1. SYNTAX: the game of recursion and discontinuity

1.1 THE NEED FOR SYNTAX

Dutch is a natural language – to a certain extent. As such, it is torn by finiteness and two infinities. First, there is the finite set of all lexical atoms, the lexicon. For the present purpose, it is immaterial how we define the elements of this set, as words, morphemes or phrases. There has to be a finite set that is not an algebra: its members are not deduced but rather elected or assigned or proposed or enforced, but by no means are they theorems of a system. Without this set being finite, a language would be neither learnable nor decidable. Second, we have the lexicon's *Kleene closure*: the infinite set of all finite strings over that lexicon. Since there is no reason to assume a longest string, there are infinitely many of them, but each is of finite length. Third, there is the infinite set of finite strings over the lexicon that we take to be expressions of the language – the language in extension. Again, it is immaterial how we define this set – by well-formedness, by interpretability, by pragmatic criteria or any other sensible norm.

Language users are familiar with each of the three sets. They recognize the elements of the lexicon. They are able to produce and recognize lists of lexical items of their language without being seduced to interpret those lists as meaningful. They are able to recognize and interpret members of the distinguished set of (would-be) sentences. At the same time, there is no reason to assume that felicitous language use presupposes any awareness of the nature or extension of these sets. In order to describe the human language faculty, however, it is advisable to postulate the sets.

The basic fact about language-as-a-system is that the two infinite sets do not coincide: a language is a *proper* subset of the Kleene closure over its lexicon. No grammarian ever proposed any language to coincide with its Kleene closure. No author of formal languages ever proposed an *anything goes* language. Such a language would be bound to challenge meaningfulness: meaning results from distinction and distinctivity, but in the Kleene closure all strings are born equal. Only length or the occurrence of individual atoms could count as contributions to meaning. Since every atom occurs in any order with other atoms, it is hard to see what an individual atom in a random string could contribute to its specific meaning. Of course, the length of a string could count as meaningful. Since the length is a fixed extensional property of a string, length can only be meaningful in the way integers are meaningful. Numbers, however, are meaningful in a way languages cannot afford to restrict themselves to. Numbers are rigid designators. Tomorrow, they will refer in exactly the same way as they did yesterday. They are *one-world* designators, and do not provide content outside that world – unless we use numbers in other languages that are not rigid. That leaves only the possibility for strings in the Kleene closure to carry meaning randomly or by oracle. It is hard, however, to look at meaning as an accident, not steered by material properties of the expression. It would make your present enterprise of reading this text vacuous. The famous but not uncontroversial concept of *compositionality* of meaning represents an effort to explain why that enterprise is not completely vacuous (even if the concept cannot be attributed to Gottlob Frege, as Janssen (1986) convincingly argues). Accidental meaning would not convey information, just sound.

Almost by definition, then, languages are 'smaller' than their Kleene closure. Since we have no reason to assume that the order of infinity between the sets is different – as is the case with *e.g.* the sets of rational and real numbers – 'being smaller' here must be approached stochastically: the chance that a string, randomly chosen from the Kleene closure, is also in the language is less than one. As a matter of fact, the chance is close to zero.

The selection procedure that (dynamically) identifies that small island of language in the sea of the Kleene closure is called grammar. It is a characteristic function over that closure, the function dividing the sheep from the goats. Grammar is the *theory* of the language. Most linguists assume that the function is operative in a man's brain, but they entertain some disagreement as to its whereabouts and 'wherefroms'. The function, however, is most certainly required in a computational system – if that system aims at interpretation. Of course, a lot of language-related tasks could be performed without access to an intensional description of the language; you don't need grammar to

count words or to find strings. But if meaning comes into play, distinction is required. No automaton can reliably assign meanings to sentences and not tell interpretable from uninterpretable ones, for the same reason that a computer that adds * and #, can hardly be trusted when adding 3 and 4. If anything is meaningful, meaning evaporates. Language lives on selectivity.

We now have the following situation: a finite lexicon L, its Kleene closure L* = $\{<w_1 \ldots w_n> \mid w_i \in L\}$ and natural language NL $\subset$ L, where NL is defined by a Boolean function f_{NL}, its grammar. NL is not necessarily well-defined. Its characteristic function may be fuzzy, and certainly a family of languages that intersects with NL can be defined or, more precisely, a family of languages that shares infinite subsets with NL. This family covers the varieties of Dutch one might want to focus on. It is more intriguing whether we could find natural languages in the complement of NL in L*, in the set L* - NL: languages that live on the same lexicon but would not share any expression with NL or, more precisely, that share at most finite subsets with NL, *e.g.* a set of one word expressions. Probably, no such languages would be accepted as possible natural languages. For formal languages, though, complementary languages can be defined and exploited in simple ways. We might want to have, then, a theory that determines which subsets of L* qualify as natural languages. That theory is the *theory of grammar,* embodying the classical Chomskyan concept of possible natural language. We won't touch upon that theory here directly, as we are dealing mainly with the theory of Dutch. We assume, however, that the theory of grammar will cover our analysis of Dutch. In defending that 'local' theory, we will have to refer to more general points, now and then, in particular when matters of restrictivity are at stake.
Grammar must be finite to be applicable. That is why a grammar cannot be identified with the language in extension. We (humans, computers) need a finite, intensional description – in whatever form: phrase structure rules, algorithms, parsing routines, integrated circuits – to handle the inevitable infinity of language. Exploiting finite means to deal with infinite sets provokes recursive use of those means. That is: infinite, structured languages may have sentences that show bracketing patterns like

(2) $\ldots[_x \ldots [_Y \ldots [_X \ldots] \ldots] \ldots]$

In these patterns, a certain type of string occurs within an instance of the same type, with or without – mostly with – intervention of some other type or category. In many natural languages, all 'major' categories (sentences, nominal constituents, prepositional constituents, verbal constituents) have

this property of re-entrance within themselves. Crucially, the pattern postpones the full interpretation of the upper occurrence until after the interpretation of the lower occurrence. And that is where syntax comes in: the syntax specifies which phrase depends on which phrase. Syntax is the carrier of recursive embedding and dependency. It defines a finite family of categories or equivalence classes and a finite class of relations between them. It assigns lexical atoms to at least one class or category. It guarantees that every string *of the language*, no matter how long or complex, can be divided into representatives of those categories. The process of that division is called *parsing*, and it is one of the more interesting ways to execute a characteristic function over the set of all strings.

This is the – by now – classical approach to grammar, as practised in that branch of mathematics that studies formal languages and automata (cf. Hopcroft *et al.* 2001). It relates grammar to computation: an explicit formal grammar determines a class of automata with respect to its use of resources, and every computable problem can be represented as a characteristic function over a set of sequences. The notion of automaton is used here in the mathematical sense, as a unit of data, processing, control, and storage, *aka* the Turing machine. The first track of the relation has become canonised as the Chomsky hierarchy, relating properties of languages, grammars and automata. The second track is due to digitalization: all automata can be defined in a digital language, with a grammar; the sentences of that language are interpretable as automata. In that sense, the circle is closed: languages have grammars, grammars determine automata – they can be written as programs – and automata are defined in languages.

In this view, syntax is the backbone of grammar because recursion resides here and recursion mediates between finiteness and the infinities of language. Moreover, in the ever-swelling literature on language evolution and animal communication, recursion has become prominent in the field of cognition after Hauser and Fitch (2003) reduced the exclusivity of human language to just that. Independent of its cognitive status, though, it must be something other than just "... the product of a given theory designed to account for language structure" (Heine and Kuteva 2007:265). To see why, consider this. Language is meaningful *because* not every conceivable string is meaningful. To tell the infinite number of meaningless strings from the infinite number of meaningful sentences, and/or determine the meanings of the meaningful ones, we need a system. That system can be learnt and/or acquired, and must be finite, for that reason. For the system to be finite and still be effective and discriminatory, re-use or re-entrance of resources (cat-

egories, constructs, embedding procedures) is essential. That is recursion: linking phrases to each other in an asymmetrical style, *ad libitum*. Recursion offers a framework for making expressions semantically dependent upon each other, as well as for thematic relations, referentiality, intensionality, mood, aspect, and so on. This view leaves open many ways of embedding expressions semantically into each other. Physical – prosodic – embedding as in (2) is just one of them. Anaphora or kataphora is another: the semantic effect of (2) – the interesting effect – can be mimicked by a structure like (3), where the indices indicate co-evaluation.

(3) $\ldots[_X \ldots it_j \ldots] \ldots \quad [_y \ldots[_X \ldots] \ldots]_j$

Here too, the valuation of the 'highest' X is dependent on the interpretation of the lower X. For example, the following two texts are idempotent. They both subsume meaningful expressions under each other.

(4) The man who read a book said that the river was too wild
(5) There was a man. He read a book. He said this. The river is too wild.

Their logical forms must be the same, as their entailments converge: in both texts, the speaking subject is described as a man reading a book, and the wildness of the river is not presented as a fact, like the speaking event, but is attributed to that man.

Anaphora, realized by copying, binding or co-reference, places the antecedent under the semantic regime of the sentence containing the pro-form. As a matter of fact, anaphora and kataphora do *overtly* what in the semantic representation of a complex sentence is conducted *covertly* by reification and binding; there, *logical* anaphora are called for. Overt anaphora and kataphora enforce recursion.

In the preceding examples, the semantic dependencies are marked by syntactic means: categories, valency, embedding, linear order. Thus, interpretation is syntax-driven, and the interpretability of language is anchored by the recursivity of syntax. That turns the syntax in a grammar into a crucial factor for the computation of meaning. In this vein, it cannot be accidental that an explicitly semantic and wide-coverage grammar model like that of Sag *et al.* (2003) sails under *Syntactic Theory*. And consequently, the nature of the syntax is part of the message. The system presented below functions well for parsing and generating Dutch. It is neither *the* syntax of Dutch nor *a theory of* syntax, but describes a viable and operational way of demonstrating what our enterprise is about: relating meaning to form in a domain where form is extremely discriminating and meaning is far from trivial.

1.2 FORMS OF DUTCH

As a natural language, Dutch shows signs of syntax. This section presents a very concise characterization of those aspects of Dutch syntax that determined the nature of DELILAH's grammar. It is neither complete nor balanced, as it focusses on those properties of Dutch phrase-structure formation that resist uncomplicated accumulative combinatorics.

Dutch is an SOV language in the following sense: the nominal and prepositional objects of a verb precede it. Verbal and propositional complements follow the verb. Prepositional complements can occur to the right of a verb, and verbal complements may occur to the left. Whether SOVness is a derived or a fundamental property of languages is not relevant here.

Some verbs selecting infinitival complements must or tend to cluster with the verbal head of that complement. This may lead to multi-clusters of verbs. Dutch is infamous for maintaining a particular order in that cluster: the selecting verb can or must precede the head of its complement. In multi-clusters of verbs that also select nominal complements, this order gives rise to crossing dependencies between nominal and prepositional objects and their respective verbs. Here is an example, inspired by Evers (1976).

(6) … dat Jan de man Marie een koe zag helpen leren dansen
… *that Jan the man Marie a cow saw help teach dance*
… 'that Jan saw the man help Marie to teach a cow (how) to dance'

The verbs cluster, and this cluster is preceded by their aggregated nominal arguments in the same order: *zag* selects *Jan*, *helpen* selects *de man*, ... and *een koe* controls *dansen*. Evers observed that in German verbs cluster too, but tend to do so in reverse order. A major difference between the Dutch and German orders is that in Dutch the head of the sentence – the verb dictating argument structure at top level – occurs more to the left. For parsing, this may be an advantage, since the sentence structure is clarified at an earlier stage. In the German order, the verb cluster must be processed as a whole before any clue as to the sentence structure and the status of the argument is revealed. The Dutch order may be more complex from the point of view of memory management (cf. section 1.8), but the more sophisticated appeal to random access memory – *i.e.* exploiting memory as it comes – may improve parsing, as one can conclude from observations reported in De Vries *et al.* (2011).

This state of affairs is rather rare among the languages of the world. It has been used to challenge the idea that the grammar of natural languages be context-free. The point for Dutch was made by Huijbregts (1976). The concept of context-freeness amounts to the claim that the internal structure of categories in natural language is not essentially determined from outside, and, equivalently, that natural languages can be parsed – and generated – with a memory management that is more restrictive than random accessibility. The discussion on the status of crossing dependencies and related phenomena has been part of the genesis of Generalised Phrase Structure Grammar; *cf.* Pullum and Gazdar (1982). Since crossing dependencies are what made Dutch famous among linguists, this phenomenon influenced our choice of the particular categorial grammar exploited in DELILAH. Considerations relevant to this choice are discussed by Stabler (2004) in an extremely useful exercise on grammatical capacities beyond context-freeness. The topic will be further discussed in section 1.8.
Crossing dependencies and verb-clustering are important sources of *discontinuity* in the syntax of Dutch: the phenomenon that two phrases that establish an essential grammatical relationship occur separated from each other rather than adjacent in the sentence. Of course, a verb and its object are striking examples of such a pair themselves. But adverbial or propositional adjuncts can also occur amidst phrases to which they cannot be linked directly:

(7) … dat Jan de man misschien Marie een koe zag helpen leren dansen
… *that Jan the man maybe Marie a cow saw help teach dance*
… 'that John maybe saw …'

Misschien 'maybe' is linked to the verb complex and modifies some event, but occurs here at a seemingly random position in the sequence of nominal phrases. Its position may have an effect at the level of information structure, though, but that is not well established. Verb clustering provokes this type of discontinuity in the *Mittelfeld*: verb clustering 'obscures' constituency, and consequently the semantic or syntactic target of adjuncts has to be determined and/or reconstructed by more complex manoeuvres than just concatenation.

Just like some other SOV languages, Dutch has a verb-second effect. In the absence of a complementizer, a finite verb or auxiliary – in Dutch, finiteness is exclusively morphologically marked – can or must occupy that position, thereby determining the status of the sentence.

(8) Niemand heeft de dronken oom durven corrigeren
nobody has the drunken uncle dare correct
'Nobody dared to correct the drunken uncle'

(9) Gaat het mis, treedt de minister af
goes it wrong, steps the minister down
'If it goes wrong, the minister will step down'

In the first clause of (9), the finite verb is in complementizer position, marking it as a (conditional) subordinate clause. Verb-second (or verb-first, for that matter) also enhances discontinutity: it can strand adverbial modifiers and may separate a verb from its immediate objects.

The complex structure of the Dutch *Mittelfeld* allows for complex forms of coordination. Cremers (1993a) argues that in Dutch almost any position in a sentence can give rise to coordination, including coordination of non-constituents. Thus, coordination is a source of discontinuity; in the following example, the verb *schilderen* 'paint' is cruelly separated from its intended objects *de auto* and *de fiets:*

(10) Ik had de auto met jou blauw en de fiets met haar groen willen schilderen
I had the car with you blue and the bike with her green want paint
'I wanted to paint the car blue together with you and (to paint) the bike green together with her'

Complex coordination is not necessarily the same as ellipsis, though ellipsis does involve coordination, in our analysis. Ellipsis, however, is the kind of complex coordination where the left context of the coordination element must be sentential.

(11) Ik zal Jan een laptop geven en jij Peter
I will Jan a laptop give and you Peter
'I will give Jan a laptop and you, Peter'
(12) * Ik zal Jan en jij Peter een laptop geven
I will Jan and you Peter a laptop give

Complex coordination in Dutch influenced the set-up of the DELILAH syntactic architecture to large extent, as will be explained in section 1.7.4.
A standard form of discontinuity entertained by Dutch is movement of constituents to the left periphery of a sentence, the position also known as *SPEC CP.* As usual, movement here is a metaphor for one constituent governing or controlling two non-equivalent positions in the sentence structure. Question words, relative pronouns, arguments and adjuncts, even verbal complements, occur at *SPEC CP* – obligatorily or optionally. Though they may come from far and deep, their origins are not arbitrary, for the sake of wellformedness and interpretability. As a consequence, Dutch defines islands of all kinds from

which a constituent cannot escape, or only under strict conditions (cf. Honcoop 1998, Szabolcsi 2006). These islands may be syntactic, lexical and/or semantic. Here are some examples, where the intended rightmost position is marked *t*.

(13) *Welke prinsen* denk jij dat *t* de receptie zullen bezoeken?
which princes think you that the reception will visit
'Which princes do you think will visit the reception?'

(14) * *Welke prinsen* denk jij dat de receptie die *t* bezochten in de kerk was?
Which princes think you that the recepetion that visited in the church was

(15) Elke sonate *die* ik *t* speelde, wekte ergernis
Every sonata that I played raised annoyance

(16) * Elke sonate *die* ik een deel van *t* speelde, wekte ergernis
Every sonata that I a part of played, provoked annoyance

(17) *Mijn vader* heeft zij *t* niet gekend
My father has she not known
'My father she never knew'

(18) * *Mijn vader* betwijfelde zij of *t* kon lezen of schrijven
My father she doubted whether could read or write

(19) *Aan het dak* hing mijn vader hammen te drogen *t*
At the roof hung my father hams to dry
'From the roof my father hung hams for drying'

(20) * *Aan de wilgen* hing de grijsaard de viool *t*
In the willows hung the old-man the violin
(compare: * The bucket he kicked)

(21) *Met onwaarschijnlijk geweld* raasden de drie orkanen *t* over de eilanden
With incredible violence raged the three tornados over the islands
'Incredible violently, the three tornados raged over the islands'

(22) * *Met onwaarschijnlijk geweld* raasde geen van de orkanen *t* over het eiland
With incredible violence raged none of the tornados over the islands

The grammar of Dutch has to account for such barriers to interpretation. And since recursion seems to be the issue here, syntax must carry the load.
One could easily come up with many more intrinsicalities of Dutch syntax. As a matter of fact, Dutch is one of those languages the structure of which has been studied and documented extensively. Above, only a few of the more complicated patterns are set out. An extensive, authorized description of Dutch is available at http://ans.ruhosting.nl/. A less descriptive, more theoretically motivated 'modern grammar of Dutch' is being developed at present by Hans Broekhuis (and others at http://www.taalportaal.org). An intriguing side-effect of trying to make Dutch syntax work for interpretation is that one cannot afford not to solve problems as they occur; otherwise, it will immediately corrupt the overall result. Here is a good example. Dutch has a class of pronouns that act as left complements of prepositions, but are not necessarily

adjacent to them. So, the infamous *er* can occur in any of the marked positions in the following sentence, but only in one of them at the same time, but it can always only be interpreted as the anaphoric complement of *over*.

(23) … dat ik (*er*) mijn vader (*er*) nooit (*er*) met mijn moeder (*er*) over heb horen spreken
… that I (it) my father (it) never (it) with my mother (it) about have hear speak
… 'that I never hear my father talk to my mother about it'

None of these positions is marked, except that *er* does not intrude into 'closed' constituents. Moreover, the same pronoun *er* – or a look-alike – can also occur as an existential marker, as a place adverb and as a partitive pronoun, and in all sorts of combined functionality. Although interesting aspects of *er* are revealed by Bennis (1986) and Van Riemsdijk (1978), one has to invent syntax that is not available anywhere in order to decide for a given sentence which *er* is at stake and whether and how it can or must be interpreted – not to mention how to accommodate *er* in generation.

1.3 THE TASK FOR SYNTAX

Having established the need for a syntax of Dutch, it is wise to reflect upon its task. Syntax is not all of grammar. In our concept of a lexicalized grammar, syntax is that dynamic module that directs unification of complex symbols. Unification of complex symbols is the core business of grammar, both computationally and cognitively. In a human's brain as well in the digital lexicon, much more information is stored than can be covered by syntax. Complex symbols, feature structures, signs, templates or whatever we call our information units, are supposed to accommodate all knowledge of language to the extent that the knowledge can be attributed to phrases (localized) and is expressible. Knowledge of language extends far beyond the combinatoric mechanisms (routines, principles, parameters, ...); knowledge of language is also about meaning, about sound, about use, about style, about semantic and pragmatic fields. Basically, syntax is only arranging the concatenation of phrases, and in this respect quite different from semantics, which is governed by functional application.
Yet, syntax is far from trivial, for the simple reason that it has to be extremely discriminating while steering the powerful engine of infinity, recursion. Sim-

plistic syntax over-generates: it allows more unification to be tested than efficient processing and subtle interpretation can afford. Too restrictive syntax under-generates: it may miss meaningful combinations of signs. Good syntax is in between. It defines the combinatorial space for a natural language. To the extent that it does so adequately and transparently, it contributes to the notion of possible language, for example by excluding permutation closure.
If syntax is to direct unification, unification and the data structures it operates on have to be defined carefully. Here we discuss the major outline. In the chapter on the lexicon the complex symbols are presented more in detail.

Our data structures are feature trees: acyclic connected directed graphs in which each node labels an attribute and its outgoing edges define the value for that attribute in that graph. Technically, this concept is not essentially different from the feature structures that characterize Head-Driven Phrase Structure Grammar (HPSG; Pollard and Sag 1994). The main difference, as will be illustrated in chapter 3, is that we do not impose any regime on the nature of features, and that we allow feature values to be plain variables. More esoterically, we consider feature graphs to be *the* objects of combinatoric manipulation, rather than 'just' descriptions of those objects, which Kracht (2004:ch. 6) takes HPSG attribute-value matrices to be. Furthermore, we call a feature graph a *template*, because DELILAH exploits feature trees not only for grammatical unification, but also for the dynamic construction of the lexicon. Here is a representation of a simple template, one of the lexical templates of the finite verb *werkt* 'works'.

(24) *a lexical template of* werkt *'works'*

```
ID:A+B
HEAD:CONCEPT:work
     PHON:werkt
     SLF:work
     SYNSEM:ETYPE:event
            FLEX:fin
            NUMBER:sing
            PERSON:2
            TENSEOP:at-pres
            VTYPE:nonacc
PHON:C
PHONDATA:lijnop(werkt,A+B,[arg(left(11),wh,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
       EVENTVAR:J
       EXTTH:agent_of~[A+B,G]
       PREDTYPE:nonerg
       SUBQMODE:L
       TENSE:tensed
TYPE:s\u~[np^wh#B+F]/u~[]

ARG:   ID:B+F
       PHON:D
       SLF:E
       SYNSEM:CASE:nom
              CAT:np
              FOCUS:focus
              NUMBER:sing
              PERSON:2
              OBJ:subject_of(A+B)
              QMODE:L
              SUBCAT:pron
              THETA:agent_of
```

The template is hanging from one (hidden) top node, *template*, denoted by TOP. From there, vertices lead to nodes ID, HEAD, PHON, PHONDATA, SLF, SYNSEM, TYPE, and ARG. There is no ordering between these vertices. The node HEAD has outgoing vertices to CONCEPT, PHON, SLF, and SYNSEM. The nodes PHON and SYNSEM under HEAD are not equal to the nodes PHON or SYNSEM under the top node: each node is uniquely defined by a path from the top node. Thus, we have three different nodes *phon* in (24): TOP:PHON, TOP:HEAD:PHON, and TOP:ARG:PHON. They are separate, distinct, unique and independent.

A node without outgoing vertices is a *leaf* or *terminal*. It represents values by definition, and may be variable or instantiated, and either complex or simple. The value for the node TOP:SLF is a complex structure itself – not a graph. The value for TOP:TYPE is a complex category constant. The value for TOP:ARG:SYNSEM:QMODE is a variable, identical to the value for TOP:SYNSEM:SUBQMODE. For each attribute, the set of values it may assume is defined and finite. Thus, a template is finite, by definition.

Templates store information about phrases, in relation (attached) to one lexical part of the phrase. From this perspective, (24) specifies a sentential phrase headed by *werkt*. Consequently, templates specify information on three levels:

(a) attributes of the phrase as a whole
(b) attributes of the (lexical) head
(c) attributes of the non-head parts of the phrase.

The properties of the head are specified as values of the node TOP. The properties of the non-head parts of the phrase are specified as values of nodes TOP:ARG. Of these nodes there may be several, but each TOP:ARG node is uniquely indexed. All other paths in the template specify values of the phrase as a whole. The information at the three levels is not necessarily disjoint. As a matter of fact, attributes sharing values at different levels of the templates specify networks of local dependencies, like agreement. In (24), all variables that occur as (part of) values occur more than once, thus specifying agreement.

Two templates unify when their specifications at relevant attributes are compatible. The unification amounts to merging the templates and is of course a template itself. Technically, the unification of two templates t1 and t2 is a template t* which subsumes both t1 and t2; a template subsumes another if it is more specific, being obtained from the other by valuation of variables. The unification result therefore is at least as specific as each of the entry templates. Unification operates point-wise. Singular attributes are compatible if they are identical. For each specified pair of attributes, there is a check whether the values are compatible. A variable is compatible with anything, and is instantiated at unification. Two constants are incompatible unless they are equal. Two structures are compatible if they match.

The syntax determines under what conditions templates – phrasal descriptions – can or must unify. As the templates are specified at different levels, the syntax is sensitive to the internal structure of the templates. In particular, the syntax determines which part of a template should unify with another

template. More precisely: if the syntax sends templates A and B to unification, it arranges unification of A with one particular subtemplate TOP:ARG of B by replacing ARG with the top of A. The other parts of B will be affected only by the possible instantiation of variables resulting from this unification.
A syntactic category, then, is a projection of the internal structure of a template. Every template is specified for a literal expressing its syntactic status; the values here are, *e.g.* S, NP, and VP. They label the phrase as a whole. If the template has attributes of type *top:arg*, these attributes may also carry syntactic labelling. If so, that label occurs in the syntactic category, exactly once. The syntactic category of a template, therefore, reflects the label of the phrase as a whole and of the labelled *arguments* in the template. It is specified as the value of the attribute TOP:TYPE. As the template is assigned to the head of the phrase, the syntactic category, being a value in that template, can also be seen as being assigned to that phrase. In that sense, the finite verb *werkt* in (24) *is of* or *belongs to* the category `s\u~[np^wh#B+F]/u~[]`.

The category relates information on the internal structure of the template to conditions under which it can be unified. Abstracting from the label B+F, which solely introduces a template internal index, the category `s\u~[np^wh#B+F]/u~[]` for *werkt* expresses the following statements:

(a) the full phrase has syntactic label S "`s `"
(b) one attribute TOP:ARG is labelled NP "`[np `"
(c) there are no other specified arguments "` ], []`"
(d) the subtemplate labelled NP can unify with a template labelled NP occurring to the left of *werkt* "`\ `"
(e) this unification is submitted to a package of conditions *wh*, specified in a syntactic rule of category merging "`^wh `"
(f) the template was not subjected to leftward or rightward unification "u"

In the same vein, a syntactic category `vp\a~[np^0]/u~[s_sub^6]` summarizes a template with phrase label VP with an argument labelled NP that can unify leftward under rule condition 0, with a completed leftward unification for another argument indicated by flag *a*, with an argument of label S_SUB that can unify rightward under rule condition 6.

The arguments that are open for unification at each side are organized as a list. An argument is open for unification only if it is in the list.

The rules of syntax specify exactly and completely the conditions under which unification can take place. They are triggered by the label of arguments and by directionality. Every rule of syntax mentions three syntactic categories:

(a) the primary category, introducing the argument to be unified and its subtemplate
(b) the secondary category, introducing a co-labelled counterpart to the argument
(c) the resulting category: the category of the merged template.

Since the rules are sensitive to labels and directionality, the syntax resulting from these rules is essentially *anti-symmetric.* Interchanging the primary and secondary categories yields no merge, or a different one. Thus, the unification process is anti-symmetric too. In this sense, our grammar complies with the fundamental propositions of Kayne (1994).

For this triple, the rule specifies a complex set of constraints:

(a) the syntactic structure of the primary template (the one with the specified argument) at unification
(b) the syntactic structure of the secondary template (the one co-labelled with the argument) at unification
(c) the syntactic structure of the unified and merged template.

The first two can be seen as preconditions of the unification. The third part of the package summarizes the output of the unification. The preconditions on the primary and secondary categories define a particular *cancellation* mode. For each argument in a syntactic category this cancellation mode is explicit. It is the main combinatory specification of a syntactic category.

Here is the general format of a syntactic rule, specifying leftward cancellation; the rightward case is similar. The operation is defined relative to the cancellation mode *i*; X⊕Y generalizes the appending of lists X and Y as XY or YX, depending on the cancellation mode.

(25)

primary category:	PRIM \ PLF~PLA / PRF~[SEC^i|PRA]
composition operator:	$\otimes_i$
secondary category:	SEC \ SLF~SLA / SRF~SRA
resulting category:	PRIM \ PLF~(SLA⊕PLA) / RF~(PRA⊕SRA)

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category);
^i: cancellation mode; *RF:* some flag determined by *i*

The constants in the formulation of a rule are these, labelled as properties of the syntactic formalism:

(a) the label of the primary category is the label of the resulting category (*output*: conservation of head)
(b) the label of the secondary category occurs as a label in an argument stack of the primary category (*input*: well-foundedness of merge)
(c) only one argument is affected by the merge (*output*: exclusiveness of merge and directedness)
(d) all non-affected arguments of the primary category and the arguments of the secondary category are assembled in the resulting category with the same labels and cancellation modes (*output*: conservation of arguments)
(e) in the resulting category, the left stacks and the right stacks of the input categories are assembled to a left stack and a right stack, respectively (*output*: conservation of direction)
(f) argument stacks are appended and the order of arguments in a stack is unaffected (*output*: stack integrity)
(g) the flag of the assembled stack for the passive direction in the resulting category equals the flag for that direction in the primary category (*output*: passivity)

Normal business in a syntactic rule implies the following properties:

(h) the label of the merged argument does not occur in the resulting category (*output*: cancellation)
(i) the target argument of the unification is at the top of its stack (*input:* linear ordering)
(j) the assembled stack for the active direction in the resulting category is marked for being affected (*output:* affectedness).

'Normal business' means that deviation from these practices marks that rule as exceptional and a candidate for reconsideration. For example, the rule han-

dling 'floating' *er* in Dutch (see (23)) may exceed normal business, as *er* placement is hardly understood.

Finally, the rules are parameterized with respect to:

(k) the labels of non-affected arguments and the label of the primary category (*input:* labelling)
(l) the arity of stacks (*input*: arity)
(m) the flag of stacks in the primary and secondary categories (*input:* flag).

The instantiation of these parameters defines a certain merge mode. So, syntax specifies which templates should be sent to unification on the base of their internal structure.

The parameters of the syntactic rule determine their variety. All parameters are finitely valued: there is a limited number of syntactic labels, templates have only finitely many arguments and stacks have a restricted number of flags. Consequently, a natural-language syntax along these lines can only have finitely many rules. As a matter of fact: syntactic labels are very restricted in number, and so is the argument complexity of templates. Therefore, the number of syntactic rules that can be specified is very limited, the main factor being the arity of argument stacks in the specification. In its present state, DELILAH's syntax comprises about ten rules *per* direction.

