the lexical templates, though retrieved by their heads, will introduce constructions in the sense that the category and the template invoke other constituents, like the *wil* templates in (421). The procedure up to the selection of the optimal unification result is indifferent to whether the final semantics of these templates is compositional or not. Since every template carries its semantic receipt with it, unification is all that counts. For example, the light verb form *heb* 'have' of type `s\0~[np^wh]/0~[np^0]` introducing the *be hungry* construction unifies with the fully-fledged template for the NP *honger* 'hunger': its semantics comes along but is not represented at the top level. Step (421)(c), therefore, is perfectly general: in all cases, the constraints of phonology and applied combinatoric category suffice to retrieve all and only the relevant templates.