1.4 THE LOGIC AND THE ALGEBRA OF LISTS, FLAGS, TYPES AND MODES

1.4.1 Categories and complex symbols

A category in a categorial constructive unification grammar like the one presented here has many tasks to fulfill. First of all, it embodies an agenda according to which the rules of grammar – the modalized instances of the rule of cancellation and composition – can operate. Secondly, it predetermines linearization of phrases in a sentence: the structure of the category imposes strict conditions on linear order. Thirdly, it defines equivalence classes of phrases

with respect to the grammar. And, fourthly, it represents the structure of lexical templates with respect to unification. As an example of this multi-tasking, suppose that the grammar-in-action, say: the parser, runs into the following category:

(26) pp\0~[n^0]/1~[np^0]

It tells the parser that at least two other phrases have to be found for this category to be satisfied, that one is a noun-type phrase to occur to the left of the other, a phrase with a noun-phrase-type of distribution, that the phrase carrying the category is neither lexical nor completely saturated, that the template which it encodes has two sub-templates which are subject to two different unification acts, and that for the categorial hypothesis – if not *pp* – to be met, at least one prepositional phrase must occur as an argument. Thus, the category broadcasts information essential to grammatical dynamics.
In categorial grammar, categories carry *all* combinatory information feeding into the grammar. For this reason, the categorial formalism is the quintessence of the grammatical message. In exactly that sense, categorial grammar is mathematical: grammar amounts to the manipulation of symbols according to an established protocol, which as such may reflect fundamental theses on the nature of the game. Unfortunately, the grammar we present here deviates from more or less canonical categorial set-ups, established *e.g.* by Moortgat (1988), Van Benthem (1991), Morrill (1994) and Carpenter (1997). On the other hand, more recent developments of the categorial practice in Moortgat (1998), and Baldridge and Kruijff (2003) indicate some convergence with the proposals of Cremers (1993a), which underlie the system presented here, in particular with respect to fine-grained modalities for composition. Still, the formalistic nature of any categorial approach to natural language requires an in-depth account of its architecture. That is what we are trying to achieve in this section.

It is of some importance to see that not all distributionally relevant aspects of words and phrases can be encoded in the combinatory category, and that a certain sense of arbitrariness is unavoidable. For example, the notion of *np* underspecifies some distributional aspects of the class of nominal constituents, for example with respect to case, number and definiteness. What, then, is the difference for a phrase between being nominal and being plural? The answer is relatively clear. In all contexts, being nominal is a required property – a phrase cannot be underspecified for category and still be rightfully selected; being plural may or may not be relevant for the combinatorics, depending on contextual requirements. This difference is a good reason to leave number to

the unification process, which is context-dependent by definition, and which leaves room for all levels of specification. In the same vein, certain less classical distinctions may make it to the combinatory level. In Dutch, for example, non-finite verb phrases may occur with and without *te*, a preposition-like and meaningless particle. Selection of verb phrases is sensitive to this particle: *willen* 'to want' takes only *te*-less vps as a complement, whereas *proberen* 'to try' exclusively selects vps headed by *te*. In no situation is *te* irrelevant. As a consequence, it is combinatorially relevant to mark a vp for having *te*. In the Delilah grammar for Dutch, we thus have two distinct atomic types: *vp* and *vpt*, for it is no use having unification decide – costly – on predictable combinatory properties.

Also, word order variation may have to be encoded at the level of categories. Again, the criterion resides in selection: if the internal word order of a constituent determines its candidacy for combinatory processes, that word order is better reflected in category labeling. Now, Dutch is a verb-second language, and the position of the finite verb determines the status of the sentence. Thus, we have distinct sentential categories for distinct positions of the finite verb.

The set of categories that one needs can only be determined empirically – by experimenting with how to get a semantic grammar to work properly, *i.e.* to interpret all and only the well-formed strings over a lexicon. Yet, we do not abandon the idea, inherent in the Polish origins of categorial grammar, that there is a class of elementary types – typically two: one for names and one for sentences – and that other categories denote functions over the domains of these elementary types. That is, we assume that all syntactic categories denote (complex or composite) functions over elementary domains, in short: that nps are of type *ett*. The (semi-)compositional semantics of our system is essentially *typed* (see chapter 2). We do not assume, however, that the syntax of Dutch must be grafted on to the internal typological structure of the categories needed. The reason for this divergence between syntactic and semantic labeling is typically pragmatic: there is no bijection between types and combinatory classes. This point was already made clear in Montague (1972) – though the motivation there was not syntactic subtlety. It is a fact of life that not every syntactic property is determined by the typological structure; many, maybe even most, are not. Often, the syntax does not exploit the semantic fine-structure of category: as was illustrated above, verb-placement is syntactically big business but it is semantically vacuous. This type of tension between combinatory and semantic interests makes the grammar of natural languages worth pursuing.

The manipulation of categories, to be pursued in this section, has one single goal: arranging the unification of complex symbols – or *signs* – in such a way that the right interpretation ensues. All interesting information – including the categories steering the process – is in these complex symbols and is treated conservatively in the unification process. The syntax as such does not add to that information. It just controls it. This control, however, is the only claim to effectivity and efficiency we can put forward. The ambiguity of natural language is overwhelming. Unification is too costly to have search space reduced while we are waiting. It is the syntax that has to select and prepare felicitous analyses; it is the syntactic routine that makes language work.

1.4.2 Basics

The syntax depends on three well-defined sets:

(a) a finite set *Types* of types
(b) a finite set *Mods* of modalities
(c) a set *Cats* of categories

The set Types consists of literals denoting syntactic and semantic equivalence classes over lexical phrases. The only restriction imposed on this set is finiteness. At least one of the types is marked for sentencehood. One may reduce the number of sentential types to one, but language calls for distinguishing propositions from questions and imperatives. A typical set of types may consist of literals like `s` for propositional sentences, `q` for questions, `np` for names and nominal quantifiers, `vp` and `ip` for tensed and untensed verb phrases, and so on. Clearly, the distribution of types over the grammar expresses our insight into the phrasal tiers, but the types themselves lack any formal content.

The set Mods of modalities is also taken to be a set of literals disjoined from Types. Modalities stand for modes of merging categories. They are defined pointwise. Since the number of merge modes will turn out to be finite, Mods is finite too.

The set Cats of categories holds the basic data structures of the grammar. Every category is a triplet <Head, LeftArguments, RightArguments>. The head is a single, bare member of Types.
LeftArguments consists of a pair <Flag, LeftArgumentList>; Flag is a label, holding information as to the present state of LeftArgumentList. RightArguments is constructed accordingly. Both the left and the right argument lists consist of

a finite number – possibly zero – of pairs <Type, Modality> in Types × Mods. The left arity of a category is the number of pairs <Type, Modality> in LeftArgumentList. Equally, the right arity of a category is the number of modalized types in the RightArgumentList. The arity of the category plain is the sum of its left and right arities. Basically, a category should be seen as an *n*-place function mapping categories onto categories, with *n* reflecting its arity.
For notational convenience, categories are written in the format

(27) Head\ LeftFlag~LeftArgumentList / RightFlag~RightArgumentList.

Non-empty argument lists are written *[Type1^Mode1, Type2^Mode2, ...]* or *[Type1^Mode1|Rest].*
This format of categories is deducible to the flat types of Buszkowski (1982). The only element added is treating all the arguments at one side of the head as one object: a list or a stack. For this defining property of our categories, we call the resulting grammar *Categorial List Grammar* or CLG.

Every phrase in the language that has to be defined is assigned to one or more categories by the lexicon or by the grammar. Here is an example from Dutch. The article *de* ('the') is lexically assigned to the category `np\u~[]/u~[n^i]`. Its left arity is zero; its right arity is one. The category expresses that its phrases are of type `np` if combined with phrases of type `n`. Combining with a phrase equals merging with a category to which that phrase is assigned. It will operate according to merge mode *i* on a category headed by *n.* If mode *i* allows operating on *e.g.* `n\u~[]/u~[]` and the noun *duivelsuitdrijver* ('exorcist') is lexically assigned to that category, the grammar may assign the phrase *de duivelsuitdrijver* to the category `np\u~[]/a~[]`. The modalized argument *n* in the original category for *de* has been cancelled, and RightFlag adapted. The category thus specifies the nature of categories to be operated on by listing heads in the argument lists. The associated modalities convey the conditions under which this merge may unfold in the modalities. Furthermore, the category stores concise information as to its merge history in the flags at the argument lists. Merging categories is therefore a complex, asymmetric operation.

Merging categories is the only combinatory operation in the syntax. It was originally introduced in Cremers (1993a). The operation is a generalization of Generalized Composition for Combinatory Categorial Grammar, as it was framed by Joshi *et al.* (1991) to capture the grammar engines constructed by Steedman (1996, *e.g.*).

In Combinatory Categorial Grammar, Generalized Composition is the main engine of analysis; here are its two instances as presented by Joshi *et. al.* (1991).

(28) $x/y \quad (...(y|z_1)|z_2 ... |z_n) \Rightarrow (...(x|z_1)|z_2 ... |z_n)$
(29) $(...(y|z_1)|z_2 ... |z_n) \quad x\backslash y \Rightarrow (...(x|z_1)|z_2 ... |z_n)$

x/y and *x\y* are considered to be the primary category of the compositions (28) and (29), respectively; the other one at the left of ⇒, headed by *y*, is called the secondary category. Every occurrence of $|z_i$ is a unit, representing either $\backslash z_i$ or $/z_i$. Directionality of arguments is not affected by composition (Steedman 1990). Generalized Composition cancels the argument */y* in the primary category against the secondary category's head *y* and yields the remaining structure *x* of the primary category with all the arguments of the secondary category stacked on top of it.

Categorial List Grammar restricts Generalized Composition by requiring the cancelled type *y* to be primitive or atomic, i.e. without internal structure. This restriction is referred to as *linearity*. In the terminology of König (1990), the resulting grammars would be characterized as first-order grammars, since there is only one relevant level of type embedding. A type is either a head or an argument to that head. Hepple (1996) describes a linearization procedure as a compilation of higher-order categorial grammars for parsing. Moreover, the syntax presented here will take into consideration the full internal structure of type *x*, which is not specified in Generalized Composition. As a consequence, Categorial List Grammar extends generalized composition form (28) to two different operations:

(30) $(...(p|w_1)|w_2...|w_m)/y \quad (...(y|z_1)|z_2 ... |z_n) \Rightarrow (...(...(p|z_1)|z_2 ... |z_n)|w_1|w_2... |w_m)$
(31) $(...(p|w_1)|w_2...|w_m)/y \quad (...(y|z_1)|z_2 ... |z_n) \Rightarrow (...(...(p|w_1)|w_2... |w_m)|z_1|z_2 ... |z_n)$

Of these, the second is a more explicit version of (28). The first form of composition is not covered by Generalized Composition, because there, the primary category's head and arguments cannot be separated. Furthermore, Categorial List Grammar will split up both the secondary and the primary category with respect to the directionality of arguments. CLG then collects the set of arguments in each direction. Consequently, the extension will be split up again into four different modes of composition. They are only distinguishable as to the relative orderings of the directed arguments in the consequent when compared to their sources (graphical marking is just for convenience):

(32) $\mathbf{p}\backslash pl_1...\backslash pl_n\ /\mathrm{pr}_1.../\mathrm{pr}_m/\mathbf{s}\ \ \mathbf{s}\backslash \mathrm{sl}_1...\backslash \mathrm{sl}_k\ /sr_1.../sr_l \Rightarrow$
$\mathbf{p}\backslash \mathrm{sl}_1...\backslash \mathrm{sl}_k\ \backslash pl_1...\backslash pl_n/sr_1.../sr_l/\mathrm{pr}_2.../\mathrm{pr}_m$

(33) $\mathbf{p}\backslash \mathrm{sl}_1...\backslash \mathrm{sl}_k\backslash pl_1...\backslash pl_n/\mathrm{pr}_1.../\mathrm{pr}_m/sr_1.../sr_l$

(34) $\mathbf{p}\backslash \mathrm{pl}_1...\backslash \mathrm{pl}_n\backslash \mathrm{sl}_1...\backslash \mathrm{sl}_k/\mathrm{sr}_1.../\mathrm{sr}_l/\mathrm{pr}_1.../\mathrm{pr}_m$

(35) $\mathbf{p}\backslash pl_1...\backslash pl_n\backslash \mathrm{sl}_1...\backslash \mathrm{sl}_k/\mathrm{pr}_1.../\mathrm{pr}_m/sr_1.../sr_l$

By simple computation, CLG extends every instance of Generalized Composition to eight patterns. The properties of these patterns are discussed in the remainder of this section. In the formatting of CLG below the directed arguments of a category are bundled into two single objects, ordered lists or stacks. In order to keep the composition or merge of categories functional, a finite number of distinct modes of composition will be distinguished.

1.4.3 Merge

Merge basically sends pairs of categories to categories, and therefore resides in the functional space $C^{C\times C}$. It consists of four sub-operations:

(36) cancelling a type in a pair <Type, Mode> at the top of an argument list of one category against the head of the other
(37) appending two pairs of argument lists
(38) reflagging the resulting argument list
(39) constructing a new category from the components

The whole operation is subjected to one out of a finite set of modalities, the one that is associated with the type to be cancelled.

Here is an example of a merge of two categories C1 = `s\u~[pp^isl]/u~[vp^rais]` and C2 = `vp\u~[np^isl]/a~[vp^cons]`. The only possible cancellation involves the type `vp` in the righthand argument list of C1 and the `vp` head of C2, under the modality `^rais`. The cancellation can be effected when C2 is the right member of the pair of input categories and the internal structure of C1 and C2 complies with the constraints defined by `^rais`. Suppose that both conditions are met. If we indicate the asymmetric merge with the infix ⊗, the merge could look like

(40) C1 ⊗ C2 → C3

```
s\u~[pp^isl]/u~[vp^rais] ⊗ vp\u~[np^isl]/u~[vp^cons] →
                                    s\u~[np^isl, pp^isl]/a~[vp^cons]
```

In this merge, the left-argument lists are appended in such a way that the argument list delivered by C2 is prefixed to the left-argument list stemming

from C1. Hereby, the order of future cancellation is fixed. At the right-hand side, only the argument list of C2 survives, as the right-argument list of C1 is emptied of its only argument after cancellation. The flag `a~` at the new list indicates that it was rightward cancelling that brought about this merge: the right-hand argument list is the affected one.

Reversing the point of view, one may now define what exactly are the constraints imposed by the `^rais` mode. In fact, two different patterns have to be stated, depending on the direction of the cancelling. The particular merge mode imposed by `^rais` is indicated by the modalized merge operator $\otimes_{\text{rais}}$. This convention is also used by Cremers (1993a) to introduce modes of application, by Moortgat (1998) to describe Multi-Modal Categorial Logics, and by Baldridge and Kruijff (2003) to represent Multi-Modal Combinatory Categorial Grammar. In terms of the last, the grammar of Cremers (1993a) might be described as multi-modal combinatoric.

Since merge is essentially asymmetric – this item is addressed in section 1.5.3 below – the process discriminates systematically between the two input categories. The category that has an argument at the top of one of its lists cancelled is dubbed the *primary* category. Its head is *Prim*, and all its others components are preceded by a *p*. The other components are identified by *f* and *a* for flags and argument lists, respectively, and by *r* and *l*, for left and right sides, respectively. The other category is doomed to be the *secondary* one, with head *Sec* and *s*'s instead of *p*'s. Finally, asymmetric append is marked by ⊕. Here then are two possible instances of the merge mode `^rais`. Capital onsets indicate variables.

(41) $(\otimes_{\text{rais}}/)$
$\text{Prim}\backslash\text{Plf}{\sim}\text{Pla}/\text{Prf}{\sim}[\text{Sec}\hat{}\text{rais}|\text{Pra}] \;\otimes_{\text{rais}}\; \text{Sec}\backslash\text{Slf}{\sim}\text{Sla}/\text{Srf}{\sim}\text{Sra} \;\rightarrow$
$\text{Prim}\backslash\text{Slf}{\sim}\text{Sla}\oplus\text{Pla}/\text{a}{\sim}\text{Pra}\oplus\text{Sra}$

(42) $(\otimes_{\text{rais}}\backslash)$
$\text{Sec}\backslash\text{u}{\sim}\text{Sla}/\text{Srf}{\sim}\text{Sra} \;\otimes_{\text{rais}}\; \text{Prim}\backslash\text{Plf}{\sim}[\text{Sec}\hat{}\text{rais}|\text{Pla}]/\text{Prf}{\sim}[\,] \;\rightarrow$
$\text{Prim}\backslash\text{a}{\sim}\text{Pla}\oplus\text{Sla}/\text{Srf}{\sim}\text{Sra}$

Prim: head of *prim*ary category; *Plf:* *p*rimary category's *l*eft argument list *f*lag;
Sec: head of *sec*ondary category; *Sra:* *s*econdary's category's *r*ight *a*rgument list

The first of these two merge modes does not impose any constraint on the input categories apart from the presence of cancelable types. As for the output, it specifies prefixing of the secondary argument list at the left-hand side and the reverse at the right-hand side. The flag `a~` at the right-hand side marks

affectedness of this argument list. The flag at the other side is provided by the secondary category.
Unlike ($\otimes_{rais}$ /), ($\otimes_{rais}$ \) does impose restrictions on the structure of the input categories. It requires the left argument list of the secondary category to be unaffected up to then, and specifies emptiness for the right argument list of the primary category. In its output specifications it essentially behaves like its rightward dual, *modulo* directionality.

Here is a sample instantiation of ($\otimes_{rais}$ /). The Dutch verb *willen* ('to want') is a typical verb raiser. It selects, among other things, infinitival complements and adjoins to the left of their verbal head. Consider one of its finite forms, for example the singular past tense form for embedded, *i.e.* verb-final, sentences: *wou*. This form introduces a subject argument it agrees with to its left, and an infinitival complement to its right. It would typically be assigned to the category `s_emb\u~[np]/u~[vp^rais]`. Both lists are marked zero, as the category originates from lexical assignment. Take, furthermore, the verb *gaan* ('to go'). One of its combinatorial options is to create a vp by selecting a directional pp to its left. Thus, it will be assigned lexically to the category `vp\u~[pp]/u~[]`; the directionality of the pp is expressed in the template of the verb, as a feature. The merge mode ($\otimes_{rais}$ /) specified above yields the following composition of these two categories, deriving a new category for the string *wou gaan* ('wanted to go'):

(43) `s_emb\u~[np]/u~[vp^rais]` $\otimes_{rais}$ `vp\u~[pp]/u~[]` → `s_emb\u~[pp, np]/a~[]`

In compliance with the format for $\otimes_{rais}$, the left argument list of the secondary category, here consisting only of the argument `pp`, is appended as a prefix to the left argument list of the primary category. By implication, the resulting category will have to cancel `ppdir` before `np`. This reflects the state of affairs where the subject of *wou gaan* is more peripheral to that phrase than to its directional argument.

1.4.4 Modalities

It is by no means necessary that every modality be specified both for rightward and for leftward cancellation. Given the nature of the example (43) for ($\otimes_{rais}$ /), it is even unlikely that the grammar of Dutch would give rise to ($\otimes_{rais}$ \). Merge is inevitably asymmetric. One head is cancelled, the other head persists. The argument lists in one direction are unaffected, those in the other direction lose

an argument. For each direction, the unfolding or execution of one argument list will be suspended, while the other is readily available for cancellation. Since linearity matters in natural language, directional duals for merging modes will be the exception, rather than the rule.

Plural specifications in one direction are possible. They should address disjointed sub-domains of Cats × Cats, though, as merge is functional. Nevertheless, it can be proven that the set of expressible merges, and thereby the set of merge modes, is finite. To see why, consider first this characterization of an expressible merge. A merge aims at the cancellation of exactly one type at the top of one argument list and does not refer to other types in that or other argument lists. It specifies the heads of two categories and exactly two appends of argument lists. It specifies two flags at most to these lists. Therefore, the gamma of specifications by a merge mode is rather limited. Moreover, all the components of a merge come from limited sets or classes of suboperations. First, the set Types is taken to be finite, in accordance with the standard condition in formal grammar that both the terminal and the nonterminal alphabet be so. As for input conditions, merge modes may specify for each of four argument lists whether they are empty or nonempty, and may specify flags for these lists. The number of different modes is very limited, by definition. As for output conditions, a merge mode can only specify argument lists that were part of the input. There are only two combinatory possibilities for constructing new argument lists out of these, resulting from the asymmetry of append. No other operations on argument lists are definable. The number of flags to specify for the new lists is as restricted as it was at input. All components of merge appear to stem from restricted sets. Consequently, the number of different combinations is bound to be finite, and so is the set Mods of merge modes.

1.4.5 Argument list

The only operations defined on argument lists are cancellation of an argument according to the modality that comes with it and appending. The latter creates a new list but leaves the lists that are feeding it intact. One could think of alternatives to this rigid form of list construction, like mixing or popping:

(a) mixing: [a,b] ∇ [d,e] = [a,d,b,e]
(b) popping: [a,b] ↵ [d,e] = [d,a,b] ↵ [e] = [e,d,a,b]
(c) appending: [a,b] + [d,e] = [a,b,d,e]

Mixing is sometimes referred to as list merge; for obvious reasons, the term mixing is preferred here. Lists are linearized, and thus they are ordered structures. Specifying append as the operation to be performed while merging categories implies conservation of these structures with respect to adjacency and precedence. Mixing respects linearity but not adjacency. Popping respects adjacency, but partially reverses linearity. As a matter of fact, appending is the only operation that respects both the local characteristic of adjacency and the global characteristic of precedence.

Comparing mixing, popping and appending, some analogies are straightforward. Popping reflects the way stacks combine. A stack is loaded and emptied in a regular fashion, element by element, by simple repetition of the same manoeuvre. Mixing is a context-sensitive operation. It requires keeping track of former moves and accounting for the internal structure of the lists involved. The analogy here is that of combining agendas, *i.e.* tasks that have to be performed in a certain order, feeding and bleeding each other. Appending, then, is in-between. It requires recursion, *i.e.* going back to a bottom case while suspending the shifts of individual elements. No bookkeeping of the internal structure of the lists involved is necessary, and it is context-free (cf. section 1.8). If append turns out to be the canonical mode of merging argument sequences, this operation alone would already fall in with the need for recursion in natural languages, as has been argued by Hollebrands and Roeper (2007). Although we adhere to append as the proper mode of merging, we will use both of the terms *stack* and *list*, as they both reflect the essential property of complete ordering. Moreover, taking argument lists to be the stores that govern cancellation of types, we will address argument list as the grammar's *agendas*.

The conservation of adjacency and precedence in the local environment introduced by a category is the reason for considering append as the structural operation underlying merge. Categories to which lexical items are assigned reflect partial knowledge of the combinatoric potency of that item. That knowledge concerns selection and subcategorization, but also linearization of the selected items. Moreover, it can be seen as constituting the domain governed by the phrase assigned to this category. This configuration is not accidental but defines the whereabouts of the phrase's interpretation. The merging of categories is the engine of phrase combinatorics in this grammar. As such, it accounts for the discontinuities natural language abounds in. But merging would be self-destructive if it were to come with the mutation or covering of the configurations that make the phrases which it combines meaningful and

interpretable. Grammar is supposed to make sense of the intrinsicalities of natural language. Appending the identifying domains of phrases, then, seems to be the more harmless option in merging categories.

1.4.6 Deduction schemes

The grammar outlined here can be framed into a sequential, deductional format. Here is a model. Types are indicated by lower case letters, lists of arguments by indexed L_i and R_j. Categories are marked by Roman capitals A, B, C. Potentially empty sequences of categories are marked by distinct Greek capitals.

(44) Axioms

unary axiom: $C \rightarrow C$

binary axiom: $A \otimes_i B \rightarrow C$ *for all* $\otimes_i$ *defined*

rightward: $a\backslash L_1/[b\hat{}i|R_1] \otimes_i b\backslash L_2/R_2 \rightarrow a\backslash L_1 \oplus_i L_2/R_1 \oplus_i R_2$

leftward: $b\backslash L_2/R_2 \otimes_j a\backslash [b\hat{}j|L_1]/R_1 \rightarrow a\backslash L_1 \oplus_i L_2/R_1 \oplus_i R_2$

(45) Rules

left rule $\dfrac{\Delta \rightarrow B \qquad A \otimes_i B \rightarrow C \qquad \Gamma', C, \Gamma'' \rightarrow T}{\Gamma', A, \Delta, \Gamma'' \rightarrow T}$

right rule $\dfrac{\Delta \rightarrow B \qquad B \otimes_i A \rightarrow C \qquad \Gamma', C, \Gamma'' \rightarrow T}{\Gamma', \Delta, A, \Gamma'' \rightarrow T}$

Note that the rightmost premise is shorter than the conclusion, in that its antecedent contains at least one category less than the conclusion's antecedent. Note furthermore that in all binary axioms the arity of the consequent is exactly one less than the sum of the arities in the antecedent, which is necessitated by the definition of merge as involving cancellation. It follows that the arity of the right-hand premise is smaller than the arity of the conclusion. Clearly, then, the decidability of this calculus depends on the decidability of the term $\Delta \rightarrow B$. We prove its decidability by induction on the length of Δ. $|\Delta| \leq |\Gamma' \Gamma'|$, by definition. If $|\Delta|=1$, the term instantiates identity or it is false. If $|\Delta| = 2$, its components satisfy, following the rule format, one of finitely many binary axioms, or it is false. If $|\Delta| > 2$, the rules apply to create a left-hand premise $\Delta' \rightarrow B'$ such that $|\Delta'| < |\Delta|$. Given the remarks on the decidability of the middle and right-hand premises in the rules, the derivability of each proposition $\Gamma \rightarrow t$ for some designated type t can thus be deduced in a finite number of steps.

For the type of grammar presented in (44) and (45), an important invariant can be established. The invariant is a simplified version of the count-invariancy that Van Benthem (1991) uncovered for the Lambek-calculus and related systems. This invariant states that there is a certain way of counting primitive types, such that theorems of that calculus of the form $\phi \rightarrow \psi$ invariantly have the same count at both sides of the arrow. Since categories in the Lambek-calculus are more complex than in our grammar, the metric is simpler too.

(46) For each type in a category C,
count an occurrence of t as +1 if it is the head, and as -1 if it is an argument,
determine t-count(C) as the sum of the occurrences of t in C,
for each sequence of categories, determine the t-count for that sequence as the sum of the t-count for the categories in the sequence.

Given this metric, the simple fact that cancellation is a zero-sum operation, and the observation that no types other than the one cancelled are affected in an axiom, the following proposition is evident:

(47) *Count Invariance*
For each axiom $\phi \rightarrow \psi$ in (44) and for every type t, t-count(φ) = t-count(ψ).

As an immediate consequence, the rules in (45) are conservative in the sense that all the operations above and below the deduction line respect Count Invariance. In particular, the conclusion and the major premise share t-counts. Thus, the grammar enjoys *deductive monotony:* no types appear or disappear apart from zero-sum cancellation.

1.4.7 The algebra of strings

The categories, and the operations defined on these, can be interpreted on a string model <L*, +>. L is the set of atomic phrases of a language and L* is Kleene closure: the set of non-empty strings over that language such that $L \subseteq L^*$ and for all $a, b \in L^*$, $a+b \in L^*$. The string operation + is taken to be associative and noncommutative. It gives rise neither to idempotency nor to persistency. As such, it is, as Morrill (1994) states, an operation on pieces of matter rather than an operation on pieces of knowledge. Another image that comes to mind when looking for interpretations is the flow of time. Language essentially extends in time. Phrases can be seen as decorated intervals. Although time may be dealt with under all kinds of logics, there is a standard perception of the flow of time as a noncommutative, *i.e.* irreversible, and nonpersistent, *i.e.* segmentable, process. The operation + is meant to reflect this analogy.

Consider furthermore the power set 2^{L^*}. A mapping $\mathfrak{R}$: Cats ∪ Types → 2^{L^*} has to be established, assigning to a category or type a set of strings *of* that category or type. This is the standard interpretation of a category as a class of strings with equivalent combinatorial behaviour. In presenting categories for this purpose we will, for the moment, abstract from flags and modalities. The operator ● indicates asymmetric associative concatenation of categories; ○ interprets this relation on categories.

(48) $\mathfrak{R}(A) = \mathfrak{R}\ (A\backslash[\ ]/[\ ])$ for all $A \in T$
$\mathfrak{R}([A_1,\ldots,A_i,\ldots,A_n]) = \mathfrak{R}(\ A_1\bullet\ldots\bullet A_i\bullet\ldots\bullet A_n\ ) =$
$\mathfrak{R}(A_1)\circ\ldots\circ\mathfrak{R}(A_i)\circ\ldots\circ\mathfrak{R}(A_n) = \{\ a_1+\ldots+a_i+\ldots+a_n \mid a_j \in \mathfrak{R}(A_j),\ 1 \leq j \leq n\ \}$
$\mathfrak{R}(A\otimes_m B) = \{\ a+b \mid a \in \mathfrak{R}(A),\ b \in \mathfrak{R}(B)\}$, for all *m*
$\mathfrak{R}(A\backslash x\sim L/y\sim R) = \{\ a \mid \forall l \in \mathfrak{R}(\text{reverse}L),\ \forall r \in \mathfrak{R}(B)\ l+a+r \in \mathfrak{R}(A)\ \}$ (the reverse of L is needed here as the order of the list of arguments to the left is the reverse of the order of strings associated with that list; flags of argument lists do not interfere with the denotation of categories.)

The syntax established here gives rise to a form of category merging that has been dubbed *mixed* or *disharmonic* composition. Its characteristic feature is that arguments from the secondary category not belonging to the direction that is affected by the cancellation are taken over by the new category. For example, if *SLA* below is non-empty, the indicated merge will involve this disharmonic composition. (The + operator amounts to standard *append*.)

(49) PRIM\PLF~PLA/PRF~[SEC^j|PRA] $\otimes_j$ SEC\SLF~SLA/SRF~SRA →
PRIM\SLF~SLA+PLA/a~SRA+PRA

In a slightly more transparent but less general notation, disharmonic composition is exemplified in the following lines:

(50) x/y y\z ⇒ x\z (rightward cancelling, leftward composition)
y/z x\y ⇒ x/z (leftward cancelling, rightward composition)

The particular appending of *PRA* and *SRA* in (49), on the other hand, is called *homogeneous* or *harmonic*. Here, the arguments entered by the secondary category, are of the direction affected by the cancellation. They are placed on top of those entered at that side by the primary category. Here are the simplified representations for harmonious composition:

(51) x/y y/z ⇒ x/z (rightward cancelling, rightward composition)
y\z x\y ⇒ x\z (leftward cancelling, leftward composition)

The main combinatorial difference between harmony and disharmony in this respect is the following. If a string is constructed with harmonious composition, there may be an alternative derivation without transfer of arguments; if a string is constructed by disharmonious composition, there is no alternative to that derivation. Again, this is illustrated in the alternative notation, but only in one direction. In (52), there are two ways to merge three categories into one: one involving (harmonious) composition of the two leftmost categories, the other using sheer cancellation twice; for the disharmonious composition in (53), there is no alternative.

(52) x/y y/z z ⇒ x/z z ⇒ x
x/y y/z z ⇒ x/y y ⇒ x
(53) z x/y y\z ⇒ z x\z ⇒ x

Consequently, disharmonious composition allows for grouping of strings that are discontinuous to such a degree that it cannot be solved by the present categories. That is: the Lambek-calculus cannot map the categories in (53) to other categories that can be reduced to a single type without resorting to disharmonious composition. Since the Lambek-calculus has been proven to be weakly equivalent to context-free grammars (Pentus 1993), the use of disharmonious composition pushes a categorial grammar beyond context-freeness. As a matter of fact, the syntax of merging listed first-order categories under modalities induces several distinct patterns of discontinuity; these will be discussed in the sections on the syntax of Dutch. The syntax (of Dutch) is one of discontinuity, rather than a syntax of gluing constituents. The basic combinatorics that is envisaged for interpretable strings is chaining rather than appending. It is designed not so much for combining strings *w* and *z* into *wz* but for combining strings *wx* and *zy* into *wzxy* or *zwyx* in a controlled manner. Such forms of string merge are immanent to natural language: strings chain each other rather than just glue together. They bubble up between each other's edges in a variety of modes. Nevertheless, we can have resort to + as the designated operation on strings, since no operation of the grammar violates the integrity of a deduced string. Merging prompts strings for peripheral association only. It does not involve any kind of extraction from or insertion into otherwise constructed strings. The discontinuous effects are solely caused by reference to the internal structure of the category that is associated with a string, *i.e.* to which a string is assigned. Once a string is derived by merge, it is maintained in the rest of the derivation. In this respect, the syntax is conservative and monotonous.

By consequence, the general strategy for constructing *ywzx* as stemming from *wx* and *yz* is mandatory. Suppose $wx \in \mathfrak{R}(A)$ and $yz \in \mathfrak{R}(B)$ for some A and B,

but there is no C such that *wz*, *yw* or *zx* ∈ ℜ(C). That is what it means to say that *ywzx* stems from *wx* and *yz*. The only derivation in the grammar depicted above runs as follows. Construct *w* and put the construction of string *x* on the agenda. Construct *z* and put the construction of string *y* on the agenda. Merge *w* and *z* and execute the agenda.
Now consider how under this regime a string *wyzx* must be derived from the same formants and with the same labour division between *w* and *x*. The string *wz* cannot be constructed, not even if for some C *wz* ∈ ℜ(C), since no rule of grammar could split it up for *y* to be inserted. Therefore, the string *yz* has to be formed first. Next, *wyz* can be derived, while putting *x* on the agenda.

1.4.8 Disharmonic composition and categorial logic: grammar and reasoning

As indicated above, CLG comes with disharmonic composition. In the present setting, disharmony results from appending a non-empty argument list of the secondary category in the passive direction, *i.e.* the direction that is not affected by cancellation – *Sla* in merge scheme (49). It has often been argued that categorial grammars involving disharmonic composition are beyond string models. Disharmony is rejected in, *e.g.*, Carpenter (1997), Morrill (1994) and Jacobson (1991). The last states that disharmonic or mixed composition, though attractive for dealing with certain phenomena, cannot be function composition. If *f* sends *a* to *ba* and *g* sends *c* to *cd* and *cd* is in the domain of *f*, then *f*∘*g* sends *c* to *bcd*, as Jacobson correctly claims for functions *f* and *g*. If the merge of categories A and B does not have this functional effect, it cannot be functional. To put it bluntly: for every argument c, the composition applied to the argument – *f*∘*g(c)* – is bound to be equal to the subsequent application of the rightmost function to the argument and the application of the leftmost function to the result of the innermost application – *f(g(c))*. In disharmonic merge, this is typically not the case. If disharmony is called for, there is no alternative to it, as was argued in the preceding section.
This argument against the functional nature of disharmony takes the directionality of categories seriously: it defines the functionality of categories in terms of linearization of strings. On the other hand, functions are not necessarily directed objects. The functional interpretation of categorial deduction under the Curry-Howard correspondence (cf. Van Benthem 1986) abstracts from directionality. As a matter of fact, this is the main reason why the correspondence is not an isomorphism over substructural logic – intuitionistic logic with selective use of structural rules – and constitutes a particular fragment

of the lambda calculus. *Merge*, as defined above, can be seen as a complex of operations, some of which deal with linearization, while others take care of the logical aspects. The linearization operation – *append*, basically – does not contribute to functionality, although *append* itself is a homomorphism in $L^{L\times L}$, where L is the class of finite lists over a given domain. The grouping or bracketing operation that comes with *merge* is as functional as it can be.

Morrill (1994: 231-232) refutes a syntax which applies selective linearization schemes: '… then the theory of syntax is not logical, in the sense of being the reflection of an interpretation of category formulas, but … a deductive system receiving definitions in terms of non-logical axioms and rules.' The point here is not merely that one could choose, operationally, between redefining 'existing' operators and adding new ones, the latter being common practice in Multimodal Categorial Logic (Moortgat 1998). The crux is the nature of the relationship between logic and grammar. In Morrill's understanding, logic starts out with irrefutable axioms, and the grammatical combinatorics reflect this reasonable base. The basic logic here is intuistionistic conditionalization. It defines three operators (left division, right division and noncommutative multiplication) in a multiplicative fashion by residuation:

(54) $a \rightarrow c \,/ b$ *iff* $a \cdot b \rightarrow c$ *iff* $b \rightarrow c \backslash a$

One could also opt for nondirected commutative operators, according to

(55) $a \rightarrow c | b$ *iff* $a \cdot b \rightarrow c$ *iff* $b \cdot a \rightarrow c$ *iff* $b \rightarrow c | a$

Other operators can be defined, also in duals, in terms of residuation. The one-place operators ◊ and □ are famous by now (see Moortgat 1998, Morrill 1994):

(56) $\Diamond a \rightarrow b$ *iff* $a \rightarrow \Box b$

Their multiplicativity can be compared to the duals *n-root* and *n-power* defined on the positive reals: $\sqrt{a} = b$ *iff* $a = b^2$.
For deduction, division is interpreted as implication and multiplication as co-occurrence. The one-place operators add modal control, by closing and opening – basically, marking and unmarking – (sequences of) categories.

From a logical point of view, an important aspect of this system is that it lacks negation, or its algebraic counterpart, complementation. The string alge-

bra has no zero-element Zero such that concatenating any String with Zero always yields String. Suppose Zero existed. Its category should be of type x/x and x\x, for all categories x. Zero would be a homomorphism over all categories. Any string String would be combinatorially and materially equivalent to $Zero^n$+String+$Zero^m$, and identity of strings would not be decidable – note that the algebra of strings is resource-sensitive. Therefore, no combinatory object in grammar denotes the empty string.

The absence of complementation in the logical basis to the categorial analysis can be well defended under reference to the nature of language combinatorics. Language is a material system, and its grammar deals with real objects; it cannot be unwound. Its combinatorics are, in intuitionistic terms, *resource-sensitive*. It irreversibly consumes time and space, and it is hard to see what operation in grammar could cover algebraic complementation or logic negation.

In the same vein, it is worth asking what aspect of natural language makes conditionalization a suitable vehicle to guide combinatorial operations. In 'classical' flexible categorial grammar (Lambek 1958, Moortgat 1988 and subsequent work) hypothetical reasoning is the fuel for the type-changing operations that drive deduction. In the calculus, deductive hypothetical reasoning is reflected in so-called *introduction* rules. These rules increase categorial complexity and cover – among others – one of the greatest discoveries in the history of linguistics, the generalizability of quantification (Montague 1972, Barwise & Cooper 1981):

(57) e → <<et>t> *or*: whenever you run into a primitive entity, you had better take a structured set of sets of entities

Its proof in elementary categorial logic involves the assumption that *e* is brought to *t* in the presence of <*et*>, to wit: *e* <*et*> → *t.* The type transition (57) withdraws the assumption that <*et*> is available. It is the same hypothetical reasoning as in classic natural deduction, but with the intuitionist proviso that assumptions and withdrawals are one-to-one.

What makes hypothetical reasoning so attractive in grammar is the previously acknowledged insight that language is a material, resources-consuming system. To say that a certain element requires another or is conditioned by another is saying that somewhere in the linguistics space (*e.g.* a sentence) that other element must be available. *Somewhere*, however, is far too weak. It never means *anywhere*. Generally, the requirement is much more specific: the other element must be available in a certain well-defined subspace of the lin-

guistic space. This subspace is identified relatively, with respect to the position of the requiring element, and in linear terms: to the right or to the left of that element. Relative linear occurrence is the language instance of logical conditionalization. In natural language, linear occurrence amounts to conditionality. It is by no means accidental that direction is part of the logical basis for natural-language analysis. Directionality is the name of the language game. And it is from directional requirements that we derive hierarchy, not the other way around: such is the message of Kayne (1994). We are still far from understanding exactly how linearity induces subordination and semantic dependency, but linearization is indispensable in the grammar base.

This being the case, we have at least two data structures for encoding the linearization requirements of phrases, *i.e.* their spaced conditionalizations.
First, there are the types exploited in Lambek (1958) and subsequential related work, notably Moortgat (1988) and Van Benthem (1991). Combinatorially, they can be seen as onions. Their internal structure might be complicated, but only their outer skin determines the combinatorial potential in a given state. The related combinatorics is naturally context-free, since the internal structure of the consequent of the primary category in the directed *modus ponens* that drives the system is left out of consideration. The full proof of the context-freeness of the Lambek-calculus based on this concept of type was presented by Pentus (1993). The computations that come with it relate to a tuple <body, outer skin>, where outer skin is the specified argument *with* its direction (see Steedman 1990 for an invariant of this nature).
Secondly, there are the 'flat' categories as derived in Buszkowski (1982). Though the author may not necessarily see them this way, they introduce the full directed internal structure of the *premisse major* as an entity in the *modus ponens*. Combinatorics defined on these data structures are context-sensitive (though not necessarily in the strong sense) because the internal structure of the premise is specified, and possibly taken into consideration. The computations that go with it relate to triple <head, leftward structure, rightward structure>. These characteristics of the flat, skinned categorial data structures also apply to the very first proposals of Combinatory Categorial Grammar, in Ades & Steedman (1982).
These two types of categorial information structures induce different calculi. The differences follow from the data structures themselves. The flat types allow for a fine-grained resource management in terms of linearization and subspaces. The onion types induce calculi that exploit the bare conditionalization, in the form of a conditional hierarchy, adding additional operations for resource management, like linear structuring.

Although a grammar may exploit conditionalization, it has no use for full complementation. Yet, negation is the landmark of reasoning, as it offers an alternative to the referent of a proposition. Without negation, we would not have truth – or whatever other device we may use to express the contingency of a proposition. Negation constitutes the smallest device that one needs to generalize over situations or states-of-affairs and to tell them apart. In grammar, however, there is no place for algebraic complementation. By consequence, natural-language grammar cannot be identified with reasoning as such. It can only be depicted as an algebra forging some structure over a set of expressions. Linguistics, then, is about the components of that algebra.

1.5 THE CALCULI

The strategy for characterizing natural languages as an algebra is choosing some well-defined starting-point to see what additional power is needed to deal with the intrinsicalities of the language at hand. That starting point could be PLA, the categorial system based on commutative residuation (55) and explored by Van Benthem (1991); it amounts to the Lambek-calculus under abstraction of directionality. This system is too weak, though. It defines no (syntax of any) natural language at all, since it induces a permutation-closure over the set of accepted strings. No natural language is known to be that free, not even Warlpiri, Hungarian or Latin. Therefore, it seems reasonable to upgrade one level, and exploit the system defined by residuation (54), *i.e.* the Lambek-calculus. In doing so, we take directionality and non-commutativity as basic, just like we take the interpretation of division as conditionalization to be basic. On the other hand, directionality and non-commutativity can be added to PLA in terms of modalized duals. Here is an example, expressing the functionality of (54) in (55):

(58) $a \to c|_i b$ *iff* $a \cdot_{i,j} b \to c$ *iff* $b \to c|_j a$

Here we can see that directionality and non-commutativity can be added to PLA just as any other regime of structural management could. The expressivity of such a system is more than sufficient to cover natural languages, it seems. It leaves room for the embedding of complex analyses of natural languages, as shown in Vermaat (1999). In this approach, every computable structure can be accommodated. It is Turing-complete. Of course, it is highly interesting

and revealing to scrutinize which embeddings are possible for every particular phenomenon or analysis. In fact, the multimodal approach exemplified in (58) seems to come with the categorial alternative to the Chomsky hierarchy called for in Van Benthem (1991).
CLG is considerably less expressive than multimodal categorial grammar. It imposes heavy restrictions on the set of derivable sequents and, on top of that, on the class of accepted strings, for any finite lexical assignment.

Morrill (1994) considers grammar as applied logic, as residing on logic grounds. In this view, adding axioms or adjusting operators just to comfort language analysis is needlessly weakening the grammar-logic connection; it amounts to redefining mathematics in order to get hold of nature. In another perspective, however, logic is defining (some of) the instruments for grammatical construal. Grammar is not necessarily applied logic, but may involve the application of logical means for natural-language analysis. This perspective finds justification in the present state of ignorance about the embedding of language in the cognitive and intellectual resources of our species. As far as we can see, at this moment we do not have conclusive evidence for language as an independent faculty, or for language as an epiphenomenon of emerged cognitive capacities, or for anything in between. This should not stop us from pursuing some hypothesis or other. In any case, one should adhere to explicating in grammar all those processes that one observes in scrutinizing natural language. Logic is extremely useful for this explication. CLG is an effort to localize the interference of selection and linearization in natural languages. Take, for example, the wrapping that canonical auxiliaries in Dutch induce. Auxiliaries, except when advanced to second position, are head adjuncts: they select a verbal complement of some sort to their right, but will typically wrap this complement around themselves in such a way that the complement's head is its right neighbour:

(59)	auxiliary	*wordt* ('is', passive aux)	x/vppas
	vppas	*in de verf gezet* ('painted')	(verb: *gezet*)
	string	*in de verf wordt gezet*	x

Alternatively, they select a complement to their left. But in that case, everything occurring to the right of the complement's head has to occur, after merge, to the right of the passive auxiliary:

(60)	auxiliary	*wordt*	x\vppas
	vppas	*met geweld gedwongen te vertrekken* ('violently forced to leave')	(verb: *gedwongen*)
	string	*met geweld gedwongen wordt te vertrekken*	x

Neither for the rightward nor for the leftward selection are string-ordering alternatives available. Any other ordering essentially affects interpretation or endangers interpretability. The following ordering variations on leftward selection show this.

(61) * met geweld gedwongen te wordt vertrekken
met geweld gedwongen te vertrekken wordt
*can only mean something like "*becomes violently forced to leave"

But then the following statement appears to be true: if the direction of the complement selection is part of the auxiliary's lexical definition, the way of wrapping that complement is part of that definition too. That is what CLG specifies. It takes discontinuity seriously, in defining natural-language grammar as computation of discontinuity. Likewise, non-adjacency of strings that are to be interpreted in relation to each other is a fundamental fact of language. Language is linear. It is a time-consuming process, and the order of events in this process is crucial. Disharmonious composition is one of the instruments for dealing with this ordering. It is not favoured or disfavoured compared to other merge formats. Logic, then, is just exploited to keep the analyses tractable.

1.5.1 Merge and Move

The presentation of the syntax hitherto has left no room for a fundamental distinction between the *move* and *merge* operations, as introduced in Chomsky (1995). Stabler (1992) essentially reduces this distinction to a matter of arity. Move is operating on one category to produce another. Merge operates on two categories to produce another. In both operations, checking features is the engine of the process. Cormack and Smith (1999) hypothesize that interpretation and merge of a constituent take place at the same level of derivation. There is no need for move in this case. CLG incorporates the view that merging implies moving, as lexical material is anchored in positions where it may or must split what otherwise would have been decent chains. The famous case at stake here is dative insertion. It is generally acknowledged that a verb and its direct object are tightly related. In many languages, how-

ever, a second object – the indirect one – may or must interfere between the verb and the direct object. This object too is licensed by the verb, but its relation to it seems looser, both syntactically and semantically. There is no particular relation between the direct and the indirect object. Nor is it the case that the complex of verb and neighbouring indirect object represents a more complete syntactic or semantic frame for the direct object. We do not know of any cases where verb and indirect object bring in interpretations that are not available to the verb without an indirect object. Since the indirect object is still part of the verbal complex, the mere merge of verb and indirect object induces relative movement of the direct object some distance away from its very licenser, the verb.

Likewise, one might look at the relation between prenominal adjectives and determiners. It is quite clear that there are several very severe restrictions operating on a determiner and its noun, ranging from the morphological shape to the quantificational nature of the resulting constituent. Adjectives have no, or hardly any, relation to the determiner, and some, but not very lexical (and thus not very computable) semantic cross-links with the noun. Still, adjectives occur at that side of the noun where determiners are bound to reside. The complex formation of adjective and noun, then, puts the determiner at some distance from its closest comrade.

In many respects, therefore, merge inevitably induces discontinuity, and discontinuity amounts to movement (cf. Cremers 2004).

1.5.2 Soundness and completeness

A sequent of the form $\Gamma \rightarrow B$ is valid if the set of phrases associated with Γ is included in the set of phrases assigned to B, *i.e.* if $\mathfrak{R}(\Gamma) \subseteq \mathfrak{R}(B)$. Here is proof that the sequent calculus presented above is sound, and in a certain sense complete with respect to the string model.

1.5.2.1 Soundness

As for soundness, *i.e.* the proposition: all that can be derived is valid, consider the following. The unary axiom is trivially valid. The binary axioms are valid by definition. We can therefore concentrate on the left one of the only two productive derivational steps:

(62) $$\frac{\Delta \rightarrow B \qquad A \otimes_i B \rightarrow C \qquad \Gamma', C, \Gamma'' \rightarrow T}{\Gamma, A, \Delta, \Gamma'' \rightarrow T}$$

Suppose $d \in \mathfrak{R}(\Delta)$ and $a \in \mathfrak{R}(A)$. By induction, $d \in \mathfrak{R}(B)$. By definition, $a+d \in \mathfrak{R}(C)$ and $\mathfrak{R}(A \bullet \Delta) \subseteq \mathfrak{R}(C)$. So for $g' \in \mathfrak{R}(\Gamma')$ and $g'' \in \mathfrak{R}(\Gamma'')$, $g'+a+d+g'' \in \mathfrak{R}(T)$ and thus $\mathfrak{R}(\Gamma \bullet A \bullet \Delta \bullet \Gamma'') \subseteq \mathfrak{R}(\Gamma' \bullet C \bullet \Gamma'') \subseteq \mathfrak{R}(T)$.

1.5.2.2 Completeness

As for completeness, *i.e.* the proposition: all that is valid can be derived, we have to prove that $\mathfrak{R}(\Delta) \subseteq \mathfrak{R}(T)$ implies derivability of the sequent $\Delta \rightarrow T$. In its full strength, this probably cannot be proven for the present system, as it lacks hypothetical reasoning and, thus, structural completeness – the property that the order of deduction steps is irrelevant for well-formedness and derivability. What can be proven, however, is a weaker statement that amounts to the following proposition.

(63) There is always an assignment of phrases to categories, such that if $d \in \mathfrak{R}(\Delta)$, $d \in \mathfrak{R}(T)$, then there is a Γ such that $d \in \mathfrak{R}(\Gamma)$, $|\Delta| = |\Gamma|$ and $\Gamma \rightarrow T$.

This means that the calculus is complete *salve* assignments of phrases to categories. The proof yields a construction of Γ as a string of categories $\Delta[X_i \leftarrow Y_i]$ which is like Δ, except for a finite number of substitutions of a category X_i by Y_i. The number of substitutions has an upper limit $|\Delta|$ and consequently, $|\Delta[X_i \leftarrow Y_i]| = |\Delta|$.

1.5.2.3 Help lemmas for completeness salve *assignment*

To prove completeness *salve* assignment, we have resort to the following lemma; here *L-LL* denotes the remnant of L after taking out that which L and LL have in common.

(64) *Binary derivability lemma*

for all Γ, $|\Gamma| \geq 2$, $\Gamma \rightarrow a\backslash L/R$ *iff*

$\Gamma = \Delta' \Delta''$ *and either*

$\Delta' \rightarrow a\backslash L_1/[b\hat{}i|R_1]$, $\Delta'' \rightarrow b\backslash L\text{-}L_1/R\text{-}R_1$ and
$a\backslash L_1/[b\hat{}i|R_1] \otimes_i b\backslash L\text{-}L_1/R\text{-}R_1 \rightarrow a\backslash L/R$

or

$\Delta' \rightarrow b\backslash L\text{-}L_1/R\text{-}R_1$, $\Delta'' \rightarrow a\backslash [b\hat{}i|L_1]/R_1$ and
$b\backslash L\text{-}L_1/R\text{-}R_1 \otimes_i a\backslash [b\hat{}i|L_1]/R_1 \rightarrow a\backslash L/R$

This lemma is to play the same role in the present completeness proof as the *canonical lemma* of Buszkowski (1982) plays in the completeness proof of the product-free Lambek-calculus.

From right to left, the lemma is evident, as it reflects the deductive aspect of the calculus. From left to right, the lemma is not trivial. It induces binary branching of the deductive process. The crucial deduction step in (62) substitutes a category *C* with a string *[A,Δ]*. If $\Delta \rightarrow B$ is derivable, it is an axiom, or it is derived by the same rule. In that case, Δ can be represented as Δ'', B' or B', Δ' for some category B' with the same head as *B*. This can be repeated until we end up with single categories only. Thus the structure 'below' *C* can be seen as a binary branching construal with a category on one branch and a string of categories on the other. 'Above' *C*, we may safely assume that C itself will be a right-hand or left-hand partner of some peripheral substring of either Γ' or Γ''. This duet, then, is substituted, in the deduction, by come category *C'*. Consequently, *C* itself will be the product of a binary process. Since every reduction decreases the number of categories and the accumulated complexity of the sequence, the structure converges, and must be binary branching. But then, at the top of that structure there are two categories covering the whole string of categories that is being deduced. The format of the categories involved in the lemma follows from the definition of merge. Note that by the monotony property of the grammar the arity of the consequent puts an upper limit on the arity of each of the categories in the antecedent.

We furthermore need the notion of *compatibility* between categories. If $A \otimes_i B \rightarrow C$, both A and B are compatible with C. If *A* is compatible with *C*, then, by binary derivability, for some *B*, $A \otimes_i B \rightarrow C$. One can easily compute that compatibility of *A* to *C* amounts to *C*'s argument lists being embeddable in *A*'s. It is noteworthy that the number of categories X compatible with a given Y is finite: Y's argument lists are finite and the set of types is finite. In fact, the number is linear in the arity of Y. Compatibility in this sense is a derivative of count invariance, established by Van Benthem (1991) for the Lambek-calculus with permutation. Count invariance expresses the property of a rule of categorial grammar that the types at the left-hand side add up to the types at the right-hand side when types can occur either positively or negatively in categories.

Finally, it must be shown that for every category that could be the product of reduction by some merge, such a merge can be found. A category is *reduced* if exactly one of its argument lists is flagged $_a\sim$. Recall that this flag indicates that one of the argument lists from which it was appended contained an argument that was cancelled by merge. If none of the list flags of a category is $_a\sim$, the category might still be a reduction; this would, however, depend on the availability of particular instances of merge. Furthermore, we assume that there is no interesting difference in denotation between two categories which

differ only in the flagging of their arguments. Flagging is control, not semantics. This was expressed in the last statement in definition (48).
Under this *proviso*, the following has to be shown:

(65) *Reducibility*
for every reduced *T,* there are categories *A* and *B* and a merge mode *i* such that $A \otimes_i B \rightarrow T$

Since categories are constructable *ad libitum*, the presence of suitable merge modes is the bottleneck here. We may safely assume that every grammar defines at least one merge mode. Let $\otimes_i$ be this merge mode and suppose, without loss of generality, that it induces a right-hand side cancellation; let *T*'s reducibility agree with this directional feature. *T* will, then, have the format $a\backslash fl_{lt}{\sim}L/a{\sim}R$. Thus it must be proven that there are argument lists L_1, L_2, R_1 and R_2 such that:

(66) $a\backslash fl_{la}{\sim}L1/fl_{ra}{\sim}[b\hat{}i|R_1] \otimes_i b\backslash fl_{lb}{\sim}L2/fl_{rb}{\sim}R2 \rightarrow a\backslash fl_{lt}{\sim}L/a{\sim}R$

The merge mode *i* specifies some appends $L_1 \oplus_{il} L_2 = L$ and $R_1 \oplus_{ir} R_2 = R$ with appropriate flagging as output-conditions, and some restrictions $inp(L_j)$ and *inp(Rj)* as input-conditions on the left and right argument lists of the two antecedent categories, including their flags. Now suppose that $fl_{la}{\sim}L1$ agrees with $inp(L_j)$ and that $fl_{ra}{\sim}[b\hat{}i/R_1]$ agrees with $inp(R_j)$; of course, we can always construct the first category so that it accords with the input requirements for *i*. So, the only thing left to show is that, given *T*, the operation $\otimes_i$ and an appropriate compatible left-hand side can be constructed. Recall that the appendings that come with the merges are defined conservatively. They are restricted to asymmetric linear gluing of lists. Consequently, L_1 is constructed in such a way that it is a sublist of *L*; as a matter of fact, it is a (possibly empty) prefix or a (possibly empty) suffix of *L*. Which of these options holds is defined by $\oplus_{il}$ as part of the definition of $\otimes_i$. But then, L_2 is the result of the subtraction *L-L1*, uniquely defined given *L* and L_1. The same reasoning can be carried over to R_1, R_2 and *R*. So, L_2 and R_2 are uniquely defined by $\otimes_i$ and the other argument lists. As for the flags of the argument lists of the antecedent categories, since they do not in any respect reflect the present state of these lists, they can be chosen freely in accordance with $\otimes_i$. So, *B* can be constructed. This proves reducibility (65).

1.5.2.4 Completeness salve *assignments*

Given binary derivability, completeness *salve* assignments amounts to the claim that $\mathfrak{R}(\Delta) \subseteq \mathfrak{R}(T)$ implies derivability of the sequent $\Delta', \Delta'' \rightarrow T$ for some bipartition Δ', Δ'' of Δ and reduced T. Recall that $\mathfrak{R}(T) = \mathfrak{R}(T')$ if $T = T'$ *modulo* flagging of argument lists. Now suppose $d \in \mathfrak{R}(\Delta)$, $d = d' + d''$, $d' \in \mathfrak{R}(\Delta')$, and $d'' \in \mathfrak{R}(\Delta'')$. Then, if $\Delta' \rightarrow A$ and $\Delta'' \rightarrow B$ for some A en B such that $A \otimes_i B \rightarrow T$, $\Delta', \Delta'' \rightarrow T$ and, by soundness, $d'+d'' \in \mathfrak{R}(T)$. So it must be proven that such A and B exist. This proof is by induction on the length of Δ' and Δ''.

(67)

(a) If $|\Delta'| = 1$, we have identity and for some A, $\Delta' \rightarrow A$.

(b) Suppose A is compatible with T. By reducibility, there is a B such that $A \otimes_i B \rightarrow T$. As for Δ'', the proof now requires $\Delta''[X_i \leftarrow Y_i] \rightarrow B$ to be derivable.

(c) If $|\Delta''| = 1$ and $\Delta'' = C$ and $C \rightarrow B$, we are done. Either $\Delta \rightarrow T$, as in case (a), or $\Delta[A \leftarrow A'] \rightarrow T$.

(d) But suppose that not $C \rightarrow B$. We assign every string in $\mathfrak{R}(C)$ to $\mathfrak{R}(B)$. Then, $\mathfrak{R}(C) \subseteq \mathfrak{R}(B)$ and in particular, $d'' \in \mathfrak{R}(B)$. It is evident that substituting C for B in Δ'' amounts to identity, and assures derivability of $\Delta[C \leftarrow B] \rightarrow T$, *i.e.* reducibility of the string Δ with the rightmost occurrence of C substituted by B to T. Moreover, $d \in \mathfrak{R}(\Delta[C \leftarrow B])$ and $d \in \mathfrak{R}(T)$.

(e) Now suppose $|\Delta''| = 2$. Again, by binary derivability, $\Delta'' \rightarrow B$ if $C'\ C'' \rightarrow B$ for some C', C''. We take the same cycle (a)-(c) as above and show that either $\Delta'' \rightarrow B$ or $\Delta''[C' \leftarrow C'''] \rightarrow B$, or $\Delta''[C'' \leftarrow C''''] \rightarrow B$, or $\Delta''[C' \leftarrow C''', C'' \leftarrow C''''] \rightarrow B$. Moreover, $d'' \in \mathfrak{R}(\Delta''[\ldots])$ and $d'' \in \mathfrak{R}(B)$. Consequently, $d'+d'' \in \mathfrak{R}(A \otimes_i B) \subseteq \mathfrak{R}(T)$.

(f) If $|\Delta''| > 2$, the reasoning (a)-(d) applies to prove the derivability of some sequent $\Delta''' \rightarrow B$, where Δ''' is Δ'' except for a number of substitutions $X \leftarrow Y$ linearly bounded by $|\Delta''|$, such that $d'' \in \mathfrak{R}(\Delta''')$, $\Delta', \Delta''' \rightarrow T$ and $d \in \mathfrak{R}(\Delta'\ \Delta''')$.

(g) Now suppose that A is not compatible with T. Then substitute A by some A' that is compatible with T and ensure that $\mathfrak{R}(A) \subseteq \mathfrak{R}(A')$. Then go for B, as above. Inevitably, $\Delta[A \leftarrow A'] \rightarrow T$ or $\Delta[A \leftarrow A'/Other] \rightarrow T$, where Other is any combination of substitutions induced by inspection of Δ''.

(h) Suppose $|\Delta'| > 1$. Then, following the track (d)-(f) above for Δ'', create a derivable sequent $\Delta'''' \rightarrow A$ such that A is compatible with T, $d' \in \mathfrak{R}(\Delta'''')$ and Δ'''' is like Δ' except for a number of substitutions $X \leftarrow Y$ that is linearly upwardly bound by $|\Delta'|$. For that A, there must be a B such that $\Delta''' \rightarrow B$ where Δ''' comes from Δ'' as described above. Again, $d'+ d'' \in \mathfrak{R}(\Delta'''', \Delta''')$, $\Delta'''', \Delta''' \rightarrow T$ and Δ is like (Δ'''', Δ''') except for a finite number of substitutions $X \leftarrow Y$ linearly bounded by $|\Delta|$. Moreover, $|\Delta| = |\Delta'''', \Delta'''|$.

This ends the proof of completeness *salve* assignments for Categorial List Grammar. So, for a string Δ of power n, for which $\mathfrak{R}(\Delta) \subseteq \mathfrak{R}(T)$, one can find, in at most n substitution steps, each of which is decidable, a string $\Gamma = \Delta[X_i \leftarrow Y_i]$, $0 \leq i \leq n$, such that $\Gamma \rightarrow T$.

1.5.3 The fundamental asymmetry of merge

In section 1.3, it was put forward that append treats argument lists, originating from two antecedent categories in a merge, necessarily asymmetrically. At each side, all the arguments of one category will be on top of the arguments of the other category. This relative order correlates with the relative distance of argument strings to the string of the category: the types on top of the argument list induce strings that will be closer to the string induced by the category's head than the strings induced by lower types. Since append is the canonical operation on argument lists, this asymmetry is included in CLG.

Furthermore, the linearization of the strings in the two antecedent categories of each merge brings in another intrinsic form of asymmetry. It implies that at least at one side the strings induced by the arguments of an antecedent category will be separated from the string induced by the category's head. Here is a scheme of this pattern for a certain instance of a defined merge mode.

(68) $(\otimes_i/)$
PRIM\PLF~PLA/PRF~[SEC^i|PRA] $\otimes_i$ SEC\SLF~SLA/SRF~SRA $\rightarrow$
PRIM\SLF~(SLA⊕PLA)/a~(PRA⊕SRA)

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list

(69) s\u~[np^j]/ a~[vp^i, pp^k] $\otimes_i$ vp\ u~[ap^l] / u~[vp^m] $\rightarrow$
s\u~[ap^l, np^j]/a~[pp^k, vp^m]

(70) prim ∈ ℜ(s\u~[np^j]/ a~[vp^i, pp^k]), sec ∈ ℜ(vp\u~[ap^l]/u~[vp^m]),
vp ∈ ℜ(vp^m), np ∈ ℜ(pp^k), np ∈ ℜ(np^j), ap ∈ ℜ(ap^l)

(71) np + ap + prim + sec + pp + vp

Here, (71) represents a string that under assignment (70) may be the result of a derivation involving instantiation (69) of merge mode (68). The string *prim* is of the category that licensed *pp* but is separated from it by *sec*. Similarly, *sec* is of the category that brought in *ap* but is not adjacent to it in (71); yet, the type of *ap* was the top of the left argument list of the secondary antecedent category in merge (69). In this case, *sec* is not adjacent to any string induced by arguments of the category it belongs to. By definition of merge, however, the string associated with the primary category will be next to at least one string induced by the primary category's argument: the string induced by the cancelled argument and the secondary category. This asymmetric aspect of merge gives rise to an essential difference in status between strings of the primary category and strings of the secondary category. We hesitate to identify

this difference with *headness*, as made relevant to categorial grammar by Pollard (1984), Hoeksema and Janda (1988) and Jacobson (1991), among others. A head – if not empty – is the predominant string in a certain substring. Its category defines the structure immediately around it. From the asymmetry observed here, however, we can only derive the following property of *string connectedness* for strings in primary categories of a merge.

(72) *String connectedness*
A string s of category C – $s \in \Re(C)$ – is connected if it can be partitioned as *w+h+v* and *w* and *v* are induced by arguments in C, or w is induced by an argument in C and v is induced by an argument of the category w belongs to, or the other way around.

Merge imposes string connectedness on the string in its primary category. The string in the secondary category lacks this property, as one of its neighbours, namely the string associated with the primary category, is not induced, directly or indirectly, by one of the secondary category's arguments. It is precisely in this sense that the primary category defines a string beyond the string by which it is introduced.
String connectedness has a dual. If a string *w+p+s+v* is 'felt' to be defined by p then there is a category C such that $p \in \Re(C)$ and *C connects p in w+p+s+v*. It is not necessarily the case, however, that w+p+s+v is a constituent. Also, there does not have to be a sequence of categories D such that $w+p+s+v \in \Re(\Delta)$, that C is in Δ, that $\Delta \rightarrow C'$ for some C' (and that C introduces the head type of C').

Summarizing, merge imposes asymmetry in the relation between the antecedent categories in each of the following senses:

(a) precisely one category has one of its arguments cancelled, *i.e.* one category is primary, the other secondary
(b) the linear ordering of the categories in the merge induces the linear ordering of the strings
(c) append on argument lists is asymmetric, *i.e.* at each side, the arguments of one category are on top of those of the other category
(d) the strings associated with the primary category are connected in the string defined by the primary category.

Asymmetry, *i.e.* irreversibility of the (dependency) relationship between elements in natural language is the resultant of linearization. In CLG, linearization is an intrinsic component of the syntactic engine. It has been argued that in the grammar of natural language precedence and dominance relations could, or should, be separated. This has been a major topic in the develop-

ment of GPSG and HPSG. It was suggested for categorial grammar by Flynn (1983). Flynn stressed that constituency relations might be at another level of universality from linear restrictions. The topic recurs in the categorial mainstream in the format of specialized permutation modalities, meant to rearrange word order in a controlled way but independent of the combinatory, reductionist engine; see Moortgat (1998) for a detailed overview, and Hepple (1990) for the original approach. The modular view on precedence and dominance, word order and constituency, permutation and composition or any other duality that captures the two-dimensionality of natural language has been challenged by Kayne (1994). Kayne holds language to be essentially antisymmetric – and thus asymmetric – because linearization *entails* dependency. In his view, a certain linearization is not an accident of a particular language, but the prerequisite to interpretability and its reflection.

The present system is hardly reminiscent of Kayne's concept of grammar. But CLG respects the interdependency of word order and constituency by exploiting only one syntactic operation that accounts both for word order and hierarchical, interpretative dependencies. The asymmetry which CLG imposes on the linear dimension and the ordering of strings comes with an irreversible inequality of the primary and secondary categories in a merge. In this respect, it must be noted that typed logics, and typed lambda calculus in particular, incorporate antisymmetry by nature: the types of function and argument differ by definition, and whatever process can be reduced to functional application would embed antisymmetry this way.

The asymmetry of CLG is, of course, not an argument in favour of linearity as the anchoring dimension of natural-language grammar. It represents, though, a choice in this respect.

1.5.4 No manipulation of structure

An immediate property of the syntactic model advocated here is that it excludes the manipulation of structure. Once a configuration has been constructed, it is fixed by the simple fact of its being constructed. Of course the syntax must be able to handle discontinuities, but discontinuities arise from merging categories, just as continuous structures do.

The anchor of this property is the monotony of the derivation. In each and every derivational step exactly one cancellation occurs. As a consequence, the structure of the categories controls the derivation with an iron hand. No empty moves can occur. In this respect too, our syntax is rigid. The phrase structure

and the derivational structure – not necessarily the derivational process – coincide. To put it in other terms, there are no type-shifts in our syntax.
Since transformations and other manipulations of structure are not necessarily without purpose, the syntax' rigidity calls for a trade-off. In DELILAH, the burden is on the lexicon. Every variation in structure has to be introduced lexically. It leads to a massive number of distinct lexical specifications for each individual member of central syntactical classes like verbs. It makes no sense to try and express this in figures, as DELILAH's grammar is in no way complete. The specification load on the lexicon calls for a powerful lexical engine, producing all these forms and expressing two types of generalizations:

(a) all the specifications of a certain verb are *of* that verb – they live on one matrix lemma;
(b) the entire variety of each matrix lemma is induced by general properties of sentence structure, *i.e.* reflects the structure of the language.

In chapter 3, the structure of the lexicon will be addressed, and there we offer arguments with respect to the viability, constructability and operation mode of a lexicon with dimensions as sketched above. Here, it suffices to repeat that the operational rigidity of the syntax re-enters in the overall system as the tremendous complexity of the lexicon. This, however, is a choice – a radically lexicalist choice.

1.6 THE CASE FOR DUTCH

1.6.1 General Format

In the preceding section, the general properties of the standard two-place operation of merge were introduced. It was noted that every merge modality specifies a format for the two categories that go into the merge, and a format for the category that emerges from them. Several aspects of the operation are general. There is always exactly one type to be cancelled, in a designated position at the edge of one argument list. The stacks are addressed as lists. Cancellation requires typological identity of that argument and the head of the other stack. The head of the output category is invariably the head of the primary input category. The (remnants of) the argument lists are appended, pairwise

and directionwise. Argument lists are marked for affectedness, *i.e.* for having or not having undergone cancellation of one or more member types. Arguments are marked with one merging mode, indicating a specific mask on the input and output of the merge operation that is to cancel that argument.

Furthermore, the nature of the specifications that can be stated in a merge modality is fixed. They come from a limited set. Argument lists of the input may be required to be empty or non-empty; non-emptiness means that the stack contains at least, or exactly, one argument, specified or not – this difference will be ignored in the calculations. Moreover, lists are marked affected or not-affected, initially unaffected by assignment and subsequently affected or unaffected by computation or specification. Specifications on argument lists are not necessary, though. The output category is specified in terms of the input components. The way of appending the input lists is determined. So is the affectedness marking on the output lists. Here is an overview of the possible specifications on input and output for a rightward cancelling modality *i*.

(73) *phrase1:* PRIM \ PLF~PLA / PRF~[SEC^i| PRA]
$\otimes_i$
phrase2: SEC\ SLF~SLA/SRF~SRA
$\rightarrow$
phrase1+phrase2: PRIM\LF~LA/RF~RA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

specifications of PRF, SRF: *affected*, *unaffected* or none
specifications of PLF, SLF: *wh, affected* or *unaffected*
specifications of PLA, PRA, SLA, SRA: *empty*, *nonempty* or none
specifications of LF: *wh, affected, unaffected*, depending on the values of PLF and SLF and the values of PLA and SLA
specifications of RF: *affected* (normally; exceptionally *unaffected*)
specifications of LA: *PLA + SLA* or *SLA + PLA*
specifications of RA: *PRA + SRA* or *SRA + PLA*

Now we can compute the total number of different modalities that can be specified within these limits. There are twelve items that may be specified: six argument lists and six flags. For each of the input arguments lists three options are available: the list is empty, the list is non-empty with a specified type on top, or the list is unspecified. The options for the two input lists are formally independent. Each of the input flags also has three options available: the list may have been *affected* by prior cancellations, it may be affected (in

fact, emptied) by a rule involving cancellation under a *wh*-mode, and it may be *unaffected* or in its lexical state. The choices are, again, formally independent of each other, but the *wh*-flag is only realized for flags of left lists. For each of the four components of the output category only two options are available, since they must be fully determined by the rule. These options are also independent, with the possible exception of the value of Rf.

Any mix of these options will define a modality. That is: a modality is a 12-tuple <Plf, Pla, Prf, Pra, Slf, Sla, Srf, Sra, Lf, La, Rf, Ra>. As an example, a modality could be <u, a, [np | [ ]], a, [ ], u, _, a, l+r, a, r+l >. Consequently, within this format at most $3^8.2^4$ = 104976 different modalities can co-exist. Materially, however, this limit is too high. Every requirement as to the emptiness of an input argument list fixes the value (for append) of the output list in that direction, since [ ] ⊕ L = L ⊕ [ ] = L for any L. Thus, the values for La and Ra are not independent of the values for Pla and Sla and for Pra and Sra, respectively. Practically, then, the limit is $(3^2\text{-}2^2).3^4.2^2(2^4 + (3^2\text{-}2^2))$ = 34020 different modalities for rightward merge.

Moreover, the flags of the output argument lists are completely determined by the values of the flags and lists of the input categories. They are therefore not so much part of a specification but computed at each merge, with the exception of Rf's value being un*affected*. This reduces the number of modalities down to half the computed number, *i.e.* 17010 and almost to one quarter (say, 9000) if we take the exception to be exceptional. Moreover, it is likely that we could establish various kinds of clusters of specifications which might exclude or induce each other. For example, it is unlikely that a certain argument list is required to be both empty and either affected or unaffected. Affectedness indicates that the phrase bearing such an argument list is not lexical but composed. Unaffectedness has the opposite flavour. How likely is it that we will discover a merge process that imposes conditions both on the internal structure of a category's component and on its 'flattened history'?

A categorial list grammar will be any set of modalities, defined on a given set of types. This view raises all kinds of questions as to the consistency of these sets, as well as with respect to the practical and theoretical equivalence between classes of modalities, and the decidability and expressibility of certain properties. Most of these questions cannot be addressed here.

In the section on patterns of discontinuity, we will reconsider the variety of merge modalities. But, clearly, the number of possible modalities ensures that no grammar constructed from them is trivial. Each selection of modalities, even if the selection is very small, as will normally be the case, represents a very specific grammar.

In practice, and in our system, the number of modalities will be relatively small. In the fragment described below, no more than twenty modalities in each direction are addressed, despite the considerable complexity of the phenomena covered. The reason is simple: the formal independence of the twelve parameters is not maintained in the practice of natural-language grammar – grammar is a subtle network of co-occurrence restrictions on parameter values.

1.6.2 A concise syntax of Dutch

1.6.2.1 Flagging argument lists

All the modalities used in the DELILAH grammar of Dutch enjoy some kind of input specification. The single mode that does not specify any input component is not used. It is unlikely that any reasonable grammar of any language will employ that mode. It amounts to the absolute insensitivity of an element for the structure of its environment. Only extra-sentential elements, like interjections of sighing, qualify for this form of context-freeness, but, even there, constituency may play a role.

The output category of the merge is always fully specified. The flags of the output argument list, appended from a pair of input lists, are computed from the flags of the input lists that are appended, and from the lists' status. The flag in the active direction of the merge will almost invariably get the value *affected*, abbreviated *a~* or *w~*. The affectedness value *w~* is assigned if the cancelled argument required application of the *wh*-modality. Most other cancellations lead to the affectedness value *a~*. This will be explained below. The other value is marked *u~*, for *unaffected*. The other output flag, the one for the argument list in the passive direction, is computed deterministically according to chart (74), the table of possibilities for two input lists and one output list in the same (passive) direction.

(74) Table of flags for output argument list: *flag ~ list*

	input 1	*input 2*	*output flag*
a.	w~_	a~[_|_]	w~
b.	u~_	w~[_|_]	w~
c.	u~_	a~[_|_]	a~
d.	_~[ ]	_~[ ]	u~
e.	u~_	_~[ ]	u~
f.	u~_	u~_	u~
g.	a~[_|_]	a~_	a~

The situation where two input lists are flagged *w~* does not occur, because of the way the *w~* flag is embedded. Just like `a~`, the flag is assigned only combinatorially and, by definition, to an empty list in the active direction. The flag can be transferred, however, to non-empty lists according to (74) a. Derivations involving the computations (74)a and b, however, are bound to be pathological and dead-ending: the *w~* flag is to occur in categories that are completed (no arguments left to be cancelled), because *wh*-extraction induces strong islands.
The most important aspect of the computation is the restoration of unaffectedness by (74)d and e. The moral here is that whenever an empty list is appended, the nature of its emptiness – emptied or lexical – gets out of sight. This embodies locality. If a constituent has completed its agenda at one side, the next merge renders its completion invisible or irrelevant for further merges; its history and its structure are enveloped in the new structure. Note, furthermore, that the output list can be marked *affected* only if at least one of the input lists is.

In the sections to follow, all the rules of the DELILAH grammar are presented, explained and exemplified. The modalities are indicated by short memo-subscripts on the merge-operator and as extensions to arguments. The rules are named after the modalities. The flag of the output list in the passive direction is to be computed according to the scheme above, and is not specified in the rule, but invariantly dubbed *LF*.

1.6.2.2 Rightward rules

1.6.2.2.1. */^isl* FOR SATURATED CONSTITUENTS

This mode requires that the secondary category is completed. It does not allow any part of a secondary agenda to be transferred to the output.

(75) PRIM \ _~PLA / _~[SEC^*isl*|PRA] $\otimes_{isl}$ SEC \ _~[ ] / _~[ ] → PRIM \ LF~PLA / a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

Because of the emptiness of the secondary argument lists, the appends are trivial. The standard application of this merge is the consumption of a noun phrase. Noun phrases are closed domains in Dutch. They do not allow for any discontinuity in their components. All noun phrases in argument lists select the */^isl* cancelling mode.

Here is an example of */^isl*:

(76) `q\u~[]/u~[np^isl, vp^x]` $\otimes_{isl}$ `np\u~[]/a~[]` → `q\u~[]/a~[vp^x]`
wil 'wants' ⊗ *de man* 'the man' → *wil de man*

Other candidates for cancelling under */^isl* are all those constituents that have been identified as absolute islands, like (embedded) questions. One of the standard categories for a verb selecting an embedded question must be `vp\u~[]/u~[q^isl]`.

1.6.2.2.2. */^transp* FOR UNSATURATED CONSTITUENTS

This merge mode is in a certain sense the opposite of */^isl* in that it requires the secondary category to be transparent instead of closed. It specifies that the argument lists in the passive direction – leftward – must both be unaffected. Being lexically assigned as such is one of the ways to meet this condition. The left agendas of both input categories are transferred to the resulting category. On the other hand, the active list of the secondary category is bound to be empty (or emptied, for that matter). Since this is a rightward rule, this requirement provokes a rightward embedding structure (...X_1(...X_i(...X_n)..)..), where the right arguments of the secondary category must be met before the constituent itself is subject to merge. As a consequence, this merge mode induces a kind of head adjunction, in that the phrases introducing the primary and the secondary head types end up being neighbours in the string.

(77) PRIM \ u~PLA / _~[SEC^*transp*|PRA] $\otimes_{transp}$ SEC \ u~SLA/ _~[ ] →
PRIM \ LF~SLA+PLA / a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

The secondary passive argument list ends up on top of the passive arguments brought in by the primary category. The secondary arguments will therefore be first to meet at the left-hand side.

The typical case here is the cancellation of the infinitival verbal complement of a verb-raising (semi-)auxiliary. This was also the example given in (41). The particular appending of the passive argument lists under this merge accounts for the famous crossing dependencies in the Dutch verb cluster. In the following example, the *np*s are indexed to show this effect.

(78) `s\u~[np1^isl]/u~[vp^transp]` $\otimes_{\text{transp}}$
`vp\u~[np2^isl, np3^isl, np4^isl]/a~[]` →
`s\u~[np2^isl,np3^isl,np4^isl,np1^isl]/a~[]`
kan 'can' ⊗ *laten geven* 'let give' → *kan laten geven*

This mode is a good example of the combination of linearization and derivation, and of grammar and control. The emptiness condition on the secondary passive argument list forces this category to complete its right-hand agenda before getting involved in this merge. Unaffectedness-marking on the two left lists and the append specification make the string instantiations of primary left arguments peripheral to the substring formed here and make them precede their string counterparts of the secondary arguments. Any string introduced to meet the demands of the primary category's left agenda is bound to occur to the left of the 'secondary' strings. Recall that no further operation can change that state of affairs. The only operations that the new argument list can be submitted to are cancelling of its top element and/or appending. Neither of these operations changes the internal order established by /^*transp*.

There is an alternative for (77). It allows for an alternative derivation if the secondary category's left agenda is lexically empty, *i.e.* is empty and unaffected at merge. This would be its format:

(79) Prim \ _~Pla / _~[Sec^*transpb*|Pra] $\otimes_{\text{transpb}}$ Sec \ u~[ ]/ _~Sra →
Prim \ Lf~Pla / a~Sra+Pra

Prim: head of *prim*ary category; *Plf: p*rimary category's *l*eft argument list *f*lag;
Sec: head of *sec*ondary category; *Sra: s*econdary's category's *r*ight *a*rgument list;
*Lf: l*eft flag (in resulting category); *Ra: r*ight argument list (of resulting category)

The primary left agenda no longer has to be unaffected. The primary category may have consumed one or more of its left arguments before entering into this merge. This would not harm the 'crossing dependency' effects here, since the merge presupposes that there are no left arguments to the secondary category. Moreover, it defines an append at the right-hand side, guaranteeing that the secondary category's arguments are met first. Here, this does not amount to crossing dependencies, but to completion of the secondary agenda before that of the primary right agenda. That is, in effect, the same result as was reached more rigidly by the emptiness requirement on Sra in (77). Thus, (79) defines a derivationally liberalized sub-case of, for example, verb clustering. There is no need, however, to compute the verbal complex of Dutch in a syntactically and semantically adequate way.

1.6.2.2.3. */^open* FOR OPTIONAL DISCONTINUITY

An intriguing variety of */^transp* eliminates the unaffectedness constraint on the secondary left agenda. Consequently, it allows the secondary category to consume the left agenda or part of it without destroying its fitness for this merge. A string associated with the primary category can therefore connect to a string of which the left edge was not lexically associated with the secondary category.

(80) PRIM \ u~PLA / _~[SEC^*open*|PRA] $\otimes_{\text{open}}$ SEC \ _~SLA/ _~[ ] →
PRIM \ LF~SLA+PLA/ a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

Note that the (remaining) left arguments of the secondary category at merge have to be put on top of the primary ones, which are supposed to be unaffected. This merge does not mix up the left arguments differently, but puts fewer restrictions on the pre-merge behaviour of the secondary category. The only difference between */^open* and */^transp* is the affectedness of the left argument list of the secondary category.

The canonical example in Dutch for this merge is the complementation of verbs that induce the 'third construction' (Den Besten *et al.* 1988, among others). Prominent among them is *proberen* 'to try', which occurs in each of the following, semantically equivalent patterns. All other scramblings are ungrammatical.

(81) … dat jij probeerde Ramses 'Egyptian Nights' te laten lezen
… *that you tried Ramses 'Egyptian Nights' to have read*
… 'that you tried to have Ramses read Egyptian Nights'
(82) … dat jij Ramses probeerde Egyptian Nights te laten lezen
(83) … dat jij Ramses Egyptian Nights probeerde te laten lezen

Here is the instance of */^open* that would give rise to (82):

(84) `s\u~[np1^isl]/u~[vp^open]` $\otimes_{\text{open}}$ `vp\a~[np2^isl]/u~[]` →
`s\a~[np2^isl, np1^isl]/a~[]`
probeerde 'tried' ⊗ Egyptian Nights te laten lezen '*Egyptian Nights* to have read'
→ probeerde *Egyptian Nights* te laten lezen

The value for affectedness of the secondary right argument list results from two empty lists being appended at the previous merge, according to computation (74)d.
This merge mode is also basic to the arguments of adjunctive automorphisms. Verbal and sentential qualifiers may occur in a lot of positions inside verbal complexes. Consider the variety of positions for the instrumental adjunct *met een gedicht* in the following sentences.

(85) … dat Henk met een gedicht Jan de kast wilde laten openen
… *that Henk with a poem Jan the cupboard wanted have open*
… 'that Henk wanted to have John open the cupboard with a poem'
(86) … dat Henk Jan met een gedicht de kast wilde laten openen
(87) … dat Henk Jan de kast met een gedicht wilde laten openen

This flexibility reflects the combinatorial potency of *proberen*, and makes the merge mode */^open* to an essential ingredient of a Categorial List Grammar of Dutch.

1.6.2.2.4. */^penins* FOR NEAR ISLANDS

The next mode accounts for the merge with near islands, *i.e.* constituents that have on their passive (leftward) agenda one specified argument at most. In practice, this merge is used to account for combinations with constituents that lack fronted or leftward dislocated elements. A constituent licenses a dislocated element *iff* its left argument list contains a *type^mode* pair `xp^w`. The rule comes in two mutually exclusive formats, differing only in the specification of secondary passive (leftward) list. This double format leads to a disjunction of specifications.

(88) PRIM \ _~PLA / _~[SEC^*penins*|PRA] $\otimes_{\text{penins}}$ SEC\ _~[ ]/_~[ ] →
PRIM \ LF~PLA / a~PRA

(89) PRIM \ _~PLA / _~[SEC^*penins*|PRA] $\otimes_{\text{penins}}$ SEC\ _~[_^*w*]/_~[ ] →
PRIM \ LF~PLA+[_^*w*]/ a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

It is grammatically very important that in (89) the append in the passive direction suppresses the cancellation of the `^w` argument in favour of cancellation of the primary category's left arguments. In this way, the intended dislocated element can only be absorbed as the final step of completing the left agenda. That is also what the merge modality `\^w` will specify as a requirement. The

example below illustrates a case of long-distance dislocation, where an argument of a deeply embedded infinitival complement is 'wh-ed', as in (91).

(90) `s\u~[np^isl]/u~[vp^penins]` $\otimes_{penins}$ `vp\a~[np^w]/u~[]` $\rightarrow$
`s\a~[np^isl,np^w]/a~[]`
dwong 'forced' $\otimes$ *een boek te geven* 'a book to give' $\rightarrow$ *dwong een boek te geven*

(91) Wie zei Henk dat Agnes hem dwong een boek te geven?
Who said Henk that Agnes him forced a book to give
'Who did Henk say Agnes forced him to give a book?'

In order to ensure that in this case the `^w` argument is properly transferred under composition, all appendings of the left argument list – independently of the particular merge mode they appear in – should be defined in such a way that `^w` arguments are stacked down. To see why, consider the case of */^transp* in section 1.6.2.2.2. If one of the complement's arguments is of the dislocated breed, this argument has to be suppressed in the output in favour of the arguments of the primary category. Otherwise, it would be at the top of the stack at the wrong moment in the derivation, namely when it is not the only item in the left agenda. That is, the output of the crucial */^transp* merge in (92) should be as in (93) rather than as in (94).

(92) Wie denk jij dat ik Agnes een boek kan laten geven?
Who think you that I Agnes a book can have give
'To whom do you think that I can have Agnes give a book?'

(93) `s\u~[np^isl]/u~[vp^transp]` $\otimes_{transp}$ `vp\u~[np^isl,np^isl,np^w]/a~[]` $\rightarrow$
`s\u~[np^isl,np^isl,np^isl,np^w]/u~[]`

(94) `s\u~[np^isl]/u~[vp^transp]` $\otimes_{transp}$ `vp\u~[np^isl,np^isl,np^w]/a~[]` $\rightarrow$
`s\u~[np^isl,np^isl,np^w,np^isl]/u~[]`

We can make sure that every `^w` starts at the bottom of its stack in the lexicon. For this purpose, we only have to change the standard append, applied to argument lists, in such a way that `^w` arguments at the bottom of either stack remain there. This is relevant to the tail of the list that is to become the upper part of the stack. An adequate reformulation of *append* for argument lists in a Prolog-like fashion will be this:

(95)
```
clg_append( L, R, LR') ←
                append( Lmin, [X^w], L),
                !,
                append( R, [X^w], R'),
                append( Lmin, R', LR').
clg_append( L, R, LR) ←
                append(L, R, LR).
```

Before gluing lists together, the left argument list is checked for the occurrence of a dislocated argument; if present, it is stacked down.
From now on, *append* at argument lists is considered to be reformulated as *clg_append*. Note that (95) does not interfere with the context-freeness of this operation (cf. section 1.8.3).

1.6.2.2.5. */^transpipp* FOR INFINITIVE *PRO PARTICIPIO*

This mode is a variation of */^transp*, deviating only in the additional specification that the secondary active list must be affected. As a consequence, no lexical category can comply with the secondary requirements of this mode. Its output conditions are the same as in (77).

(96) PRIM \ u~PLA / _~[SEC^*transpipp*|PRA] $\otimes_{\text{transpipp}}$ SEC\ u~SLA/ a~[ ] →
PRIM \ u~SLA+PLA / a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

Just like */^transp,* this merge mode is mainly applied for dealing with the verb cluster. This particular variation is partly responsible for the so-called *infinitivus-pro-participio* effects. An auxiliary verb may select a participle as the head of its verbal complement, but can also require an infinitive when the complement itself contains a verbal cluster. Here are some relevant data; note the form of *willen* 'to want' in the examples.

(97) Jeroen had die baan wel gewild
Jeroen had that job wanted
'Jeroen would have wanted that job'
(98) Jeroen had die baan wel willen nemen
Jeroen had that job want accept
'Jeroen would have wanted to accept that job'
(99) * Jeroen had die baan wel willen
Jeroen had that job want
(100) * Jeroen had die baan wel gewild nemen
Jeroen had that job wanted accept

The mode */^transpipp* is assigned to the verbal arguments of auxiliaries that are sensitive to the *ipp*-effect. Unfortunately, that the secondary active argument list is affected is not the whole story here. It appears that the argument cancelled in the construction of the secondary category must be *vp*, and nothing else:

(101) Hij had gewild dat ik er bij was
'he had wanted that I was present'
(102) * Hij had willen dat ik er bij was
he had want that I was present

This particular requirement cannot be expressed in our syntactic formalism. It is handled by the specification in the template that the candidate *ipp* is a semi-auxiliary itself – a qualification that is restricted to a certain group of verbs taking infinitival complements. This group is qualified in Cremers (1983) as taking infinitival complements that can only be interpreted below propositional level. *Zeggen* 'to say' is not in that class. It may take infinitival complements, but they must be interpreted as propositions. Therefore, it should be excluded from *ipp*-effects, and it is:

(103) Merel had gezegd te blijven zingen
Merel had said to keep sing
'Merel said that she would keep on singing'
(104) * Merel had zeggen te blijven zingen
Merel had say to keep sing

This justifies the qualification of *ipp*-candidates as semi-auxiliaries.

1.6.2.2.6. */^sentop* FOR WH-ISLANDS

This mode requires the secondary passive list to be flagged *w*. This flag results from application of merge mode \^wh, to be discussed below. With this it indicates that the last left mode dealt with a cancellation under this merge. The present mode consumes that flag.

(105) PRIM \ u~PLA / _~[SEC^*sentop*|PRA] $\otimes_{\text{sentop}}$ SEC \ w~[ ]/ _~[ ] →
PRIM \ u~PLA / a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

The mode canonically applies to the first of the pair of categories to which an argument *wh*-phrase is assigned. In DELILAH's lexicon, a wh-word like *wie* 'who' is assigned to a pair of co-occurring categories, $C^{left} * C^{right}$ (see also chapter 3). The asterisk in the category disappears as soon *wie* is used combinatorially. Two adjacent categories remain. The rightmost of these is simply `np\u~[]\u~[]`, dealing with the selectional properties of the word as an argument to some other phrase in the sentence. The leftmost category that comes

with *wie* selects the sentence that has been built by the merge in which the rightmost category evaporates, and qualifies it. It may be qualified as a question, a postnominal modifier, or whatever operation *wh*-elements can perform. Here is the example of *wie* introducing a question; the operation $\otimes_{wh}$ is introduced below as part of a leftward rule.

(106) wie :: `q\u~[]/[s^sentop] * np\u~[]\u~[]`
(a) `np\u~[]\u~[]` $\otimes_{wh}$ `s\u~[np^w]/u~[]` → `s\w~[]/u~[]`
(b) `q\u~[]/[s^sentop]` $\otimes_{sentop}$ `s\w~[]/u~[]` → `q\u~[]/a~[]`
(b)*(a) `q\u~[]/[s^sentop]` $\otimes_{sentop}$ `np\u~[]\u~[]` $\otimes_{wh}$ `s\u~[np^w]/u~[]` → `q\u~[]/a~[]`
wie 'who' ⊗ *slaapt* 'sleeps' → *wie slaapt*

When the *wh*-term is not an argument, it has a single category. This category cannot expect a particularly constructed right partner. For example, *hoe* 'how' just requires its sentential argument to have an unaffected left argument list, thus enforcing that the last and decisive merge of this argument has not been leftward – compare the flag computation (74). The relevant mode is given below; it is as specific as (105) and therefore an exclusive alternative to it.

(107) PRIM \u~PLA /_~[SEC^*sentop*|PRA] $\otimes_{sentop}$ SEC \u~[]/ _~[] → PRIM \u~PLA /a~PRA
(108) `q\u~[]/[s^sentop]` $\otimes_{sentop}$ `s\u~[]/a~[]` → `q\u~[]/a~[]`
hoe 'how' ⊗ *vlieg jij* 'fly you' → *hoe vlieg jij*

1.6.2.2.7. */^adj*

The last right merge is a very tolerant one: the primary category does not affect the status of the secondary category. The latter imposes all its specifications on the output. Even the output flag in the active direction is dictated by the secondary category, instead of being *a~* or *w~* by default. The combinatorial and derivational effect of this merge is almost zero. The mode is particularly suited for the merges not rising from selection or subcategorization, but rather from adjunction or modification. Under these circumstances, the head types of the two categories will be equal, *PRIM* = *SEC*.

(109) PRIM_~PLA/_~[SEC^*adj*|PRA] $\otimes_{adj}$ SEC_~SLA/RF~SRA → PRIM\ LF~PLA+SLA/RF~SRA+PRA

As a typical case of this mode, consider the merge of a preposition introducing a verbal adjunct with a displaced *r*-pronoun as nominal complement and the verb:

(110) … vp\u~[r^*wh*]/u~[vp^*adj*] $\otimes_{adj}$ vp\u~[np^*isl*]/u~[ ] →
vp\u~[np^*isl*, r^*wh*]/u~[ ]
(waar heeft hij het boek) mee 'with' ⊗ *geschreven* 'written' →
mee geschreven
'(what has he the book) with written'
what did he write the book with?

The specific order in the left output list follows from the redefinition of *append* (95).

This completes the set of rightward merge modalities, activated in the grammar of Delilah Dutch. There are seven of them, one of which, to wit */^transpipp,* is merely added to handle a peculiarity of Dutch syntax. The other six represent core modalities, in that they represent essential combinatorial processes in Dutch sentence formation. Some of the rightward merges discussed below are labelled like left-directional merges. Except for the island modalities, which represent rigid cancellation without any transfer of combinatorial agendas, they seem to be only directional images. In fact, there is little symmetry in the selection of merges for dealing with Dutch syntax. Linearization itself is the source of asymmetry.

1.6.2.3 Leftward Rules

It is attractive to call leftward cancellations backward, and rightward ones forward, following the terminology first used by Ades & Steedman (1982). It is also misleading, however, as this terminology refers to the parsing process instead of to the linearity of sequence formation. The direction of an argument may influence but will not control the parsing process.

1.6.2.3.1. \^*isl* for saturated constituents

The most rigid form of rightward cancellation (75) does have its leftward mirror image. The arguments cancelled under this mode are basically of the same types as the ones which are submitted to */^isl*: nominal phrases, determiner phrases, quantifiers. The standard case is the object arguments of verbs. The canonical position of these objects is to the left of the verb, if Dutch is an SOV lookalike.

(111) Sec_~[ ]/_~[ ] $\otimes_{isl}$ Prim_~[Sec^*isl*|Pla]/_~Pra → Prim\a~Pla/Rf~Pra

Prim: head of *prim*ary category; *Plf: p*rimary category's *l*eft argument list *f*lag;
Sec: head of *sec*ondary category; *Sra: s*econdary's category's *r*ight *a*rgument list;
*Lf: l*eft flag (in resulting category); *Ra: r*ight argument list (of resulting category)

1.6.2.3.2. \^*transp* FOR AUXILIARY INVERSION

This mode is labelled like */^transp* since it applies to order variants of configurations in which that mode is invoked. Nevertheless, it enforces different types of input conditions. The primary category is forced to cancel its left argument first, whereas the secondary category is bound to be unaffected in both directions, and therefore lexical. The general formulation would be thus:

(112) SEC\u~SLA/u~SRA $\otimes_{\text{transp}}$ PRIM_~[SEC^*transp*|PLA]/u~PRA → PRIM\a~SLA+PLA/u~PRA+SRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

The canonical application for this merge is the case of auxiliary inversion: certain auxiliaries or semi-auxiliaries may take the head of their verbal complements to the left, leaving the rest of their – or that head's – complement, if any, to the right. (For an intriguing and almost discouraging analysis of Germanic verb cluster phenomena see Wurmbrand 2006, and for encouraging efforts to formalize the phenomena in different syntactic settings see Van Dreumel and Coppen 2003 and Bouma and Van Noord 1996). Here are relevant examples with the modal auxiliary *kunnen*.

(113) … dwingen kan te werken
… *force can to work* 'can force … to make … work'
(114) … * willen kan laten werken
… *want can let work*
(115) … kan dwingen te laten werken
(116) … * dwingen te laten kan werken
(117) … * dwingen te laten werken kan

The phenomenon is not restricted to infinitival complements but also occurs with participles and the temporal and passive auxiliaries, under slightly different conditions. In particular, the auxiliary triggering the inversion can be infinitival just as well as finite.

(118) … dat Gezinus in het parlement probeerde te worden gekozen
… *that Gezinus in the parliament tried to be chosen*
… that Gezinus tried to be chosen in parliament
(119) … dat Gezinus in het parlement gekozen probeerde te worden
… *that Gezinus in the parliament chosen tried to be*

The relevant merge in constructing (113), for example, is:

(120) `vp\u~[np1^isl]/u~[vp^open]` $\otimes_{\text{transp}}$
`s_vn\u~[vp^transp,np2^isl]/u~[]` →
`s_vn\a~[np1^isl,np2^isl]/u~[vp^open]`
dwingen 'force' ⊗ *kan* 'can' → *dwingen kan*

Since primary categories introducing this mode will generally be auxiliaries or semi-auxiliaries with just one (verbal) complement, we may restrict the mode to situations in which the primary right argument list is empty, except for the verbal complement. Under these assumptions, format (112) boils down to:

(121) Sec\u~Sla/u~Sra $\otimes_{\text{transp}}$ Prim_~[Sec^*transp*|Pla]/u~[ ] →
Prim\a~Sla+Pla/u~Sra

Prim: head of *prim*ary category; *Plf:* *p*rimary category's *l*eft argument list *f*lag;
Sec: head of *sec*ondary category; *Sra:* *s*econdary's category's *r*ight *a*rgument list;
Lf: *l*eft flag (in resulting category); *Ra:* *r*ight argument list (of resulting category)

1.6.2.3.3. \^*open*

The most liberal of all merges does not impose any input restrictions whatsoever. As such it belongs to the small class of modes with these characteristics, differing from each other only in the output specifications.

(122) Sec_~Sla/_~Sra $\otimes_{\text{open}}$ Prim_~[Sec^*open*|Pla]/_~Pra →
Prim\a~Sla⊕Pla/Rf~Sra⊕Pra

The standard application here is to the automorphic argument of an adjunct, *e.g.* by an adverbial constituent to a VP. In that case, however, we might as well opt for some more restrictive version. For example, we could consider the primary category itself to be 'closed', in having no arguments unsaturated, except the one to be cancelled. Such a restriction would imply that the primary category itself represents an island; this, indeed, has generally been considered to be an identifying feature of adjuncts since Ross (1967). Here is this more restricted version.

(123) Sec_~Sla/Rf~Sra $\otimes_{\text{open}}$ Prim_~[Sec^*open*]/_~[ ] → Prim\a~Sla/Rf~Sra

Since the flag of an empty list hardly charges the relevant flag in the output, all output lists stem from the secondary category and the passive flag may also be equal to the passive input of the secondary category. The dominance of the secondary category, again, seems to be in accordance with the nature

of adjunctival modification. The primary category merely intervenes in the secondary structure. In the following example, *heeft* 'has' introduces the secondary category, and *waarschijnlijk* 'probably' the primary category to an \^*open* merge.

(124) Elke student heeft waarschijnlijk de boeken willen verkopen
Every student has probably the books want sell
'Every student probably wanted to sell the books'

For the sake of generality, however, we will stick with the liberal (122). It is subsumed by the more restrictive (123).

1.6.2.3.4. \^*wh* FOR THE MOTHER OF DISCONTINUITY

This is the rather basic merge mode that cancels a leftward-dislocated complex for its licensing argument. It is supposed to be the final move in the completion of an argument structure. At every previous merge, the argument with this mode was suppressed, in the sense of being kept at the bottom of the output stack, according to the revised definition of list append as clg_append in (95). Thus, when cancelling an argument under mode \^*wh*, there is no remaining active list in the primary category. The merge is designed to consume elements that in other frameworks may be considered to live in the specifier of a complementizer phrase. At output, the left argument list is flagged *w* for being affected by this particular cancellation (cf. section 1.6.2.1).

(125) Sec_~[ ]/_~[ ] $\otimes_{wh}$ Prim_~[Sec^*wh*]/_~Pra → Prim\w~[ ]/Rf~Pra

It seems mandatory for the secondary active (leftward) list to be empty. Since this category is assumed to represent a left edge element, it must be completed at that side. The condition that the secondary passive list is also empty amounts to the claim that only arguments can be cancelled in this way, not heads. In this sense, the condition reflects the percolation of a *wh*-marking to some maximal projection, inducing pied piping.
Applications of this merge mode comprise standard wh-movement phenomena, topicalization with verb-second and subject lefting. As for Dutch, we take them to be the same because they exclude each other: a classic structuralist argument. These structures are exemplified in the following set (for /^*sentop* see section 1.6.2.2.6); recall that *wh*-arguments come with double categories (star-categories; see (106)).

(126) `q\u~[]/u~[s^sentop]` $\otimes_{\text{sentop}}$ `np\u~[]/u~[]` $\otimes_{\text{wh}}$ `s\u~[np^wh]/a~[]` → `s\w~[]/u~[]`
wie 'who' $\otimes^2$ *denk jij dat werkt* 'think you that works' → *wie denk jij dat werkt* 'who do you think is working?'

(127) `np\u~[]/u~[]` $\otimes_{\text{wh}}$ `s\u~[np^wh]/a~[]` → `s\w~[]/u~[]`
die man 'that man' ⊗ *heb ik zien slapen* 'have I see sleep' → *die man heb ik zien slapen* 'that man I saw sleeping'

(128) `np\u~[]/u~[]` $\otimes_{\text{wh}}$ `s\a~[np^wh]/a~[]` → `s\w~[]/u~[]`
ik 'I' ⊗ *slaap* 'sleep' → *ik slaap* 'I am sleeping'

In (126), it is assumed that lexical argumentative *wh*-elements come with two categories, rather than one (see also at */^sentop* above). That is the main difference between these elements and the dislocated entities in (127) and (128). The relevant merges under *\^wh* as such are equal.

1.6.2.3.5. *\^adj* FOR FREELY OCCURRING ADJUNCTS

This is the leftward counterpart of */^adj*, introduced in section 1.6.2.2.7. It has the same motivation and application.

(129) SEC\SLF~SLA/_~SRA $\otimes_{\text{adj}}$ PRIM_~[SEC^*adj*|PLA]/_~PRA → PRIM\SLF~SLA⊕PLA/RF~PRA⊕SRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

Here is a characteristic example:

(130) … `s\u~[]/a~[]` $\otimes_{\text{adj}}$ `s\u~[s^adj,r^wh]/u~[]` → `s\u~[r^wh]/u~[]`
… *zaten zij* 'sat they' ⊗ *op* 'on' → *(daar) zaten zij op* 'on (that) they sat'

1.6.2.3.6. *\^part* FOR PARTICLES

Finally, in the leftward division, we need the inverse of an adjunct merge: a selected item the cancellation of which as a secondary category does not provoke affectedness.

(131) SEC\u~[ ]/u~[ ] $\otimes_{\text{part}}$ PRIM\u~[SEC^*part* |PLA]/_~PRA → PRIM\u~PLA/RF~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

There are all kinds of restrictions on the secondary input, as it is mainly applied to lexical particles that may occur separate from their licenser. Those particles behave as if they are part of that licensing phrase, in that they may

occur in positions in which other arguments of that licenser cannot occur. The example below has a resultative small clause predicate *groen* 'green' – such a predicate may occur with almost any transitive verb. It can take any position in a cluster where it is accessible as the left argument of the verb to be met first, independently of verb-raising configurations, though not every position is felicitous for every type of particle – but the \^*part* modality generalizes over 'real' particles and resultatives:

(132) ... dat Agnes Henk de auto groen wil laten verven
... *that Agnes Henk the car green wants have paint*
... 'that Agnes wants to have Henk paint the car green'
(133) ... dat Agnes Henk de auto wil groen laten verven
(134) ... dat Agnes Henk de auto wil laten groen verven

The relevant merge for (134) is

(135) `ap\u~[]/u~[]` $\otimes_{\text{part}}$ `vp\u~[ap^part,np^isl]/u~[]` → `vp\u~[np^isl]/u~[]`
groen 'green' ⊗ *verven* 'paint' → *groen verven* 'paint green'

Because the *u*-flag at the left argument list remains in the output, the resulting category can participate in the verb clustering directed by the */^transp* mode of the complement of *laten* (see section 1.6.2.2.2.).

Since particles of the sort targeted by this mode in a certain sense disturb the order of cancellation of the verb's left arguments, it makes sense to consider a little more liberal version of (131), where it is not required for the particle to be at the top of its stack.

(136) Sec\u~[]/u~[] $\otimes_{\text{part}}$ Prim\u~[... Sec^*part* ...]/_~Pra → Prim\u~Pla/Rf~Pra

Prim: head of *prim*ary category; *Plf:* *p*rimary category's *l*eft argument list *f*lag;
Sec: head of *sec*ondary category; *Sra:* *s*econdary's category's *r*ight *a*rgument list;
Lf: *l*eft flag (in resulting category); *Ra:* *r*ight argument list (of resulting category)

It introduces an *anytime* cancellation strategy for a designated class of constituents, namely exactly those that are marked with the cancellation mode \^*part* by the lexicon. On the one hand, this complicates the derivation: every category has to be checked for containing a ^*part* argument in its left stack. On the other hand, it makes a huge set of lexical specifications redundant for every particle taking verbs in which the particle is just 'hopping' in the left argument list.

Since particles in Dutch seem to exhibit the *anywhere* property that (136) tries to account for (cf. Van Dreumel and Coppen 2003), we take this to be the canonical version of (131).

This completes the set of leftward merges. Again, the number is very restricted. One of the six, to wit (112), deals with a local inversion, the subtle auxiliary switch in the Dutch verb cluster. The other five represent merges in the core of Dutch syntax.

1.6.3 Dealing with adjuncts

In the exposition above, several modes were introduced aimed at dealing with adjunctive modification. The nature of this modification is automorphistic. An adjunct sends its argument to the same type the argument comes with. Consequently, adjunctive modification does not contribute essential ingredients to well-formedness or saturation. Moreover, adjunctive phrases are not very picky. They are usually capable of modifying different sorts of phrases, thus taking different sorts of categories as secondary category. In Dutch, they hardly impose conditions on the derivational status of their arguments. Finally, adjuncts themselves are not subject to discontinuous operations, peripheral to the sentential backbone as they are. They are rigid islands, with the exception of the extraction of so-called r-pronouns in adjunctive prepositional phrases.
In CLG, then, it is not very difficult to conceive of a category for an adjunct. For example, a lexical adjunct that can occur as the left modifier of a verb phrase – say: *snel* 'quick(ly)' – is categorizable as follows:

(137) `vp\u~[]/vp^adj`

The */^adj* mode is defined liberally enough in (109) to allow for all kinds of interference between this category and a constituent headed *vp*. Apart from (137), the word *snel* would also have to be assigned to categories taking sentences into sentences, nouns into nouns, and so on. This would yield a restricted class of assignments – recall that because of merge modes there is no need to come up with different categories for different arities of verbs, not even when immediate (left) adjunction to these verbs as heads of *vp* has to be enabled. The class would be unsatisfactory, nevertheless, since it is not very general and would have to be repeated all through the lexicon. As an alternative, adjunctive phrases may be assigned to a categorial classifier, to be instantiated at need in the combinatorial process. Such a classifier would

look like (137) but it would have variable typing. The variable is valuated in a closed finite class of types, *e.g.* {vp, s, n}.

(138) `X\u~[]/u~[X^adj]`

Subsequently, in a certain environment, a transfer rule is needed to give the relevant instantiation. As a matter of fact, since not all automorphic merges behave exactly alike, we could even parameterize the merge mode in assignment (138), having target type and merge mode specified in one rule, yielding the category

(139) `X\u~[]/u~[X^M]`

Derived merge rules will have this format:

(140) $\texttt{X\u~[]/u~[X^M]} \otimes_{M} \texttt{vp\Lf~Sla/Rf~Sra} \rightarrow \texttt{vp\Lf~Sla/Rf~Sra}$
only if
$\texttt{vp\u~[]/u~[vp^adj]} \otimes_{adj} \texttt{vp\Lf~Sla/Rf~Sra} \rightarrow \texttt{vp\Lf~Sla/Rf~Sra}$

(141) $\texttt{X\u~[]/u~[X^M]} \otimes_{M} \texttt{s\Lf~Sla/Rf~Sra} \rightarrow \texttt{s\Lf~Sla/Rf~Sra}$
only if
$\texttt{s\u~[]/u~[s^isl]} \otimes_{isl} \texttt{s\Lf~Sla/Rf~Sra} \rightarrow \texttt{s\Lf~Sla/Rf~Sra}$

It should be stressed that following such a track would not affect the outline of the grammar as given in section 1.4, but merely shorten lexical specifications. Equivalently, then, for every adjunct in the lexicon one can spell out at which types it applies and in what ways. This track is chosen in the present state of Delilah. The choice is mainly sensitive to the degree of sophistication with which the lexicon can be addressed.

As a matter of fact, the grammatical treatment of adjuncts puts a heavy burden on the grammar under any implementation. The more radical solution would be to treat adjuncts as optional arguments all the way down. This is essentially what Bouma and Van Noord (1994) suggested, and it is implicit in the cognitively inspired combinatoric model of Vosse and Kempen (2000). In our system, this proposal would amount to cancellation of arguments which would not affect the arity of the primary category: adjunctive arguments are optional as well as multiple. Here is one of the hypothetical rules for such a mode.

(142) $\textsc{Sec}\backslash\_\sim[\,]/\_\sim[\,] \otimes_{adj} \textsc{Prim}\backslash\textsc{Lf}\sim[\textsc{Sec}\texttt{\^{}}\textit{adj}|\textsc{Pla}\,]/\_\sim\textsc{Pra} \rightarrow \textsc{Prim}\backslash\textsc{Lf}\sim[\textsc{Sec}\texttt{\^{}}a\textit{dj}|\textsc{Pla}]/\textsc{Rf}\sim\textsc{Pra}$

Prim: head of *prim*ary category; *Plf: p*rimary category's *l*eft argument list *f*lag;
Sec: head of *sec*ondary category; *Sra: s*econdary's category's *r*ight *a*rgument list;
*Lf: l*eft flag (in resulting category); *Ra: r*ight argument list (of resulting category)

In this rule, the secondary category is the adjunct, to be eaten by a VP- or S-headed category, but without being cancelled from the left-argument list and without affecting the status of that list. The insularity of the adjunct is recognized in its empty argument lists.
There are clear grammatical advantages in treating adjuncts as optional arguments. In particular, all *wh*-types of operations on adverbial and adsentential adjuncts are easier to define if they are treated as arguments of higher functors in a (142)-type approach. Under the syntax given in section 1.6.2, it is simply not possible to handle a primary category as dislocated: all discontinuity is attributed to the way argument lists are merged. This merge cannot (re-) define or reconstruct the position of the primary category. Redefining adjuncts as secondary categories in the relevant merge would solve this problem.

The main problem for this redefinition, from a linguistic point of view, is that the argument approach to adjuncts seems more feasible for verbal functors than for nominal functors. Adjectives and relative clauses ad-modify nominal arguments, and the reversal of (142) would imply that nouns, too, become functors over optional modifiers. Such a strategy has been suggested by Janssen (1986) for relative clauses. The processing of nominal domains, however, would become more complex if nouns were optional functors.

Apart from the question whether it would be wise to have the adjunct at the top of the stack and not use the *anywhere*-proviso of section 1.6.2.3.6, (142) suffers from violating Count Invariance (47). That, however, would be very damaging to the (chart-) parsability of the system – see section 1.8. The only way to handle adjuncts as arguments, then, is to neglect them in deduction, introducing the following axiom:

(143) $\text{Adjunct}\backslash\_\sim[\,]/\_\sim[\,] \otimes_{\text{adj}} \text{Prim}\backslash\text{Lf}\sim\text{Pla}/\text{Rf}\sim\text{Pra} \rightarrow \text{Prim}\backslash\text{Lf}\sim\text{Pla}/\text{Rf}\sim\text{Pra}$

This introduces typological anarchy and global anywhere-ness. It violates the well-foundedness of merge (25)(b) and challenges the resource sensitivity of the grammar.
One way or another, reconstructing adjuncts as arguments involves trespassing on resource sensitivity, as can be read from the alternatives (142) and (143). Unfortunately, however, as will be made clear in section 1.8, placing adjuncts outside the frame of resource sensitivity endangers the system's computability. And computability is what we are after.
As a conclusion, then, the lexicon has to carry the load for syntax: multiple assignment of categories in a well-organized lexicon is easier to handle than

an increase of combinatorial power. We stick with adjuncts as primary categories, and pay the price of multiple typing.

This choice has one drawback that deserves special attention. Adnominal adjuncts can appear separate from their targets by the intervention of categories that eat their targets. The resulting structure is called *extraposition*. Here are two examples:

(144) Wie heeft de man zien werken met de hoed?
'Wie has the man seen work with the hat'
Who has seen the man with the hat work?
(145) Wie heeft de man zien werken die gisteren staakte?
'Who has the man seen work that yesterday striked'
Who has seen the man who was on strike yesterday work?

Under the present categorization, these sentences have to the following pattern (with simplified categories for clarity):

(146) ... np vp\np np\np

There is no way of bringing the adjunct category to its target before the target has been swallowed by a category – *vp* – that is intransparent to the adnominal adjunct. Now suppose that the *np* 'de man' would be categorized as looking for some adjunct to its right, and the adjuncts *met de hoed* and *die gisteren staakte* as primitive categories:

(147) ... np/adj vp\np adj

Clearly, then, there is an easy disharmonic composition of the two first categories (cf. (49)) bringing the negative and the positive versions of *adj* towards each other

(148) ... np/adj vp\np adj ⇒ ... vp/adj adj ⇒ ... vp

Treating adnominal adjuncts as arguments here would therefore have the major advantage of accounting for extraposition, and even of offering a theory of extraposition in terms of disharmonic composition. Following this track, however, every *np* would become ambiguous between looking for an adjunct and being saturated – a major complication in the derivation and increase of combinatorial power. The alternative of sticking with higher-order adnominal adjuncts calls for the ad hoc treatment of extraposition.

At this moment we have no decisive arguments for either strategy, and leave this question undecided.

1.7 THE GRAMMAR OF DISCONTINUITY AND COORDINATION

1.7.1 The source of discontinuity and connectedness

Natural language abounds in discontinuities: elements that depend on each other, do not necessarily come together, and are not necessarily adjacent. Grammar is the study of the nets rising from these discontinuous interdependencies. Yet, not everything goes. For a structure like

(149) A B A' B'

no grammarian of any language would offer the analysis

(150) ((A A') (B B')).

It is a heartfelt desire of grammarians to present analyses that are – in a very precise way – *connected*:

(151) *Connectedness*
For any sequence (A B A') the structure of the sequence is either (A (B A')) or ((A B) A').
There is an asymmetric relation $\mathfrak{R}$ such that (A B A') is a constituent only if either $\mathfrak{R}$(A, $\mathfrak{R}$ (B, A')) or $\mathfrak{R}$($\mathfrak{R}$ (A, B,) A')).

In Steedman (1990) this is presented as the invariance *Adjacency* for categorial combinatorics: you can combine categories only if they are adjacent.
In axiomatic calculi of categorial grammar without permutation, there are no values for types *y* and *w* such that the sequence

(152) a b y\a w\b → x

can be deduced for a product-free type *x*. The sequence is more discontinuous and less connected than these calculi can afford.

This does not mean that discontinuity and connectedness in natural language are unrelated. Our present framework, with categories that merge and have to express some linearization at the same time, discontinuity and connectedness paradoxically arise from the same source: the internal complexity of categories. It is important to see that this internal structure not only reflects linearization but also and equivalently induces semantic structure, generalizing dependency.

1.7.2 Discontinuity in CLG

In CLG, phrases are related to categories that express their selectional condition. If a phrase occurs discontinuously, *i.e.* separated from its selecting phrase, the nature of that discontinuity as well as the nature of the interveners must be expressed by the interaction of two other categories: the category of the selecting phrase and the category of the intervening phrase. The nature of the interaction of the two categories is given by the cancellation modality on the argument that complies with the head of the category of the selecting phrase. Here is an example with a right occurring selecting phrase and a left adjuncting intervener. *I* is the intervening phrase, *SD* the selected phrase, to be separated from *SG*, the selecting phrase. Cat_x stands for a category of phrase *X*. For the ease of the exposition, let us assume that *I* is the only intervening phrase.

(153) SD I SG
Cat_{SD} Cat_{I} Cat_{SG}

$Cat_{SD} = H_{SD}\backslash LF_{SD}{\sim}SLA_{D}/RF_{SD}{\sim}SRA_{D}$
$Cat_{I} = H_{I}\backslash LF_{I}{\sim}L_{I}/RF_{I}{\sim}[H_{SG}\text{^}MSG|R_{I}]$
$Cat_{SG} = H_{SG}\backslash LF_{SG}{\sim}[H_{SD}\text{^}MSD|SLA_{G}]/RF_{SG}{\sim}SRA_{G}$

(154) $Cat_{I} \otimes_{i} Cat_{SG} \rightarrow Cat_{I+SG}$
$H_{I}\backslash LF_{I}{\sim}L_{I}/RF_{I}{\sim}[H_{SG}\text{^}MSG|R_{I}] \otimes_{MSG} H_{SG}\backslash LF_{SG}{\sim}[H_{SD}\text{^}MSD|SLA_{G}]/RF_{SG}{\sim}SRA_{G} \rightarrow$
$H_{I}\backslash LF_{I+SG}{\sim}([H_{SD}\text{^}MSD|SLA_{G}]\oplus_{MSG}L_{I})/RF_{I+SG}{\sim}(R_{I}\oplus SRA_{G})$

(155) $Cat_{SD} \otimes_{j} Cat_{I+SG} \rightarrow Cat_{(I+SG)+SD}$
$H_{SD}\backslash LF_{SD}{\sim}SLA_{D}/RF_{SD}{\sim}SRA_{D} \otimes_{MSD}$
$H_{I}\backslash LF_{I+SG}{\sim}([H_{SD}\text{^}MSD|SLA_{G}]\oplus_{MSG}L_{I})/RF_{I+SG}{\sim}(R_{I}\oplus SRA_{G}) \rightarrow$
$H_{I}\backslash LF_{I+SG+SD}{\sim}((SLA_{G}\oplus_{MSG}L_{I})\oplus_{MSD}SLA_{D})/RF_{I+SG+SD}{\sim}((R_{I}\oplus SRA_{G})\oplus_{MSD}SRA_{D})$

H_X: head of Cat_X; *Mx*: cancellation mode for H_X; *PLF: p*rimary category's *l*eft argument list *f*lag; *SRA: s*econdary's category's *r*ight *a*rgument list; *LF: l*eft flag; *RA: r*ight argument list; R_X: *r*ight argument list introduced by category X

Note that the intervener types the merged phrase *SD+I+SG*. It is the primary category in the first merge, (155) cancelling the head of the selecting phrase, and determines the primary category in the second merge, cancelling the head of the selected phrase. Abstracting from direction, this is the only relationship between the phrases *SG*, *I* and *SD* that can be reconstructed by CLG and complies with the definition of connectedness (151). Connectedness is realized in CLG as generalized composition, taking over 'agendas' under merge. Because connectedness is interpreted here as composition, the category Cat_I, besides intervening, is also bridging between *SD* and *SG* and their respective categories. While intervening, it brings the two occurrences of H_{SD} – the negative and the positive one – together in adjacent positions, after all.

In CLG, then, discontinuity is accounted for inasmuch as bridges exist or can be constructed. Connectedness has a clear-cut categorial condition, in the form of a predictable construal, within the limits of finite combinatorics, of bridging interveners. The construal of discontinuity is dictated by the specification of the merge modes involved in the bridging, M_{SD} and M_{SG} in the example above. In particular, they may maximize or minimize the effects of discontinuity by the way they append argument lists and by zero requirements on argument lists of the categories involved. Clearly, the merge under mode M_{SG} may introduce additional discontinuity if L_I, SLA_G or R_I is not empty: this merge will mark the phrase associated with the primary category, or the phrase associated with the secondary category, or both as an intervener with respect to the arguments in these lists.

This construal of discontinuity in CLG is will be scrutinized in the next sections.

1.7.2.1 Managing discontinuity

To get a clear idea of what discontinuity under CLG is up to, consider two phrases *P* and *S* – for primary and secondary string, respectively – and their respective categories PRIM\PLF~PLA/PRF~[SEC^j| PRA] and SEC\SLF~SLA/SRF~SRA. Let *PLA*, *PRA*, *SLA* and *SRA* be sequences of strings with types that can be cancelled against the corresponding arguments in the categories of *P* and *S*. *PRIM* and *SEC* stand for the smallest phrases with the corresponding head categories, *i.e.* for the phrases introducing the primary and the secondary head types, respectively. Depending on the specification of the mode *j*, any of the following strings may model this family of merges. PRIM and phrases selected by it (*PLA* and *PRA*) are italicized for transparency. For each sequence, a number indicates how many of the four argument strings are separated from their head string. The dislocated argument strings are underlined.

(156) SLA *PLA PRIM* SEC SRA *PRA* :: 1
(157) *PLA* SLA *PRIM* SEC SRA *PRA* :: 1
(158) SLA *PLA PRIM* SEC *PRA* SRA :: 2
(159) *PLA* SLA *PRIM* SEC *PRA* SRA :: 3

Several aspects of this exercise are noteworthy. First, none of the four strings is fully continuous. As a consequence, every merge with four non-empty argument lists induces some form of discontinuity.
Secondly, the number of discontinuous pairs varies with the ways of appending (non-empty) argument lists.
Thirdly, all discontinuities respect connectedness, in the sense that there is a bracketing available which complies with (151). For example, (159) may be parsed as follows:

(160) (*PLA* ((SLA (*PRIM* SEC)) *PRA*) SRA)

This follows from the fact that absolute discontinuity of degree 4 is impossible: in the active direction – rightward in the examples – the primary argument string *PRA* cannot be dislocated, since it is bound to be more peripheral than the string introduced by the cancelled secondary head, its former companion. The degree of discontinuity, then, can be controlled by the grammar in either of two ways. One way is requiring one or both of the relevant input argument lists to be empty. The other way is appending the lists in such a way that arguments cannot intervene between other arguments and its selecting phrase. The latter way induces crossing dependencies. In that case, the primary category – the intervener – imposes as little discontinuity on the secondary category's combinatorics as possible.

Here are some instances of merge mode definitions to this effect, guided by the examples above. Only the elements relevant to the strategy are specified.

(161) PRIM_~[ ]/_~[SEC^j|_] $\otimes_j$ SEC_~SLA/_~_ → PRIM_~SLA/_~_
(162) PRIM_~PLA/_~[SEC^j|_] $\otimes_j$ SEC_~[ ]/_~_ → PRIM_~PLA/_~_
(163) PRIM_~PLA/_~[SEC^j|_] $\otimes_j$ SEC_~SLA/_~_ → PRIM_~SLA+PLA/_~_

The two ways of reducing discontinuity under the presupposition of connectedness are not independent of each other. Choosing the empty list strategy makes the append strategy redundant. There are two ways to avoid discontinuity by empty lists. If the secondary argument list is doomed to be empty, as in (162), that category is sealed to cover a (partial) island. If the primary argument list is marked empty, as in (161), this can only mean that the other

arguments – which are required to have been cancelled already – are more intrinsic to the primary phrase than the secondary argument.
In this vein, crossing dependencies as in (157) and (159) are the inevitable result of a close connection between primary and secondary category on one hand and linearization and antisymmetry on the other. In other words, crossing dependency is a normal configuration, induced by a sound strategy of appending non-empty passive argument lists in such a way that connectedness is assured.

The following table contains an overview of the discontinuous effects of every possible specification of argument list. The condition in that table means that the specification has the effect mentioned only if that condition is fulfilled. Otherwise, it is without any effect on linearization.

(164) *Table of merge effects for rightward cancelling*

specification	*combined with*	*linearization/bracketing*	*linearization effects*
PLA := [ ]		SLA (PRIM SEC .).	discontinuity
SLA := [ ]		PLA (PRIM SEC.).	continuity
PRA := [ ]		.(.PRIM SEC) SRA	continuity
SRA := [ ]		.(.PRIM Sec) PRA	continuity
SLA+PLA	PLA, SLA ≠ [ ]	PLA (SLA (PRIM SEC .).	crossing disc
PLA+SLA	PLA, SLA ≠ [ ]	SLA (PLA (PRIM SEC .).	discontinuity
PRA+SRA	PRA, SRA ≠ [ ]	.(. PRIM SEC) PRA) SRA	discontinuity
SRA+PRA	PRA, SRA ≠ [ ]	.(. PRIM SEC) SRA) PRA	continuity
PLF := u	SLA+PLA, SLA ≠ [ ]	PLA (SLA (PRIM SEC.).	crossing disc
PLF := a	SLA+PLA, SLA ≠ [ ]	PLA" (SLA ((PLA' PRIM) SEC.).	incoherent disc
SLF := u	SLA+PLA, SLA ≠ [ ]	PLA (SLA (PRIM SEC .).	crossing disc
SLF := a	PLA+SLA, PLA ≠ [ ]	SLA" PLA (PRIM (SLA' SEC .).	incoherent disc
SRF := u	PRA+SRA, SRA, PRA ≠ [ ]	.(. PRIM SEC) PRA) SRA	crossing disc
SRF := a	PRA+SRA, SRA, PRA ≠ [ ]	.(. PRIM (SEC SRA')) PRA SRA"	incoherent disc

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

The above scheme suggests a hierarchy in specifications of merge modes affecting linearization. The strongest effect results from emptiness conditions on input argument lists. Any such specification guarantees continuity or connected discontinuity at its side, without further specifications being necessary. Next are the ways of appending. Their effect is conditioned by

non-emptiness of any of the argument lists involved. Their effect, therefore, is dependent. Finally, the effect of affectedness specification subsists on the choice for particular appends, which were already dependent themselves. Thus, flags are the subtlest instruments for influencing linearization. Appends bring in discontinuity; flags regulate the nature of the discontinuity. Recall that flags are sensitive to derivational history in ways specified for Dutch in (74): a flag *u* encodes that the list has not hitherto been activated, or that it was not involved in the last few merges. As a rule, the conditions on argument lists model the phenomena as such, whereas the conditions on flagging correlate with conditions on phenomena.

The hierarchy renders some combinations redundant or nonsensical, or just marked. For example, it is not easy to see what any appending or flagging might add to the empty input specification. In the same vein, flagging specifications have effect only when append is discontinuous. Moreover, it may seem that there may be more options to the same effect. Compare for example the following merge modes.

(165) PRIM_~PLA/_~[SEC^*i* |PRA] $\otimes_i$ SEC_~SLA/u~SRA $\rightarrow$
PRIM_~PLA $\oplus_i$ SLA/_~SRA+PRA

(166) PRIM_~PLA/_~[SEC^*j* |PRA] $\otimes_j$ SEC_~SLA/a~[ ] $\rightarrow$
PRIM_~PLA $\oplus_j$ SLA/_~PRA

(167) PRIM_~PLA/_~[SEC^*k* |PRA] $\otimes_k$ SEC_~SLA/_~[ ] $\rightarrow$
PRIM_~PLA $\oplus_k$ SLA/_~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

At first glance, the linearization effects may appear similar: in each case, the relevant order of phrases will be . . *PRIM* SEC SRA *PRA*. The bracketings differ, however: ..(..*PRIM* SEC) SRA) *PRA* for (165), ..(..*PRIM* (SEC SRA)) *PRA* for the two other cases. Only the first specification ensures the merge of the primary category with a lexical secondary category. (166) excludes a lexical status for the secondary category, whereas (167) is indifferent in this respect. The options thus vary in their derivational consequences.

The line of reasoning followed above leaves no room for discriminating between processes handling continuous structure and processes dealing with discontinuity. All structure and all interpretation result from merging complex categories, and the mere merge of complex categories provokes forms and degrees of discontinuity. The opposition between *move*-type operations inducing discontinuity and *merge*-type operations inducing continu-

ous structures in generative grammar is counter-productive in CLG. If merge deals with linearization, move is redundant (Cremers 2004).

1.7.2.2 Discontinuity and minimalism

Generative grammar deserves credit for making explicit that discontinuities format sentence structure. In the minimalist emanation of the generative view, the distinction between *merge* and *move* (or *copy+merge*) highlights discontinuity as one of language's omnipresent properties. Moreover, the formulation of *move* as a one-structure-operation, handling features in one local tree, illustrates that discontinuity is neither unbounded nor random, but inherently determined by properties of the structure. The general idea is that *merge* builds structure, and *move* changes it, subsequently. This amounts to restyling the division of labour between context-free phrase structure rules and transformations, as established in earlier stages of generativism.
It is noteworthy that subsuming discontinuity under *move* does not mean that discontinuity is defined independently. As a matter of fact, there is circularity. Discontinuity is marked by applying *move*, and if something moves it is discontinuous.

Categorial grammar combinatorics does not favour a principal distinction between operations of *move* and operations of *merge*. In the first categorial approaches to *wh*-movement, for example, forms of type-raising were proposed for handling the interpretation of leftward dislocated elements. Later (*e.g.*, Moortgat 1988), special binary extract and insert operators were introduced to cover these configurations. They were subject, partially, to the same logic that governed the operators taking care of continuous sequences. From the point of view of linearization, however, these operators were of the 'anywhere' kind: *AB* was interpreted as the set of those split strings *ww'* such that *wbw'* is in *A* if *b* is in *B*, but the category gave no information as to where the string was to be invaded. In recent extensions of multimodal categorial grammar, as outlined in Moortgat (1997), the toolkit has been extended to such a degree that specialized indexed operators, exclusive to certain structural configurations, can handle discontinuous or permutative instances of linearization. Discontinuity is handled by specialized structural operations on marked structures. With these wide coverage formalisms, it is possible to translate the minimalist distinctions between *move* and *merge* into categorial terms, as was shown by Vermaat (1999).

Still, it has not been decided that *merge* and *move*, *i.e.* continuous and discontinuous linearizations, really are different processes needing distinct treatments

in a grammar. Scrutinizing the definitions of *merge* and *move* as presented in Stabler (1992), and putting technical details aside, we are led to assume the following distinction between the two operations. *Merge* deals with two trees where one tree occupies one position in the other tree. *Move* deals with two trees where one tree is related to two positions in the other. *Move* projects one tree twice, *merge* projects two trees once. Distinguishing these two processes is mandatory only to the extent that not every subtree of a tree is doubly or multiply projected in the end. That is, if every tree that is 'placed' by *merge* were to be doubly committed by *move*, one might as well come up with one single operation *mergemove* that performs the structure building and the position linking in one hit. In that case, discontinuous occurrence would be the fate of every phrase, in the exact sense that every phrase except the one associated with the top node links two or more not necessarily adjacent positions in the structure. Moreover, as *move* would be bound to *merge*, all double linking would be specified locally. It would amount to the conception that the occurrence of every phrase in a sentence is backed by a chain: to be a constituent is to be the head of a chain. Of course, this is not really far removed from having every configuration licensed by feature checking, *i.e.* by matching pairs of features which are introduced at different positions. Here is an attempt to formulate such a unified structure building operation *mergemove*.

Let every lexical element come with a three-leaf labeled stucture $[_X$ Spec [X Compl]] such that Spec and Comp have the same type. For example: an intransitive verb like *walk* would be assigned $[_{vp}$ NP [walk NP]]. Spec and Comp are supposed to be chained, having the same relevant features except for lexical content. At each lexical structure, the position in which the argument is lexicalized – either Spec or Comp – is marked; we will use italics to do so. The head is lexical *per se*. So, the category of *walk* might be $[_{vp}$ *NP* [walk NP]]. The structure $[_X$ Y [H Y]], having no lexicalizations at all, is supposed to be equivalent to just H. *Mergemove* is defined on every pair of trees $[_X$ *Y* [.X .Y]] or $[_X$ Y [.X .*Y*]] and $[_Y$ Z1 [. Y. Z2]], where either Z1 or Z2 is the lexical position and the dots mark possibly intervening right branching structures. It is assumed, however, that the head of every structure is uniquely defined as the highest position equal to the top label. This has the same effect as the directional head marking in Stabler's (1992) formalization of minimalistic combinatorics. This yields the following definition of *mergemove*:

(168) *Mergemove*

$[_X\ Y\ [.X\ .Y]] + [_Y Z1\ [.\ Y.\ Z2\ ]] \Rightarrow [_X Z1\ [_X\ Y\ [.\ X\ .[_Y.Y.\ Z2]]$

$[_X Y\ [.X\ .Y]] + [_Y Z1\ [.\ Y.\ Z2\ ]] \Rightarrow [_X Z1\ [_X Y\ [.\ X\ .[_Y.Y.\ Z2]]$

At *merge*, the lexical content of the head of the selected structure is placed in the position marked in the selecting structure. To see how things would turn out, consider the following mini-grammar of a language over the alphabet {1,2,3,4}. The lexicon would be the following set of structures:

(169) { [$_1$ *2* [1 2]], [$_1$ 2 [1 *2*]], [$_2$ *3* [2 3]], [$_2$ 3 [2 *3*]], [$_3$ *4* [3 4]], [$_3$ 4 [3 *4*]], [$_4$ e [4 e]] }

Clearly, 1 is supposed to select 2, 2 to select 3, and 3 to select 4. Below is the derivation of the number 4321.

(170) [$_1$ *2* [1 2]] + [$_2$ 3 [2 *3*]] ⇒ [$_1$ 3 [$_1$ *2* [1 [$_2$ 2 *3*]]]]
[$_1$ 3 [$_1$ *2* [1 [$_2$ 2 *3*]]] + [$_3$ *4* [3 4]] ⇒
[$_1$ *4* [$_1$ 3 [$_1$ *2* [1 [$_2$ 2 [$_3$ *3* 4]]]]]
[$_1$ *4* [$_1$ 3 [$_1$ *2* [1 [$_2$ 2 [$_3$ *3* 4]]] + [$_4$ e [4 e]] ⇒
[e[$_1$ *4* [$_1$ 3 [$_1$ *2* [1 [$_2$ 2 [$_3$ *3* [$_4$ 4 e]]]]]]] (= [$_1$ *4* [$_1$ 3 [$_1$ *2* [1 2 *3* 4]]]])

The linearization of the italicized elements and the head position gives the desired result.
The choice of categories is far from trivial. Under the present lexicon, not every order of the digits is derivable, while all chains are nested around 1. In particular, no digit word can be derived in which 1 and 2 are not adjacent. This is due to the selectional restrictions implemented in the categories, *i.c.* the choice to have 2 selected by the element that invariably projects the top node. To exclude some more orders, one can limit the lexicon. For example, by taking the category [$_1$ *2* [1 2]] out, one excludes the derivation of any number representation in which 2 occurs to the left of 1, which is half of the original language. The extension and the nature of the language are thus completely localized in the lexical specification. The above implementation proves that one can consistently and non-trivially consider *merge* and *move* to be a single operation.

1.7.3 Patterns of Dutch

This section contains an overview of the specifications used in the concise syntax of Dutch of section 1.6.2. Instead of naming lists by direction, the table generalizes over directionality and names list (primary or secondary) passive or active, active being the direction in which the cancellation takes place. Only if some explicit value is required by the merge mode is this requirement specified in the relevant column. Even if modes seem to have the same specifications, they may differ as to exceptional output flagging, which is not accounted for in this table. Recall, furthermore, that appending of left lists

always implies down-stacking of a *wh*-argument in either list, according to provision (95).

(171) *Table of merge specifications for* Delilah*'s grammar of Dutch*

Prim\Ppf~Pp/_~[Sec^*I* |Pa] $\otimes_{I}$ Sec\Spf~Sp/Saf~Sa → Prim_~appendP/_~appendA
Sec\Saf~Sa/Spf~Sp $\otimes_{J}$ Prim_~[Sec^*J* |Pa]/Ppf~Pp → Prim_~appendA/_~appendP

mode	*Pp*	*Sa*	*Sp*	*Ppf*	*Saf*	*Spf*	*appendP*	*appendA*
/^*isl*		[ ]	[ ]					
/^*transp*		[ ]		u		u	Sp+Pp	
/^*open*		[ ]		u			Sp+Pp	
/^*penins*		[ ]	[ ]/[x^*wh*]				Sp+Pp	
/^*transpipp*		[ ]		u	a	u	Sp+Pp	
/^*sentop*		[ ]	[ ]	u		w/u		
/^*adj*							Pp+Sp	Sa+Pa
\^*isl*		[ ]	[ ]					
\^*transp*	-/[ ]			u	u	u	Pp+Sp	Sa+Pa
\^*open*	-/[ ]						Sp+Pp	Sa+Pp
\^*wh*		[ ]	[ ]					
\^*adj*							Pp+Sp	Sa+Pa
\^*part*		[ ]	[ ]	u	u	u		

Prim: head of *prim*ary category; *Ppf, Paf: p*rimary category's *p*assive / *a*ctive argument list *f*lag;
Sec: head of *sec*ondary category; *Sa, Sp: s*econdary's category's *a*ctive / *p*assive *a*rgument list;
*Sra: s*econdary's category's *r*ight *a*rgument list

From this table, one can read some aspects of the merges needed for this fragment of Dutch. Before going into details, however, it should be stressed that other grammars of the same fragment may be equally efficient or more enlightening. One might even compare grammars with respect to the instruments they choose for determining the family of merge modes doing the job. Below, we will discuss the use of the discontinuity toolkit in Delilah's grammar.

What immediately strikes the eye is that almost all input specifications of empty lists concern the secondary category. The only cases of such specification for a primary list represent an alternative formulation for a rule without such specification. These alternatives are driven by reflection, not by necessity, though. Thus, in this grammar the main load of discontinuity management is put on the secondary category. The primary category is often required to be unaffected in the passive direction and to allow the secondary passive agenda to be the first to be met, thus reducing the chance of incoherent discontinuity: the secondary arguments are close to the secondary head. It is, furthermore,

remarkable that in particular SA – the edge of the secondary category necessarily separated from the primary category – tends to be completed before the merge occurs.

None of the merges involves a requirement on non-emptiness of input argument lists. The possible specification that an input list be affected is used only once, in the mode */^transpipp*. This mode exclusively deals with *infinitivus-pro-participio* effects, which occur only in the presence of complex verb clusters. All other affectedness specifications require an input list to be unaffected.

As for the modes of append, it is noteworthy that the passive direction always complies with the side the primary category is on. Thus, from a directional point of view, the active appendings in the leftward cancellations all reflect the priority of the secondary left agenda, *i.e.* the imposition of continuity. Accordingly, the passive appendings in the rightward cancellations express the same priority, now invoking coherent discontinuity. The only exception here concerns a secondary left list that only contains a *wh*-element, this element being the only one to escape from the secondary domain.

As a whole, the grammar of Dutch according to this specification exists on only a few instruments of discontinuity management. The variety of means is rather limited.

(172) *Table of discontinuities in Dutch according to section 1.6.2*

mode	*PP*	*SA*	*SP*	*append mode*	*5*	*3*	*1*	*2*	*4*	*6*
/^isl		[ ]	[ ]			*PLA*	*PRIM*	SEC	*PRA*	
/^transp		[ ]		SP+PP	*PLA*	SLA	*PRIM*	SEC	*PRA*	
/^open		[ ]		SP+PP	*PLA*	SLA	*PRIM*	SEC	*PRA*	
/^penins		[ ]	[ ]			*PLA*	*PRIM*	SEC	*PRA*	
/^transpipp		[ ]		SP+PP	*PLA*	SLA	*PRIM*	SEC	*PRA*	
/^sentop		[ ]	[ ]			*PLA*	*PRIM*	SEC	*PRA*	
/^adj				PP+SP; SA+PA	SLA	*PLA*	*PRIM*	SEC	SRA	*PRA*
\^isl		[ ]	[ ]			*PLA*	SEC	*PRIM*	*PRA*	
\^transp	-/[ ]			PP+SP; SA+PA	*PLA*	SLA	SEC	*PRIM*	*PRA*	SRA
\^open	-/[ ]			SP+PP; SA+PA	*PLA*	SLA	SEC	*PRIM*	SRA	*PRA*
\^wh		[ ]	[ ]			*PLA*	SEC	*PRIM*	*PRA*	
\^adj				PP+SP; SA+PA	PLA	*SLA*	SEC	*PRIM*	SRA	*PRA*
\^part		[ ]	[ ]			*PLA*	SEC	*PRIM*	*PRA*	

PRIM: head of *prim*ary category; *PPF, PAF: p*rimary category's *p*assive / *a*ctive argument list *f*lag;
SEC: head of *sec*ondary category; *SA, SP: s*econdary's category's *a*ctive / passive *a*rgument list;
*SRA, SLA: s*econdary's category's *r*ight/*l*eft *a*rgument list; 1-6: peripherality indication

Table (172) shows the resulting discontinuities for the fragment. Here, the reference to active and passive direction is directionalized again as left and right lists and flags. The string positions are numbered from inside out, to indicate peripherality with respect to the kernel of primary and secondary head. Continuous structures are shaded. Italics indicate dependencies determined by the primary category.

From this table one can see that the grammar of Dutch – though infamous for its discontinuous verbal clustering – avoids many possible sources of discontinuity. With the notorious exception of the rules for left and right adjunction, all rules keep or may keep argument lists empty. Some rules do not introduce discontinuity at all – those requiring both secondary lists to be empty. Recall, furthermore, that in the active direction the primary arguments cannot occur discontinuously. As a result, discontinuity arises in one direction at most. Although we cannot deduce this result, it can hardly be accidental. Therefore, we present it as a conjecture on CLG-type grammars for natural language.

(173) *conjecture*
In a CLG grammar of a natural language, no rule imposes discontinuity in two directions.

Basically, the conjecture reduces the relevance of the grammatical formalism for 'real applications' to a relevant fragment of the formalism. Interestingly, this seems to be on a par with statements like the following:

(174) Although the set of terms of the full typed lambda calculus is *context-free*, the set of lambda terms characterizing the Lambek fragment of type logic (Lambek categorial grammar) is a *non-context-free subset* (Van Benthem 1991:109).
(175) Combinatory Categorial Grammar is mildly context-sensitive (Joshi *et al.* 1991).

The first statement refers to the fact that not every lambda term gives a recipe for a deduction in Lambek categorial grammar (see sections 1.4. and 1.8). Not every lambda term is a Lambek proof as well. The second statement says that a certain formalism transgresses the boundaries of context-freeness without falling into the full expressivity of context sensitivity (see also section 1.8). This is argued by Cremers (1999) to hold for CLG too. In both cases, powerful machinery is used only selectively when applied to natural-language grammar. Moreover, section 1.8.6 reports on Van de Woestijne's (1999) observation that Delilah's CLG does not seem to meet its *worst case complexity* when parsed; we now conjecture that this is because of (173).

1.7.4 Coordination as discontinuity

Coordinated phrases are inherently discontinuous. In the presence of coordinators, at least one phrase is separated from its selector or its selection by other phrases, among which is the non-connected coordinator. Coordinators have no special licensing relation with any phrase or category. Thus, they intrude on the combinatoric process. This is the case both for constituent and non-constituent coordination. Here are some examples; the peripheral edge of each of the conjuncts is indicated by a dot.

(176) Ik heb .haar echtgenoot. en .diens moeder. ontmoet
I have her husband and his mother met
'I met her husband and his mother'

(177) Ik heb .haar echtgenoot de fiets. en .diens moeder de auto. gegeven
I have her husband the bike and his mother the car given
'I gave the bike to her husband and the car to his mother'

(178) … dat ik .de jongens wil. en .de meisjes moet. begeleiden
… that I the boys want and the girls should supervise
… 'that I want to supervise the boys and should supervise the girls'

In the sentences above, the non-coordinated selector or selection that is related to both conjoined phrases is underlined. Each underlined phrase is separated from at least one of its relatives by at least the coordinator itself. Unless the coordinator itself is selected, this state of affairs marks non-connected discontinuity.

In the preceding section, we argued that discontinuity is a normal attribute of sentences in our grammatical analysis. Nevertheless, there are several good reasons to treat coordination as a structure of a nature different from other discontinuities. These reasons will be discussed below. They amount to the thesis that the type of discontinuity established by coordination is unconnected by necessity, in the sense of (151).
Under the present assumptions, our grammar is not suited to deal with this type of discontinuity. Therefore, DELILAH handles coordination by a distinct algorithm that is fed by the grammar and feeds onto it, but is steered by essentially different procedures and aims. The algorithm is introduced and accounted for in Cremers (1993a) and Cremers and Hijzelendoorn (1996), and will be discussed separately in section 1.8.8 as part of the DELILAH automaton. It sets coordination apart from the computation of other linguistic structures because the mechanics of these computations do not fit the nature and the scope of coordination.

The arguments for a dedicated treatment of coordinated structures are given below.

1.7.4.1 Conjunctions do not select

There are only a few, mostly focus-related restrictions on the occurrence of conjunctions and conjuncts in a sentence. The general rule is that if the conjuncts can be properly contrasted with each other at the level of information structure and are syntactically compatible, the conjunction is well-formed and interpretable. Well-formedness, then, depends on the information contour rather than on the structural properties of the sentence. A recipe for forming coordinated structures could be this. Take any well-formed sentence without coordination. Choose any position to the right of some phrase. Take some string to the left of that position. Construct another string that is contrastive to the selected one, and insert the conjunction with the constructed string in the chosen position. The result will generally be a well-formed coordinated structure.

The main challenge to this recipe is the coordination of singular filters generated by non-empty atoms (singular referential DPs) in subject position; their conjunction may violate agreement under this procedure. Elsewhere we argue that this phenomenon is neither syntactically nor semantically persuasive (Cremers 1993a: ch. 2; Cremers 2002).

There is no reason to assume that conjunctions are biased towards certain types of phrases or categories. In particular, conjunctions neither select nor license either of the conjuncts. Consider the following phrase.

(179) Geen enkele ambtenaar durfde mij een notitie over de begroting en ...
No civil servant dared me a memo on the budget and ...
'No civil servant dared ... (to) me a memo on the budget and ...'

Even when the focus structures of these phrases are given, no grammarian can tell by inspecting only this left environment of a coordinator which part of the phrase will turn out to be conjoined. As a matter of fact, every dot in the following representation of (179) may turn out to be the left edge of the conjunct, but only one – in rare cases of ambiguity: two – will be. The selection of this edge is not determined by the structure of the phrase.

(180) .Geen enkele ambtenaar. durfde .mij .een .notitie .over .de .begroting en ...

Being a conjunct, then, is not an independent configurational property of a phrase but the outcome of a process (Cremers 1993b, Harbusch and Kempen

2006, but see Houtman 1994 for a defense of the opposite view). *Conjunct* is neither a category nor a type. It is not even a function. We take the argument embodied by (180) to weigh heavily against any categorial characterization of conjunctions in languages like Dutch.

The argument is directed against proposals to consider conjunctions as types with essential variables or as multi-typed operators in categorial grammar, as put forward by Wittenburg (1986), Moortgat (1986), Steedman (1990), Houtman (1994) and others. The argument also opposes analyses of conjunctions as phrasal heads, as suggested by Johannesen (1997) and Munn (1993). It even runs counter to the 'classic' conjunction-reduction approaches, since grammatical processes of the transformational type target specified categories and/or constituents. But generative grammar seems to have left coordinative structures as a discrete object of study: none of the 77 chapters of the Companion to Syntax (Everaert and Van Riemsdijk 2006) focuses on them.

1.7.4.2 Conjunctions are not selected

A good argument for conjuncts playing a role in the semantics of certain maximality expressions was made by Postma (1995). He observes that in Dutch phrases like *met man en muis vergaan* ('with man and mouse sink'), *have en goed verliezen* ('stock and barrel loose') entail universality or totality, and that it must be the semantics of conjunction that brings about this effect. In a sense, the conjunction is selected by the verb as an expression of degree. Hoeksema (1988) and Poß (2010) put forward other lexicalised forms of coordination.

These expressions, however, are all frozen, and meaningful only to the extent that they are frozen. Notwithstanding Postma's observation that the conjunction may introduce maximality, it makes no sense to say that the conjunction is selected by the verb as a conjunction. It is selected as an expression of degree. Therefore, no phrase selects a conjunction. Conjunctions do not occur on the demand side of the grammar. Note that conjunctions and adjuncts differ in this respect. It is relatively unproblematic to reverse the normal automorphic analysis of adjuncts into an argument analysis in which adjuncts are selected, even with certain limitations, by other phrases (see section 1.6.3). Analyses along these lines have been put forward by, *e.g.* Bouma en Van Noord (1994) and Janssen (1986). Such a move is not conceivable for conjunctions. Conjunctions are that unspecific that selecting them as arguments would simply double (or more than double) the set of categorial specifications, according to the formula: if category X is selected then also *X and X* and *X or X*. No analytical gain would result from doing so, however.

1.7.4.3 Coordinations do not obey Count Invariance

Count invariance (47) – repeated below – is a major distinctive feature of resource- sensitive type logics and categorial grammars.

(181) *Count Invariance (repeated)*
For each axiom $\phi \rightarrow \psi$ in (44) and for every type t, t-count(φ) = t-count(ψ).

It identifies the combinatoric engine of these systems as establishing a relationship (*e.g.* cancellation) between two items fulfilling opposite roles in the structure. By definition, coordination entails the *multiplication* of certain roles in a configuration, *i.e*. it spreads identity rather than opposition. In the presence of coordination, the balance between selectors and selectees breaks down, not by accident, but by neccessity. Only by considering conjunctions to be unselective automorphic selectors can one overcome this imbalance. This is effected by assigning them to type $(x\backslash x)/x$, where the two negative occurrences of x restore the balance against the two positively occurring conjuncts which they aim to cover. Another strategy advocated by Dalrymple *et al.* (1995) is to weaken the resource sensitivity of their (linear) logic. In either approach, however, count invariance (a reflection of resource sensitivity) loses its edge: we cannot tell which is the balance to be restored before having parsed the sentence in order to identify the conjuncts that bring in the type doubling. And we must parse because the conjuncts themselves are not identified locally. This paradox of coordination is addressed in Cremers (1993a) and Cremers and Hijzelendoorn (1997a). In the latter it is argued that the best counts we can get in the presence of coordination are (complex) inequalities, whereas count invariance is expressed in terms of equalities.

1.7.4.4 Coordinations do not mean

In this section, we will argue that a conjunction *i.e.* the unit formed by the two coordinates and the coordinator, in general cannot be interpreted prior to the interpretation of the sentence or sentences which the coordination is part of. First, recall the statement from preceding sections that the identification of the coordination is the result of parsing, not a sub-process of it, like the identification of any other constituent. This alone already implies that the meaning of the conjunction cannot contribute independently to the meaning of the sentence that is construed from parsing it. But even if we abstract from this contradiction, and assume that coordinations contribute to the meaning construal by oracle, it is hard to determine what that contribution could be. Suppose we were, by oracle, capable of identifying any conjunction as some constituent of a certain

category with structure $[_{cat}$ *cat conj cat]*. In order to establish a meaning for it, we must be able to identify an operation executed by the conjunct on the meanings of the conjuncts, yielding one 'merged' meaning. It is tempting to interpret *conj* as a certain operation defined on the algebra D_{cat} of denotations of category *cat.* It is well-known, though, that conjunctions do not show boolean behaviour, in the sense that they denote a fixed algebraic operation on their arguments' algebra. The standard example is with quantifiers, *i.e.* denotations of type *np*. In sentence (182), the coordination is most likely interpreted as the meet over the two quantifiers it conjoins: what is predicated of the coordination is predicated of each coordinate. In (183), on the other hand, nothing is predicated of each of the coordinates. Many scholars have suggested that we need something like a group denotation to get the semantics in order here. In (184) it is rather the join over the two denotations that should be the coordination's contribution to the sentence's meaning. The reason is quite clear. The coordination of (184) does not denote the set of spots on earth that are both Norwegian and Swedish, but rather the set of spaces that are (either) Norwegian or Swedish.

(182) De regering heeft [Smit en De Vries] bevorderd tot ridder in een of andere orde
'The government has Smit and De Vries promoted to knight in one or other order'
(183) De regering heeft [Smit en De Vries] belast met het onderzoek
'The government has Smit and De Vries charged with the investigation'
(184) In Noorwegen en Zweden rookt niemand meer
'In Norway and Sweden smokes nobody anymore'

The semantic effect of coordination may depend on functional properties of its environment, as was made abundantly clear by Zwarts (1986), and on denotational properties of the predication that the coordination is involved in (see Link 1983 for the original argument and *e.g.* Winter 2001 for an extensive treatment). These interactions may be steered by laws of logic and algebra, but it is impossible to identify an independent contribution to the semantics of a sentence made by coordination. Rather, the opposite is the case: in order to define the nature of the coordinative linking we have to interpret the sentence it is part of.

Does this position endanger compositionality? Janssen (1997) considers arguments against compositionality, as brought forward by Higginbotham (1986) with respect to the interpretation of *unless*. It is observed that *unless* behaves differently in different functional domains.

(185) Every person will eat steak unless he eats lobster
(186) No person will eat steak unless he eats lobster

In an upward-entailing environment such as the nuclear domain of a universal quantifier, *unless* denotes disjunction. In a downward-entailing environment it denotes conjunction. Janssen, who considers compositionality to be a requisite of grammatical architecture rather than a decidable property of language, offers two ways out in this case: either *unless* is constructed as a function from contexts to values, or it is considered to be ambiguous. We may try to subsume the context dependency of constituent coordination under this analysis. The first solution – context valuation – would not solve our problem, because there is no definite context available for coordination (see section 1.7.4.1). *Unless* precedes or follows a proposition offering the relevant context. Coordination is part of the context that should license its interpretation, since there is no proposition independent of the coordination. Moreover, coordinations may be part of each other's contexts, yielding circular traffic under this approach.
The second solution – inherent ambiguity – is the last resort. For conjunctions it would amount to a huge number of ambiguities, none of which would be locally decidable (Cremers 1993a). If conjunctions were multiply ambiguous, one would expect languages to come up with specialized elements, reducing ambiguity. Though some languages have a specialized element for group formation (a kind of 'with' element, see Payne 1985) these languages do not have explicit sentential coordination at all. Languages that have propositional conjunction, though, use it in all other environments, introducing massive ambiguity under this approach.

The propositional (and boolean) base of coordination is reflected in the proposal in Partee and Rooth (1983) to interpret all coordinations as the meet of propositions: $|[|[a]|$ and $|[b]|]| = \lambda\varphi.\ [\varphi(|[a]|) \wedge \varphi(|[b]|)]$, where φ's type can be derived from the coordinates' type. The argument to this function must be provided by the sentential construction surrounding the coordination. Here, the local indeterminism of coordination strikes back: such remnant environment cannot be established without parsing the whole construction and determining the coordinates. But that is the problem. Though Partee and Rooth succeed in generalizing the typological aspect of the meaning of the coordination, their proposal does not solve the conjunction paradox: in order to determine the coordination's contribution to the semantic construal of parsing, parsing must be completed including the determination of the coordinates. It is because of this paradox that we claim that there is no local semantic construal for coordination. The meaning of *John or Bill* and *John to swim and Bill to walk* does not exist independently of the semantic construal of a sentence.

The arguments above entail that conjunctions be treated *syncategorematically*, not subjected to the core categorial engines of grammar, as proposed in Grootveld (1994) from a generative perspective and in Cremers (1993a). We take Kempen's (2008) analysis that coordination is an updating construction, to be in the same spirit.

1.7.4.5 Ellipsis is coordination speciale

Everaert and Van Riemsdijk (2006) not only subsume coordination under other constructional phenomena but also the massive handbook lacks a chapter on ellipsis. That is not accidental. Coordination and ellipsis are intrinsically related, along the following lines of reasoning.

Firstly, to be elliptical is a prerogative of sentences. It does not make a lot of sense to talk about incomplete VPs or DPs, as they may occur discontinuously or underspecified in languages under normal business. Secondly, ellipsis calls for a propositional interpretation. There is no reason to call *six* an elliptical DP in the sentence *I have six*, but it is inevitable to call *John six* elliptical in the sentence *I had five and John six*. Third, ellipsis needs textual context to be interpretable. The sentences *John had five* and *I'll catch it* are perfectly interpretable as propositions without any reference to textual context. This is characteristic of certain anaphora, as was pointed out by Hankamer and Sag (1976). The sentence *John six* has no propositional interpretation outside a textual context licensing some semantic reconstruction. It is elliptical.

In short, we take ellipsis to be sentential, propositional and textual. In this vein, ellipsis is essentially defined as coordinative. It introduces a phrase of a category that is the canonical target of coordination and is interpreted as a proposition, and it has the same inferential status as the context it is licensed by – the logical function of conjunction.

This property can be illustrated by having a close look at instances of ellipsis for which coordinative analysis is not obvious, like VP ellipsis and comparatives; the elliptical phrases are in italics.

(187) Sue called John before *Bill did*
(188) Sue painted John more realistically than *Bill did*

Clearly, (187) is equivalent to the following sentence.

(189) Sue called John and Bill called John and Sue's call was earlier than Bill's call.

Bill called John is entailed, and this is because it is in 'semantic conjunction' with the main sentence: a conjunction of propositions entails each of its conjoints. In the same vein, the meaning of (188) must be represented as

(190) Sue painted John in a certain manner and Bill painted John in a certain manner, and Sue's manner of painting John was more realistic than Bill's manner of painting John.

Again, there is no difference in inferential status between the ellipsis and its licenser. The coordinative nature of comparison was pointed out by Hendriks (1995). The semantic and inferential co-ranking of ellipsis and its textual antecedent – backward ellipsis is a *contradictio in termini*s – explains why VP ellipsis cannot occur in sentences like

(191) * Sue sang so loudly that *Bill did*

VP ellipsis fails here, because the relationship between the intended propositions *Sue sang (with loudness x)* and *Bill sang (with loudness y)* is not just conjunction at top level, but (asymmetric) causation.

Elliptical constructions like gapping, sluicing and answering are explicitly coordinative, and bound to their antecedents in the sense that they cannot afford any textual intrusion.

(192) * I visited John, Sue went to Amsterdam and Mary Bill.
(193) * John visited someone, Sue went to Amsterdam but I forgot whom.
(194) * Who went to Amsterdam? Why are we here? John.

Therefore, we assume that ellipsis occurs exclusively in the (right) context of (semantic) coordination of sentences. Yet, coordination and ellipsis are not the same – coordination is basically not reductional but augmental. Under the present analysis, ellipsis is a subcase of coordination, a combination of augmenting and rearranging a preceding structure. Since we handle coordination syncategorematically, ellipsis is treated this way too.
The algorithm dealing with coordination identifies ellipsis as those instances of coordination for which categorial directionality is weakened. To illustrate the point, compare

(195) Destijds wou ik Susan en jij Marie benaderen
'in those days wanted I Susan and you Marie approach'
In those days I wanted to approach Susan and you, Mary
(196) Destijds wou ik Susan benaderen en jij Marie
'in those days wanted I Susan approach and you Marie'
(197) Destijds benaderde ik Susan en jij Marie
'in those days I approached Susan and you Mary'

The coordination algorithm deals with the first sentence straightforwardly: the components of both conjoints are adjacent at the level of the sentence-to-be. The second sentence is elliptic. To the left of the coordinator a full propositional sentence occurs; the right-hand side is not interpretable as such. The right-hand conjoint cannot be mapped onto a coherent sequence at the left. The conjunction is sentential, in all respects. (197) seems to be in between: there is a full sentence to the left of the conjunction, but the right-hand conjoint maps onto a coherent sequent of constituents to the left. But the sentence is not ambiguous in any interesting sense. Ambiguity, however, would be expected if coordination and ellipsis were different procedures. As a matter of fact, the sentences (195) - (197) share non-directional aspects of the argument structure – their *arity* and linearity. In order to parse ellipsis, then, we can restrict ourselves to applying the coordination algorithm under relaxed directionality in the presence of a full sentence. The computational nature of this algorithm will be discussed in section 1.8.8.

1.8 PARSING THE SYNTAX

The Chomsky hierarchy of formal languages (e.g. Hopcroft *et al.* 2001) connects languages, grammars and automata by stating that if languages are sets, their grammars embody programs to determine membership and that the differences in memory management between programs may reflect differences in the grammar. The hierarchy used to be the dominant frame for evaluating grammatical and procedural properties in the early days of computational linguistics, when storage was expensive and processing slow. But it is not technological progress alone that diminished the Chomsky hierarchy's importance. Firstly, theoretical linguistics itself seems to have lost its interest in the hierarchy: both for the generative enterprise and for its competition – from categorial grammar through HPSG to construction grammar – the hierarchy is too crude to catch up with the tendencies towards semantically relevant, fine-grained and construction specific grammatical analyses; there is more than context-freeness on earth and in heaven (*e.g.* Van Benthem 1991). Secondly, the parsers themselves turned out to be intrinsically more complex than the grammars or the grammatical structures which they were supposed to operate on as far as natural language was concerned (McCawley 1968): even if one describes natural-language grammar as a complex of regular structures, the overall system regulating the labour division is certainly not just finite-state.

Thirdly, of course, the hierarchy became less relevant in computational linguistics as grammars tended to be avoided in favour of non-symbolic models. Yet, the nature of the parsing process of a symbolic grammar is a central benchmark for two evaluations: first, comparison of the complexity of the natural-language recognition and interpretation problem with other problems and, second, its pretentions as a model of human language processing. We assume that these are different and even independent valuations. To say that natural-language recognition is nearly as hard as the Tower of Hanoi, chess or the Salesman problem is saying little about human language processing. To say that human language processing leans on memory routines rather than on on-line computation (*cf.* Daelemans and Van den Bosch 2005) is saying little about the problem of its mathematical complexity. Both in the domain of complexity as well as in that of modelling processing, the nature of the parsing procedure is distinctly relevant.

In this section we will reflect extensively on the DELILAH parser, by comparing it to the parsing of other grammatical frames, in particular to Combinatory Categorial Grammar (CCG) as revealed in Steedman (2000), for example. The section is largely based on previous work by Peter Zinn and Christiaan van de Woestijne (Zinn 1993, Van de Woestijne 1999); the latter author also developed the core of the chart parser. The coordination algorithm was added to the chart parser by Mathieu Fannee (Fannee 2006). Robustness strategies at the syntactic level were investigated by Poeder (1994).

1.8.1 The syntax of CCG

As stated earlier, CLG is closely related to Combinatory Categorial Grammar (CCG). The definition of CCGs below is based on Joshi *et al.* (1991).

(198) *Definition*

A CCG, G, is denoted by (V_T, V_N, S, f, R) where
V_T is a finite set of terminals (lexical items),
V_N is a finite set of non-terminals (atomic categories),
S is a distinguished member of V_N,
f is a function that maps each element of V_T to a finite set of categories,
$C(V_N)$ is the set of categories, where
$V_N \subseteq C(V_N)$ and
if $c_1, c_2 \in C(V_N)$ then $(c_1/c_2) \in C(V_N)$ and $(c_1\backslash c_2) \in C(V_N)$,
R is a finite set of combinatory rules.

CCG's central rule-scheme is generalized composition, again according to Joshi *et al.* (1991). It comes in two directions.

(199) *Generalized Composition forward*
$(x/y)\ (y|_1z_1|_2 \ldots|_mz_m) \Rightarrow (x|_1z_1|_2 \ldots|_mz_m) \qquad (m \geq 0)$

(200) *Generalized Composition backward*
$(y|_1z_1|_2 \ldots|_mz_m)\ (x\backslash y) \Rightarrow (x|_1z_1|_2 \ldots|_mz_m) \qquad (m \geq 0)$

Here, $x, y, z_1, \ldots, z_m$ are meta-variables over $C(V_N)$, and each $|_i \in \{\backslash,/\}$. For m=0, these rules correspond to function application and for $m > 0$ to function composition. The set R contains a finite subset of these possible forward and backward rules, i.e., for any given CCG only some of the combinatory rules will be available. Furthermore, restriction conditions can be put on calling combinatory rules in R. Rules can be constrained in two ways: the initial non-terminal of the category to which x is instantiated (the head) can be limited, or the entire category to which y is instantiated can be restricted. In Joshi *et al*. (1991), the mapping function f may also assign categories to the empty symbol, ε. CLG does not use this feature. Excluding the empty category makes CLG cycle-free: there is no non-terminal category $A \in C(V_N)$, such that $A \Rightarrow^+ A$, that is, no A can derive itself in one or more steps. CLG is cycle-free because each derivational step cancels exactly one category, while empty moves cannot occur (cf. section 1.5.4). Note that for m=0, these rules simplify to the Forward and Backward Application rules of classical AB grammars (Ajdukiewicz 1935, Bar-Hillel 1953).

1.8.2 Comparing the syntaxes of CCG and CLG

In CLG, rules (199) and (200) have been adapted. Rule (199) has been replaced by two instantiations, and repeated here for convenience. These two rules differ only in the order in which the lists of arguments are appended. These rule types are the only ones in CLG, as merging is the only combinatory operation in the syntax (cf. section 1.4.3).

(201) *Generalized Composition forward in CLG, discontinuous*
$(\ldots(p|w_1)|w_2\ldots.|w_m)/y\ \ (\ldots(y|z_1)|z_2 \ldots |z_n) \Rightarrow (\ldots(\ldots(p|z_1)|z_2 \ldots |z_n)|w_1|w_2\ldots.|w_m)$

(202) *Generalized Composition forward in CLG, continuous*
$(\ldots(p|w_1)|w_2\ldots.|w_m)/y\ \ (\ldots(y|z_1)|z_2 \ldots |z_n) \Rightarrow (\ldots(\ldots(p|w_1|w_2\ldots.|w_m)|z_1|z_2 \ldots |z_n)$

In the same vein, backward composition (200) has been instantiated by two CLG rules.

(203) *Generalized Composition backward in CLG, discontinuous*
$(\ldots(y|z_1)|z_2 \ldots |z_n) \quad (\ldots(p|w_1)|w_2\ldots.|w_m)\backslash y \;\Rightarrow\; (\ldots(\ldots(p|z_1)|z_2 \ldots |z_n)|w_1|w_2\ldots.|w_m)$
(204) *Generalized Composition forward in CLG, continuous*
$(\ldots(y|z_1)|z_2 \ldots |z_n) \quad (\ldots(p|w_1)|w_2\ldots.|w_m)\backslash y \;\Rightarrow\; (\ldots(\ldots(p|w_1|w_2\ldots.|w_m)|z_1|z_2 \ldots |z_n)$

The instantiations are logically not neutral. The CCG forward rule (199) is entailed by the discontinuous (201) but not by the continuous CLG forward rule (202). CLG allows for one more merge mode than CCG.

CLG defines several restrictions and alternations in comparison to CCG. The most relevant ones are listed below.

(a) in CLG, categories are required to be linear, that is, without any bracketing (Cremers 1989, 1993a). They may be represented as $y|_1z_1|_2 \ldots|_mz_m$, where y is primitive or atomic, and $m \geq 0$. A linear category is a particular instance of a CCG category. Compared to Lambek's set {n,s} of non-terminal symbols, CLG uses a larger and more fine-grained set. Yet, CLG non-terminals could be analyzed as fixed or de-activated {n,s}-complexes; the CCG category *np* represents a lambekian category *(n,(n,s)),* but is not addressed as such by the syntax. As far as we can see, nothing hinges on the cardinality of the primitives from a syntactic point of view; Roorda (1991) proved that even a one-type-categorial logic may work to distinguish what has to be distinguished, and Montague (1972) operates a two-type semantics but declares the object atom *e* (the lambekian *n*) syntactically inert. Regarding the number of non-terminals, there is no fundamental difference between one-, two- and multi-atoms approaches.
In CLG, due to categories being linear, an argument can only be cancelled against the primitive head of the secondary category. More accurately, when both the primary and the secondary category have been fully expanded and argument lists are not empty, *disharmonious composition* cannot be avoided in the sense that argument lists in the passive directory are also merged: disharmony is the standard option for composition when linear and complex categories are involved (cf. section 1.4.8). This is no different in CCG. As can be read from the composition schemes (199) and (200) where vertical slashes abstract from directionality, arguments of the secondary category pass over onto the resulting category even when their direction is opposite to the slash that induces the cancelling. In (205), you find two simple but legitimate instances of generalized composition, showing disharmony in the italicized negative type /*z*.

(205) $x/y \quad y\backslash z \;\Rightarrow\; x\backslash z$
$y/z \quad x\backslash y \;\Rightarrow\; x/z$

In standard categorial grammar of the lambekian breed, composition and application allow both for complex heads and for cancelling of complex types. This will not induce structural ambiguity, as the internal structure of complex heads is fixed by a particular bracketing. In the case of $np \otimes (s\backslash np)\backslash np$, the rightmost *np* is the only candidate for cancellation. Of course, the complex head can be abbreviated by an atomic label. However, atomic labels for complex heads hide linguistic information and will yield an uninteresting grammar (Zinn 1993). The computationally interesting property of using simplex labels as the trigger for rule invocation is applied in CLG, and formulated as *linearity*. When heads are required to be primitive (simplex) categories without any structure, calling a rule boils down to matching two basic (simplex) types – which is a cheap test.

(b) In CLG, the form of the categories has been adapted for both computational and linguistic purposes. CLG assigns categories of the format $x\backslash_1 l_1 \backslash_2 \ldots \backslash_m /_1 r_1 /_2 \ldots /_n$, where x is a simplex head, and l_i and r_i constitute lists L and R of left and right arguments, respectively, taken from V_N, and m, n ≥ 0. A category may then be written as $x\backslash L/R$. L and R may be the empty list [ ], for m=0, and n=0, resp. Non-empty lists L and R have been split further into regular (L_{reg}, R_{reg}) and special arguments lists (L_{sp}, R_{sp}). L_{sp} accommodates wh-movement or, better, any form of peripheral dislocation. Its size is one element at the most, kept at the end of the list of leftward searching arguments (L is a 'priority stack'). Its counterpart R_{sp} is currently not used (it is the empty list). In the definition of rules, CCG's general operator $|_i$ has been instantiated by the directional equivalents $\backslash_i$ and $/_i$. Moreover, the relative order of the leftward searching arguments is reversed with respect to the order displayed in traditional category notation: the leftmost element of both lists represents the 'searcher', which is to be satisfied first; it is the innermost argument. Thus a representation of the argument lists as Prolog lists – stacks, essentially – becomes straightforward.

(c) Some words – typically *wh*-words – are assigned a pair of categories, $C^{left} * C^{right}$, where C^{left} and C^{right} are single categories from $C(V_N)$. Such a pair is called a *double type* (or star-category). C^{left} and C^{right} have different selectional roles and can never be cancelled against each other, by definition (see section 1.6.2). Double types are not derived dynamically by some rule that can be applied repeatedly, but are defined statically in the lexicon.

(d) All arguments of CLG's categories are flagged by *modes*. These flags enforce certain restrictions on the applicability of reduction rules. For example, a flag

may require an argument list to be empty. Moreover, every rule instance must assign a flag value to the head of the resulting category. Modes of composition are included in CCG as well (Joshi *et al*. 1991, Baldridge and Kruijff 2003).

(e) With respect to the reduction rules used, CLG's Generalized Composition rules obey Count Invariance (see section 1.4.6), directionality and adjacency. Directionality means that if a category *c* asks for some argument category *c'* on its left, it should find the lexical element which introduces the head of *c'* to the left of the lexical constituent bearing the category *c*. The same reasoning is valid for right-searching arguments. Directionality prohibits permutation from being introduced. Steedman (1990) states that any combinatorics is to be limited by some form of adjacency. The inference engine of the grammar is supposed to instantiate the types in the antecedent of the axiom schemes only by adjacent types in the sentence, in the order given by the scheme. This restriction, too, prevents permutation from slipping in (Cremers 1993a). It can be reformulated by requiring that axioms with antecedent strings of more than two members are not allowed. This is stated by the Binary Derivability lemma (64), which excludes rules of the form $x \Rightarrow y$, where *x* and *y* are single non-identical categories, any form of type raising, i.e. $x \Rightarrow y\backslash y/x$, permutations of the form $x\,y \Rightarrow y\,x$, expansions of the form *if* $x \Rightarrow y$ *then* $x\,x \Rightarrow y$, contractions of the form *if* $x\,x \Rightarrow y$ *then* $x \Rightarrow y$, nor do they accept or derive empty symbols, ε. Van Benthem (1991) shows that (some combination of) rules like Permutation, Expansion, and Contraction even decrease the generative capacity of categorial grammars.

(f) CLG's rules are degree-decreasing: the number of basic categories on the right hand side is strictly lower (*i.e.* <) than that on the left hand side. Here, heads and arguments of the same basic type are cancelled in accordance with the Van Benthem count invariance protocol (Van Benthem 1986). Cremers and Hijzelendoorn (1996, 1997a, and 1997b) extend count invariance to free coordination and left dislocation.
Degree-decreasing rules imply that the language membership of CLG is decidable, as the height of the derivation tree is bounded upward by the length of the input string. To put it differently, for CLG, being a cycle-free grammar (cf. section 1.8.1), the length of a derivation – *i.e.* the number of steps – is linear in the length of the generated string. Exactly this theorem was proven for *context-free* cycle-free grammars by Harrison (1978:418). Stabler (1992) proves a closely related version: for any cycle-free context-free grammar, there is a linear bound to the size of the derivation tree – *i.e.* the number of nodes – as a

function of the length of the terminal string that is the yield of that tree. Both theorems illustrate the close relationship between CLG and CFG.

1.8.3 Comparing the generative capacity of CCG and CLG

The format of the categories in CLG differs from the format in CCG in several respects. We discuss the main differences, and explore the consequences for the generative capacity.

(a) In CLG, the category's arguments have been split into argument lists for leftward and for rightward searching arguments. The first element refers to the innermost argument. This gives greater freedom in choosing to reduce to the left or to the right, while in a standard categorial format this order is explicitly specified in the order of the arguments. This increases the strong recognizing capacity of CLG. That is: CLG recognizes more different structures than CCG. Maintaining more than one argument list does not affect the weak recognition capacity of CLG (Cremers 1999). CLG and CCG share the same string language, but may assign different structures to it. In CCG, using only categories in standard format, the same strong recognizing capacity can be achieved by adding categories created by mixing the left and right arguments in a different way from the categorization of the word. Because of Lambek's theorem $(a/b)\backslash c \Leftrightarrow (a\backslash c)/b$, these categories are (weakly) equivalent. The theorem is responsible for generating an exponential number of analyses for a constituent. Any derivation tree over CLG can be transcribed into a tree containing only categories in standard form. In principle, lexical entries might be assigned a large number of different categorizations. Because of rule restrictions, however, each lexical entry has only one typical categorization that is effectively taken into account. Thus, the lexicon can be organized in such a way that any category is supplied with one list of arguments (modulo the list of special arguments) in a standard order. This eliminates ambiguity on the lexical level. Nevertheless, as words can still have different categorizations, i.e. non-mixed-up equivalents, structural ambiguity is not taken out of the grammar.

(b) In CLG, argument lists are flagged for affectedness (*cf.* section 1.6.1). These flags can be seen as indices of the category's head (Van de Woestijne 1999; one could read a category $x\backslash a{\sim}L/u{\sim}R$ as having a head xau, for example, yielding a category $xau\backslash L/R$. By definition however, flags may change as a consequence of cancellation and composition; basically, therefore, one

could claim that the primary head is affected by cancellation and composition. This runs counter to the definition of head in CCG by Joshi *et al.* (1991); here, the head is essentially unaffected by composition. However, the flag on the result category does not constrain the applicability of a rule, but merely serves as an output feature. In order to stay within the CCG framework, we would have to abstract away from flags. On the other hand, flags do not contribute to the generative capacity of the grammar. This follows from the fact that both category names and flag names, hence any combination of them, come from finite sets. Affectedness flags limit the search space for the applicable rules, but do not exclude proper derivations. They are processing features, rather than syntactic markers.

(c) In CLG, *wh*-arguments come with a particular cancellation mode and are treated in a special way. In merging argument lists, they are 'suppressed' in that they always end up at the bottom of the merged list, in order to be left-peripheral when cancelled (*cf.* section 1.6.2.3.4). This special treatment is very restrictive, and affects neither count properties nor the preservation of directionality. The restrictions are the following: an argument list (whether lexical or merged) may contain one *wh*-argument at most and a *wh*-argument occurs only in a left argument list. These restrictions start out from the lexicon, and are maintained under composition. Clearly, the special treatment is context-free: in merging lists, check for the occurrence of *wh*-elements; if none is present, just append; if more than one is present, cancel the operation; if there is exactly one, put it at the bottom of the appended stack. The only effect of this treatment is on the word order: the *wh*-argument is bound to be 'consumed' at the left periphery of the categorial complex that introduced it.
The context-freeness of this special append operation of two lists L and R can be shown by constructing a pushdown automaton, or merely a pushdown transducer, that uses an input tape containing the category symbols of L, followed by those of R, and one stack, on top of which the dislocated element is pushed as soon as the category symbols of L have been written to the output tape, and that is popped as soon as the category symbols of R have been written to the output tape. Pushdown automata are equivalent to context-free grammars.

(d) Furthermore, in the grammar of Dutch certain pronouns must be allowed to be cancelled (deleted from an argument list) without being on top of the stack, *i.e.* without having a fixed position in the order of arguments. These pronouns occur (among other functions, which are not under discussion here) as left arguments or complements of prepositions. They come in three forms: *hier* 'here, this', *daar* 'there, that' and *er* 'there, it' (see also section 1.2).

By necessity, they are a distinct class of *pro-np*s. The rule concerning their cancelling differs from all other rules in grammar, in that it does not match the top of the stack but has to check a complete argument list. Still, processing it is context-free as the argument stack – not to be confused with the stack of a pushdown automaton – is simply a list, and cancelling, that is deletion of a list element, requires only regular operations. Even this dirty feature of Dutch does not unbalance the grammatical set-up, although *er*-pronouns lean heavily on *disharmonious composition*, a notorious composition mode beyond context-freeness, which we need to handle crossing dependencies, among others (*cf.* section 1.4.8). Neither the repression of *wh*-elements discussed above nor the anywhere-cancellation of *er*-type pronouns permutes argument lists.

(e) While expressing the rules of CCG, the syntax of categories is also involved. In CCG's Generalized Composition rules, the full internal structure of the secondary category is taken into account, while in CLG, composition is also sensitive to the full internal structure of the primary category. Shifting part of the load to the primary category does not increase or decrease CLG's generative capacity. In particular, 'opening' the primary category as such does not induce permutation closure, nor does it result in the so-called MIX languages (Cremers 1999).

(f) In CLG, two types of categorial assignments are used that may refer to themselves, to wit the *double type* for *wh*-elements, and the automorphism type for adjuncts. The syntax of a double type, represented by a pair of categories $C^{left} * C^{right}$, could compromise the grammar or its generative capacity when C^{left} and C^{right} are cancelled against each other. However, this is made impossible because of appropriate lexical assignments. The only disadvantage of double types, then, is having to treat one more category per sentential domain. Adding this constant number does not affect CLG's generative capacity. For double types see also section 1.6.2.2.6 and chapter 3.
Automorphism types are types that can combine with categories headed by a category from a subset from V_T to form a new category of that head. Their status is discussed in section 1.6.3. In CLG, this subset is restricted to some predicative and sentential basic types. Procedurally, this definition means that an inert type *automorphism*, when heading the primary category, is replaced by the head of one of the secondary category, while changing/constituting left or right argument lists up to the merge mode level. This is implemented by assigning a unique label to an automorphism in the lexicon and by applying a rewrite rule to the label once, and as soon as the automorphism type is involved in a derivation. The syntax of a rewrite rule takes the same form as

normal CLG rules. By its nature, a rewrite rule does not obey count invariance or reducibility, but is to be regarded as a non-recurring translation step that constructs a suitable instance of a combinatory rule as needed during the derivation process, and can be taken in polynomial time and space. This rewrite procedure prevents the lexicon from being overloaded with completely predictable forests of adjuncts. CLG's generative capacity is not affected by rewrite rules as they do not introduce new structures.

(g) CLG's Generalized Composition rules can be derived from CCG's Generalized Composition rules (199)-(200) as follows. In these rules, the primary and the secondary heads x and y may be complex categories by definition. Substituting $(p|w_1|w_2....|w_m)$ for x in (199) gives (202), taking time and space proportional to the number of categories, i.e. m+1. Thus, (202) is an ordinary rule of CCG, though written down more restrictively, to be derived from (199) in linear time and space. Such a substitution cannot be made to achieve (201), however, because separating the primary category's head and arguments cannot be done in one of CCG's Generalized Composition rules in constant time and space. However, although a category is defined recursively, each lexical category from V_N, and consequently each derived category from $C(V_N)$, has a finite internal complexity. It is easy to see that an algorithm exists splitting the head and arguments of a generalized category $(p|w_1|w_2....|w_m)$, taking time and space that are proportional to the number of arguments. In fact, Prolog's `append/3` predicate (in reverse) is such an algorithm. Specified as a Prolog list, splitting only costs constant time, as splitting a list into its head and tail is a one-step operation. By appending ('chaining') the arguments, which takes time and space proportional to the number of arguments of the primary category, the result category is achieved. Thus, (201) is an extraordinary rule of CCG, to be derived from (199) in no more than linear time and space.

The overall conclusion is that there is a polynomial mapping function from CLG's category format to CCG's more general form. CLG's rule syntax is a more practical, and slightly more liberal reformulation of CCG's rule syntax, and can be derived from it by polynomial means. CLG is maximally as complex as CCG; this means that, if the recognition task for CCG can be done in polynomial time, it can also be done for CLG in polynomial time. Although CLG's generative capacity is a little larger than CCG's, it is practical for the chapters to come to regard CLG as an instance of CCG.

1.8.4 CLG and the Chomsky hierarchy

The first categorial grammars only defined rules for Forward and Backward Application. Bar-Hillel *et al.* (1964) have shown that these grammars have the same *weak generative capacity* as context-free phrase-structure grammars. Lambek (1958) added new deduction rules to these A(jdukiewicz) B(ar-Hillel) grammars, in particular, rules deriving complex categories by hypothetical reasoning: if *a* and *b* go to *c*, *a* goes to *c* under hypothesis *b.* These rules add to the descriptive power of AB grammars, as they allow for more combinatorics, like *harmonic (function) composition* $x/y \;\; y/z \rightarrow x/z$ and $x\backslash y \; y\backslash z \rightarrow x\backslash z$, and *type raising* $x \rightarrow y/(y\backslash x)$ and $x \rightarrow y\backslash(y/x)$. Pentus (1993) proved that Lambek grammars are still context-free. Adding 'residuation modalities' to Lambek (1958) grammars does not extend their weak generative capacity beyond context-freeness (Jäger 2003). CCGs generalize the Forward and Backward Application rules by Generalized Forward and Backward Composition rules, respectively (Steedman 1990). This means that CCGs, including CLG, at least recognize the context-free languages. CLG's extended form of Generalized Composition cannot be exploited to recognize the permutation closure of the so-called MIX language $\{a^n b^n c^n : n > 0\}$ (Cremers 1999). The MIX language is more than context-sensitive. Adding modal operators plus structural postulates greatly increases the complexity of Categorial Grammars (Jäger 2003). The crux is what is to be understood as 'structural postulates'. CLG's rules of Generalized Composition can hardly be taken as 'structural', as they can be derived from CCG's rules in polynomial time. The same is true for the finite set of modal operators. Joshi *et al.* (1991) show that CCGs, *e.g.* CLG, are weakly equivalent to Linear Indexed Grammars, which are in the class of the Mildly Context-Sensitive grammars. Linear Indexed Grammars (LIGs) were defined by Gazdar (1985) as a restriction of the Indexed Grammars, introduced by Aho (1968) as a generalization of CFGs. Indexed Grammars may be described as CFGs in which a stack of so-called stack indices is associated with each non-terminal. Rule invocations can be limited by restrictions on the top of the stack. In a LIG, only one daughter non-terminal can inherit its parent's stack, instead of all daughters.

Joshi (1985) defined a class of formal grammars which are only slightly more powerful than CFGs, but which still allow for descriptions of natural languages in a linguistically significant way. This class, dubbed *mildly context-sensitive languages* (MCSL), is described by Joshi (1985) and Joshi *et al.* (1991) to have at least the following properties.

(1) CFLs are properly contained in MCSL;
(2) Languages in MCSL can be parsed in polynomial time;
(3) MCSGs capture only certain kinds of dependencies, such as nested dependencies and certain limited kinds of crossing dependencies;
(4) Languages in MCSL have the constant growth property.

CCGs, including CLG, are an extension of AB grammars, which are weakly equivalent to CFGs, satisfying (1). Polynomial time parsers for CCG do exist; see below. This satisfies (2). Regarding (3), Joshi *et al.* (1991) give the example of "subordinate clause constructions in Dutch or some variations on them, but perhaps not the so-called MIX (or Bach) language, which consists of equal numbers of *a*'s, *b*'s, and *c*'s in any order". CLG was designed to capture the Dutch verb cluster in subordinate clauses, including crossing dependencies. Obviously, CLG also handles various instances of nesting dependencies. As was mentioned above, CLG is not able to generate the MIX language. CLG, then, may be said to comply with (3). The constant growth property requires that the length of the sentences generated and put in order by the grammar grows linearly. This is certainly not the case for $\{a^{2^n} \mid n \geq 0\}$. In CLG there is no substantial numerical condition, like the sentence's length, on the generated structures. Look-ahead is restricted to one neighbour's category only, and by inspecting the first element of the list of leftward arguments and the first element of the list of right searching arguments, including the *wh*-position, at the same time. CLG, then, complies with (4). In conclusion we can say that the language generated by CLG is not inconsistent with MCSL.

The positioning of Categorial Grammar in the Chomsky hierarchy is not obvious. In the spirit of Van Benthem (1993), Jäger (2003: 106) states "that there is tight connection between interaction postulates and generative capacity. This may eventually lead to a taxonomy of languages that is much more fine-grained than the traditional Chomsky hierarchy." MCSL is an excellent candidate for an in-between class of 'reasonable' grammars that are linguistically relevant and semantically potent. If CLG lives in MCSL, it is in good company.

1.8.5 Parsing CCGs

For any computational problem, we are interested in algorithms that can solve the problem. We want to know how efficiently these algorithms operate in terms of running time, memory usage, or other computational resources. Most of the time, by 'most efficient' we mean 'fastest', as time considerations

are often the most important in practice. Time complexity relates the execution time of an algorithm to the size of its input. Here, the problem is deciding whether a string (a sentence) is in the language generated or recognized by a grammar of some type. A parser is an algorithm to solve this problem. For a CFL and an MCSL the problem is known to be polynomially solvable. That is: for any arbitrary input sentence, *i.e.* in the *worst case*, the largest amount of time needed by the parser to decide whether the sentence is in CFL or in MCSL is a polynomial function of the size of the input, *n*. Such a parser is called *efficient*. When the major term of the polynomial function is n^3, ignoring constants and lower orders of magnitude, the parser is said to run in cubic time, or, to put it formally: to have $O(n^3)$ time complexity. The 'Big O' notation is useful when analyzing and comparing the efficiency (or: scalability) of algorithms – especially when the size of the input becomes large – independent of properties of actual hardware. The key to efficiency is for a parser to be more rigid than the grammar. Where a grammar defines the allowed structures in a language, a parser blocks the redundant, albeit grammatical ones (Eisner 1996).

Above, it was shown that there is a relationship between CLG and CCG, which can be established in polynomial time and space. For practical purposes, we regard CLG to be an instance of CCG. A well-known parsing algorithm for CCGs, including CLG, and displaying polynomial time complexity is that of Vijay-Shanker and Weir (1990). It is based on a well-known parsing technique for context-free grammars, namely the chart parsing technique (Kay 1973, 1980), and runs in n^6 time.

Chart parsing algorithms are based on dynamic programming. This technique systematically fills in a table of solutions (or, more general: *items*) to sub-problems. When the table is complete, and spurious ambiguities have been removed (see below), it contains not only the solution to each sub-problem, but also to the problem as a whole. Because of this property, the way chart parsers handle alternative solutions is dubbed *parallel*, as opposed to *backtracking*, the latter keeping only one solution at a time in memory. Tabulation is necessary to face the inherent ambiguity of natural language. Chart parsing algorithms may differ as to the type of items they use (their data structure), and the processing order (which follows from the algorithm's control structure).

When parsing of a grammar *G* is the problem, the table is a chart (or: well-formed substring table). A chart is a two-dimensional array that relates *spans* (substrings, denoted by starting and ending position) to categories (or, more generally, constituents encoded by sub-trees). A sentence has a parse, *i.e.* it is

in $L(G)$, if the chart has an entry that relates the substring starting at position 1 and ending at position n to some designated symbol, for example s. For CFGs, a parser does not have to know how a constituent is constructed, but only that it can be constructed. As a consequence, there are Cn^2 different constituents possible to be constructed for a sentence of length n, where C is the number of different categories in the grammar. A constituent can be constructed in multiple ways. There are $O(n)$ different ways of building a constituent that is $O(n)$ in size. Building Cn^2 constituents in $O(n)$ ways takes $O(n^3)$ time, assuming adjacency of constituents. Adjacency is guaranteed by categorial combinatorics. Chart parsers, then, run in cubic time. Generated constituents that cannot be reached from a valid designated symbol, like s, are purged from the chart. A sub-tree is never computed more than once, and never stored more than once, so that it can be shared among parses, which explains the efficiency of chart parsers. However, as the chart is a compact data-structure, called a *shared forest*, containing analyses to sub-sentences, the analysis for the whole sentence must be constructed by putting all pieces together by travelling systematically through the chart. To retrieve the first analysis, this procedure takes polynomial time; to find all analyses, however, takes exponential time.

A chart, being a data-structure, is neutral with respect to parsing and search strategy. The same is true for CFG rules; they may be said to be declarative statements. In categorial grammar, however, the categories encode partial derivation trees. It is easy to see them as the items of a chart parser. From a parser's point of view, a category with an adjacent neighbour category directly triggers the calling of an appropriate reduction rule. In turn, the resulting category, together with another adjacent neighbour category, fires a reduction rule that fits their requirements, etc. until the designated symbol, *e.g. s*, is reached after a finite number of reductions. In other words: categorial grammar rules are one-way traffic, that is, directional. They are procedural. A bottom-up approach (or: data-driven search) is the natural control structure for a parsing algorithm for lexicalized grammars, like categorial grammar.

A top-down approach (or: hypothesis-driven search) for parsing our type of categorial grammar would be rather artificial. Given a hypothesis about a phrase, it is very inefficient to guess two categories that constitute the hypothesis, as categories can be fairly complex (recursive). This also means that a top-down grammar filter does not make sense. One might add statistics – *e.g.* extracted from treebanks – so as to determine a realistic hypothesis about a phrase, consisting of two possibly complex daughter categories. However, applying probabilistic means *to steer the derivation* is contra the rigidity of

our *lexicalized* syntax (cf. section 1.3): it might cut off possible analyses on non-grammatical grounds. Applying probabilistic means for selecting analyses post-deriviationally is unproblematic. Extensions to CLG benefit from a bottom-up parsing regime. The algorithm for removing spurious ambiguity operates in a bottom-up fashion. Regarding robustness, it is pointless to make any top-down hypotheses for partial derivation structures. A semantic filter will gain maximum efficiency only when it can be applied to fully specified logical forms, as early as possible and at any derivational level. Semantic structures that are created in a top-down fashion may be incomplete and therefore be of less use for filtering or constraining analyses. Top-down procedures that cannot be applied in parallel to a bottom-up parser for CLG, such as determining scopal dependencies (cf. chapter 2 on semantics), can only be applied post-derivationally, that is at the moment that all words of the input sentence have been processed and all data structures have been computed.

One of the main disadvantages of a bottom-up parsing strategy is that structures (sub-trees) are created that will never lead to the designated top symbol. As a top-down grammar filter cannot be implemented efficiently while filling the chart, because of the recursive nature of categories, nodes that are not descendants of a valid top node are filtered from the completed chart. While constructing the parse tree(s), disambiguation takes place by deploying a bottom-up semantic filter (see below). Empty categories – being another problem for bottom-up parsers – do not occur in CLG.

In addition to a bottom-up control structure, the parsing strategy needs a way to consume the input symbols. A straightforward implementation is a left-to-right approach, building one-word constituents first, starting at position *i-1,* and ending at position *i*, for *i*=1..*n*, then proceeding to two-word constituents, starting at position *i*-2, and ending at position *i*, etc. while each level builds on the results of previous levels, that is, building constituents from shorter to longer ones in a strictly bottom-up manner. Finally, a constituent is built from position 1 up to *n*. A left-to-right bottom-up processing order ensures that all analyses of shorter ranges are ready before reductions of longer ranges are tried.

Vijay-Shanker and Weir's (1990) algorithm for parsing CCGs is based on the Cocke-Kasami-Younger (CKY) algorithm (Kasami 1965, Younger 1967), which is an example of a bottom-up chart parser. The CKY algorithm requires the grammar to be in Chomsky normal form: grammar rules are of the form $A \rightarrow a$ with A a non-terminal and *a* a terminal symbol, or $A \rightarrow A_1\ A_2$, with A, A_1 and A_2 non-terminals.

1.8.6 Parsing CLG

The grammar rules in CCG, as defined by Joshi *et al.* (1991) and including CLG, can be said to be in Chomsky normal form, as they take the format of binary rewrite rules. This means that standard CFG parsing algorithms, including the standard CKY chart algorithm, but even a non-deterministic shift-reduce parser, will suffice as a parser for CCGs. Stated otherwise: standard CFG parsing algorithms, including chart parsers, are 'insensitive' to CCGs as defined by Joshi *et al.* (1991), including CLG. Obeying count invariance, directionality and adjacency (cf. section 1.4), CLG's rules can be processed locally. In general, a rule's format does not determine its generative capacity. For example, the disharmonious composition rule has a context-free rule format, but generates mildly context-sensitive structures. This is possible due to the fact that the two categories in a CCG rule entertain a relationship that can be defined by *any* function. Categories in a CFG rule are atomic and functionally independent. The categories in a CCG are *indexed*, as in Linear Indexed Grammars (cf. section 1.8.3). They are data-structures, or *complex symbols.* The locality of the context-free rule format compensates for the richness of information compiled in these data-structures. The combination of locality and rich information is reminiscent of the deterministic program of Marcus (1980), which merges grammatical specification and control.

CLG – like any grammar for natural language – includes large context-free sub-parts. In CLG, these sub-parts follow from applying only the basic case of the Generalized Composition rules, namely Application. This is enforced by using only some designated flags (to wit: /^*isl* for saturated constituents, /^*penins* for near islands, /^*sentop* for *wh*-islands, and \^*part* for particles) which require the secondary category's arguments list to be empty (cf. section 1.6.1). They reduce the grammar to a standard AB grammar, which has been proven to be context-free.

In general, however, CLG's generative capacity is beyond context-freeness. This means that there are families of sentences that a standard CKY algorithm will not parse in cubic time, but in exponential time instead. Vijay-Shanker and Weir (1990) explain this behaviour by noting that categories spanning part of the input can grow proportionally to the input size, and, as a consequence, be exponential in number. In CLG, this can happen only to categories occurring in the final verb cluster with crossing dependencies in Dutch, and which can have arbitrarily long sequences of arguments. However, these arguments can only be of type *np*. This implies that the number of possibilities for argu-

ment lists is bounded for each value of the list length, and does not exceed a boundary that is polynominal in the sentence length. By taking advantage of the fact that regardless of the length of a category only a bounded amount of information (its head and its innermost argument) is necessary for determining which rule to apply, Vijay-Shanker and Weir (1990) turn the exponential worst-case behaviour of the standard CKY algorithm on CCGs into polynomial behaviour, namely $O(n^6)$. Their chart-parsing algorithm is made sensitive to the characteristics of CCG. Komagata (1997) concludes that their method only removes a possibility that rarely occurs in realistic grammars. Experiments with a standard CKY algorithm on CLG for a typical set of Dutch sentences display an average-case behaviour of $O(n^3)$. The same behaviour is reported for experimental parsers for realistic fragments of English and Japanese (Komagata 1997, 1999). Parsing natural language, then, is just one of the problems that have bad worst-case performance, but good average-case performance.

1.8.7 Extending and restricting a parser for CLG

Natural language is ambiguous. A parsing algorithm for natural language will compute different structural descriptions for the same sentence. This is called structural ambiguity. In Lambekian categorial grammar, this ambiguity is even more apparent due to *type raising* $x \rightarrow y\backslash(y/x)$, an immediate consequence of the rule of slash introduction, and associativity of composition. For instance, *John likes Mary* might be parsed as $[_S$ *[John likes] Mary]* or as $[_S$ *John [likes Mary]]*, applying $(s/np)\backslash np$ and $(s\backslash np)/np$ for *likes*, respectively, which yields different parses for the *same* sentence with the *same* meaning. CCG's flexible notion of constituency allows for the structure *[John likes]*, which indeed must not be blocked as can be seen in coordinated sentences, like *John likes, but Harry hates Mary*. This type of ambiguity has its source in the grammar, and not in the language. Hence, it is called *spurious ambiguity* (Wittenburg 1986). On the other hand, rule-based parsers, confronted with a grammar or lexicon that is not *sound* (soundness: all word-forms are correctly defined by lexical entries) and not *complete* (completeness: the lexical entries are the only definitions of the word-forms) may not produce any analysis at all for some correct sentences. This asks for robustness.

1.8.7.1 Removing spurious ambiguity

Spurious ambiguity in Lambekian categorial grammar is caused by the property of *structural completeness*: when a sentence is grammatical, it can be derived under all binary bracketings (Moortgat 1988). This gives rise to an

exponential number of possible analyses. Approaches for eliminating spurious ambiguity can be distinguished in syntactically and semantically oriented methods. The syntactic method of Eisner (1996) only allows for normal-form parses. "A parse tree is in normal form, when the result of every rightward (leftward) [Generalized] Composition rule is not composed (or applied) over anything to *its* right (left). ... Thus, every functor will get its arguments, if possible, before it becomes an argument itself." Lexical categories are preferred to derived ones, and application is preferred to composition. This method will block the structure $[_{s/np}$ *John likes]* in *John likes Mary*: it is created by rightward composition and cannot serve as an argument of a functor to its right afterwards. This very structure will be allowed, however, in deriving *whom John likes:* it can serve as an argument of a functor to its left. Although Eisner (1996) specifies the restrictions for strictly rightward and leftward operating rules, he notes that disharmonious ('mixed') composition rules are not harmed. He proves that *all* spurious ambiguity originates in associative forward (backward) "chains", like A/B/C C/D D/E/F/G G/H, and that normal-form restrictions in fact only allow right-branching derivation trees. Any CCG parser can be easily adapted to produce only normal-form parses by marking the result of each rule invocation, and using the markings in the next rule invocation, *i.e.* by adding tags to the grammar rules. With the Dutch verb cluster – which is an instance of a backward chain – in mind (cf. section 1.2) it turns out that Eisner's tags are a bit too coarse for implementation in CLG. Instead, CLG makes use of carefully designed sets of modes and flags for a fine-grained control of the combinatorics, entertaining syntactically rigid derivations. For example, it distinguishes between islands, near-islands, and *wh*-islands. In principle, the grammar rules with modes and flags attached to them produce unique analyses to an arbitrary string of categories. This follows from the non-associativity of the CLG calculus. Thus, CLG avoids spurious ambiguity at the cost of giving up direct composition of fully specified semantics (but see the next paragraph on semantic methods for eliminating spurious ambiguity). However, in order to define CLG in a more principled way, some grammar rules are underspecified, which leaves some room for spurious ambiguity. A typical example is the attachment of adjuncts. Many cases can be removed at a local level by an adapted version of an algorithm by Hepple and Morrill (1989) that was developed in Vijay-Shanker and Weir (1990). Operating on a full chart, it inspects each triple of adjacent constituents and checks whether they have been combined in two different ways with the same result category. The only possibilities are a left-branching and a right-branching structure. We decided the right-branching parse to be the 'normal form' (cf. Kayne 1994) and removed the left-branching derivation tree (not the categories themselves).

By applying this method bottom-up and recursively, all and only the spurious ambiguity will be removed from the chart, according to Vijay-Shanker and Weir (1990). Their method is defined for spurious ambiguity that arises from the associativity of (generalized harmonious) composition. Since nothing in their method hinges on the *type* of composition, spurious ambiguity that might arise from CLG's *disharmonious composition* is removed as well. In conclusion, all possible sources of spurious ambiguity in CLG will have been removed before the chart is travelled in order to construct the parse trees.

Another main stream of methods that eliminate spurious ambiguity is semantically oriented. This approach, *e.g.* Karttunen (1986), eliminates a constituent when its logical form is equal to or subsumes a logical form that has already been derived and stored in the chart. This check might require exponential time. Note that the concept of a chart – storing computed structures only once – is exploited here for syntactic as well as for semantic descriptions. In general, if two interpretations share the same syntactic structure, it may not suffice to store this structure in the chart only once, because its semantic structure may require different variable bindings for different interpretations. Although sharing of semantic substructures is certainly possible, the concept of a chart is most efficiently exploited for syntactic structure. In CLG then, the chart is used only for syntactic purposes. After the chart has been filled and all spurious ambiguity has been removed, one or more parse trees are constructed from the chart. Semantic readings are constructed compositionally. These readings start out in the lexicon as underspecified Stored Logical Form frames which are unified in parallel to each derivation step. An SLF frame is a pre-derivational specification of the functor/argument relationships, which as such does not change during derivation. Derivationally, they will get more and more instantiated. A chart that has been freed from spurious ambiguity will only give rise to possibly plural but definitely unique logical forms. Where Eisner (1996) tries to keep only those syntactic parts together that will enable exactly one parse in each semantic equivalence class (by blocking non-normal form parses in the grammar), the CLG framework already assigns the semantic contours in the lexicon. Clearly, rigid grammar, avoiding spurious ambiguity, can only come with underspecified compositional semantics. (See also chapter 2 on semantics.)

1.8.7.2 Adding robustness

As for robustness, grammatical and lexical deficiencies need to be remedied. A chart parser that cannot assign a complete parse tree for an input sentence might assign sub-trees to parts of the input sentence. This only makes sense

when it is done in a bottom-up fashion. A *cover-over-a-chart* or *cover* for short is a sequence of sub-analyses to sub-parts of the input, together spanning the whole sentence. A cover consisting of one sub-analysis only, headed by a single sentential type, corresponds to the traditional notion of a parse tree. The definition of a chart parser guarantees that all possible sub-analyses are found and stored in the chart. Thus, the notion 'cover' does not apply to non-parallel parsing methods, such as a shift-reduce (backtracking) parser. Finding the best possible cover is a non-deterministic process, as it is an instance of an optimization problem. Possible evaluation criteria include: to what extent is the cover fragmented (the number of sub-analyses), to what extent are right-branching structures to be preferred over left-branching structures, and to what extent are the sub-analyses' top-nodes sentential and saturated. Of course, the application of these criteria depends on the type of information gathered in the parsing process. Currently, the DELILAH parser selects best parses primarily on the basis of fragmentation and secondarily, by inspecting the complexity of the fragments' logical form (see also chapter 2). But the available information is so extensive that many other selection procedures could be implemented too. Since robustness basically corrects grammatical deficiencies, fragmentation of the parse may also be repaired by relaxing those grammatical restrictions that resisted composition of combinatory categories or unification of complex symbols. These are three more or less independent sources of fragmentation:

- two adjacent fragments strand by lack of appropriate categorial typing – this results from underspecification;
- two categories resist composition because of not complying with the actual mode of composition (cf. section 1.4.4, 1.6.2 and 1.7.3) – this is due to overspecification;
- two complex symbols cannot be unified because of inconvergence of specifications – this too is due to overspecification.

Fragmentation resulting from underspecification – basically: the lack of appropriate combinatorial material – can only be repaired by adding new combinatory options, by means of a categorial hypothesis. Because the CLG operates a rigid syntax, inspection of a fragmented sequence may invoke an 'educated guess' on one or more combinatorial options that would have resolved the fragmentation. This is the general idea:

(206) *If* < C_1, ...C_j ... C_n > is a sequence of combinatory categories *and* $C_j \rightarrow \alpha \backslash [C_{j-1}...C_1]/[C_{j+1}...C_n]$, *then* C_1... C_j....$C_n \rightarrow \alpha$

The projected category is a hypothetical extension of the specification of the phrase with category C_j. Projecting this type of hypothesis is to be seen as a learning tool. It is less suited, however, to solve on-line parsing problems.
As for fragmentation resulting from overspecification: DELILAH's modalized grammar offers the opportunity to 'neutralize' modes of composition post-derivationally and see what happens to a cover when modal combinatorial restrictions are lifted. That is: two fragments may be composed according to generalized non-modal composition rules of the following type, where the merges of argument lists are chosen so that discontinuity effects are minimalized (cf. section 1.7.2):

(207) `a\List1/[b^nonmode|List2]  b\List3/List4 →`
`a\List1+List3/List4+List2`

If this relaxation leads to unification of the associated complex symbols and to a non-fragmented interpretation, this composition seems to be acceptable: it results from basically sound combinatorics. Note, moreover, that adding this post-derivational relaxation does not introduce an *everything goes* derivation – a form of robustness that would violate the semantic regime, which lives on the idea that being meaningful is not an accident but the outcome of structural subtlety.

Concerning robustness at the lexical level, information in the lexicon may be absent, incomplete or simply wrong. A simple, but rough approximation is to regard missing information to be a *name* (of type *np*). In categorial grammar, however, there are better options, since categories are not absolute, but relative encodings of the grammar's agenda. The arguments of a category refer to one or more neighbours to the left and/or to the right. When in an input sentence exactly one word is missing in the lexicon, an estimate of its category – modulo adjunctive types – can be derived from the other categories in the sentence by applying some count protocol (cf. Van Benthem 1986, Cremers and Hijzelendoorn 1997a). Such a protocol determines supply and demand of heads and arguments in a string of categories, and applies to CLG. Moreover, because of CLG's typical rigidity, the estimate is even deterministic, given some hypothesis as to the string's ultimate goal. The strategy applied here basically follows from 'inverting' the operation of reduction rules. This is possible for CLG, as will be demonstrated in section 1.9.1, where CLG's recognizing grammar is converted into a generating grammar.

This means that there are different *learning strategies* possible for the parser to construct or reconstruct a missing type. The options include a hypothe-

sis for the result type, *e.g. np*, the direction(s) in which missing categories are searched for, and the number of them in each direction. Strategies may depend on the position of the missing category in the sentence, and do not depend on the parsing algorithm, but operate on the level of the pre-terminal analysis of the input sentence. The matter is pursued in Van 't Veer (2007).

1.8.7.3 Semantic filter

In addition to the filter that eliminates spurious ambiguity, a semantic filter has been designed that ranks semantic readings post-derivationally. All logical forms that arise from the parse trees in the chart are valid and different. They differ mainly as to their 'lexical feed': the way the lexicon has delivered building blocks to the semantic combinatorics, and therefore the level to which lexical aggregation – semantic collocates, extended lexical units, constructions – is reflected. Clearly, a logical form accounting for lexical aggregation is in some sense more adequate than a logical form that does not. Sentence (208) is more sophistically represented by logical form (209) than by (210), though the latter can hardly be seen as a wrong interpretation.

(208) John kicked the bucket
(209) died'(john')
(210) ιx. bucket'(x) & kicked(j,x)

In Delilah, different readings are ranked by evaluating the semantic complexity of their Stored Logical Form at top level, *i.e.* at the level of the whole sentence. In successful analyses, small semantic structures that correspond to large parts are favoured above large semantic structures corresponding to small parts. This distinction, however, makes sense only when comparing viable interpretations; therefore, semantic complexity is a property of the analysis as a whole, and not of its parts. The semantic complexity is calculated by measuring certain properties of an SLF, including depth, items per store and number of the (resolved) semantic variables. Since all semantic structure stems from the lexicon, the degree of involvedness of the top-level SLF reflects the lexical semantics of the analysis. The ranking of readings resulting from this measurement can be used to select readings.
There is no canonical way to measure the complexity. The first implementation of this type of post-derivational reading selection was sketched in Cremers and Reckman (2008). There, the metrics were calibrated by human judgements on the outcome of the ranking. Other approaches may be equally valid. The important point is that SLF makes this computation feasible (see chapter 2) as part of the parsing process.

1.8.8 Parsing coordination

In section 1.7.4, it was argued that coordinative structures are handled extragrammatically by a dedicated algorithm, as the coordinates as such are not syntactically marked. Such an algorithm was developed in Cremers (1993a), and implemented in a backtrack parser according to Cremers and Hijzelendoorn (1997a). Fannee (2006) converted the algorithm into a tier of the chart parser. In both implementations, parsing a coordinated sentence consists of

(a) marking the boundaries of the left and right coordinates by finding a conjunction symbol; it has a combinatorially inert category, which is not taken into account by the parser
(b) parsing the left and right coordinates separately up to some 'fully reduced' level, which is slightly different, but comparable to an analysis of a cover, revealing that the coordination algorithm in fact implements a robustness strategy on incomplete phrases
(c) finding and matching levels of corresponding linguistic structures by applying a specialized, deterministic decomposition algorithm to the level of analysis pointed at by (b)
(d) constructing a top level analysis for each conjunct, *e.g.* a sentential analysis.

In abstracto, here is what the algorithm does. Given a sequence of categories, one of which is a conjunctive, with hypothetical reduction to α

(211) $[_{\alpha}$ c1 c2 c3 & c5 c6 c7]

the algorithm determines, by inspecting the nature of the categories at both sides of the conjunction, which category has to be decomposed in order to reveal a coordinate and which other categories form a coordinate, *e.g.*

(212) $[_{\alpha}$ c1 c2 $[_{c3}$ c4 *c5*] & *c5* c6 c7]

and resolves all elliptical structure to the left and the right of the conjunction, yielding

(213) $[_{\alpha}$ c1 c2 c4 c5 c6 c7] & $[_{\alpha}$ c1 c2 c4 c5 c6 c7].

Subsequently, the parser pursues the hypothesis that both sequences <c1, ..., c7> actually reduce to α according to normal syntactical and semantical combinatorics. More concretely, it computes the well-formedness and interpretability of the structure

(214) $[_s\ [_{c1}$ John$]\ [_{c3}\ [_{c4}$ loves$]\ [_{c5}$ his dog$]]\ [_{\&}$ and$]\ [_{c5}$ her puppies$]\ [_{c6}$ a lot$]]$

by transforming it into

(215) $[_s\ [_{c1}$ John$]\ [_{c4}$ loves$]\ [_{c5}$ his dog$]]\ [_{\&}$ and$]\ [_s\ [_{c1}$ John$]\ [_{c4}$ loves$]\ [_{c5}$ her puppies$]\ [_{c6}$ a lot$]]$

in order to test whether the sequences of categories to the left and the right of *and* reduce properly to S.

Determining the category in the span of which an outer border of a coordinate is 'hidden' – c3 in the examples (211) and (214) – is the algorithm's major task and achievement. The algorithm subsists on CLG's rigidity: instead of pumping up – in a particular context – the conjunction to some hypothetical category with structure $\beta\backslash\beta/\beta$ and testing that hypothesis on the present partial analyses, it sticks with the normal categorial reductions, and compensates for the lack of categorial flexibility with deterministic inspection of the reductions' result. Cremers (1993a) argues that this strategy is sound and complete in the following sense:

(a) it deals with both constituent and non-constituent coordination
(b) keeping the conjunction combinatorially inert and promoting it to a position 'outside of the brackets' respects the semantic integrity of the original phrasal conjunction.

Currently, the algorithm is defined and implemented for binary conjunctive structures only. Even this restriction is proven to complicate the parsing process for coordinations when compared to the parsing of non-coordinative sentences by significantly enlarging search space (Cremers and Hijzelendoorn 1997a) – the procedure by definition operates on partial analyses. This complication is inevitable, given the inherent syntactic underspecification of coordination discussed in section 1.7.4. There can be no doubt that allowing for multiple coordinations will again increase the complexity of the parsing task. This increase, due to partiality, is bounded only by the fact that the algorithm itself does not add to the complexity of the parsing process, as it works in polynomial time for each cover (Fannee 2006). The rise of complexity follows from the nature of coordination, not from the application of the parsing algorithm.

The core of the algorithm will suit the parsing of other coordinative phenomena, like discontinuous ellipsis. We expect this to be the case because the algorithm is essentially a repair mechanism, a 'late' and high-level implementation of grammatical knowledge, applied to solid syntactic information. All phenomena that cannot be detected or unveiled by normal combinatorics for composition of adjacent phrases can be delegated to this type of repair strategies.

1.9 GENERATING BY SYNTAX: AGENDAS AND LINEARIZATION

There are many good reasons for modeling language generation. Most of them point to the potential usefulness of generating natural-language systems for information retrieval and disclosure of any kind, as is widely discussed in Reiter and Dale (1997) and the many workshops on natural-language generation nowadays. In this section, we will concentrate, however, on modelling generation as the ultimate test for the adequacy of a computational grammar: your grammar is as good as the sentences produced by it. The question one can ask a grammarian, then, is: did you construct your grammar so that it is explicit and precise enough to generate well-formed and interpretable sentences without having resort to pre-established frames, scenarios or templates? If you want to know whether a grammar leaks, and where, build a free generator for it and evaluate its yield. This is the issue in this section. In the chapter on semantics, it will be argued, in addition, that a generator needs free generating capacity to generate from logical form, since logical representations are bound to be underspecified with respect to the sentences underlying them.

1.9.1 Two-directional grammar

From a computational perspective, the main difference between parsing natural language and generating natural language is linearization. When you parse a sentence, the order of words is input; when you generate, the order of words is output.

Prolog's definite clause grammars (dcgs) are famous for being applicable in two directions: the same set of rules can be used for parsing as well as for generating (Pereira and Warren 1983). As a matter of fact, the difference between parsing and generation evaporates in such a formalism. Although DELILAH is not founded on dcg, it is instructive to study the limitations of dcg. The reversibility of dcgs hinges on a tight connection between linearity and constituency: to be a constituent in a rule is to be well-ordered and continuous in a sentence. Literally between brackets, the constituents can be related to each other in every expressible manner – including *e.g.* semantic subsumption – but the linear ordering of a phrase cannot be manipulated. Fixing linear order is the backbone of reversibility in dcg. A slightly weaker but very insightful conclusion in this vein is drawn by Van Noord (1993:62) for reversible grammars in general. Moreover, it must be noted that any indeterminism in a dcg, *e.g.* caused by alternative expansions of rules, leads to irreversibility of the grammar in the sense that certain structures – those covered by rules lower in the

parsing hierarchy than their alternatives – cannot be generated without redefining Prolog's built-in depth-first strategy for unification and recursion. Consequently, every discontinuity in dcg has to be spelled out as a sequence of well-defined and well-ordered constituents. No essential variables can be used to cover a variety of strings: dcgs operate by recursive instantiation of strings, and string variables in the rules cannot be instantiated. Of course, this makes the whole complex of non-constituent coordination indefinable as a 'normal' rule in dcg. Since we do not deal with coordination in the combinatory grammar anyway, this does not have to prevent us from using reversible dcg as the combinatory engine. There is, however, another form of discontinuity in Dutch that transgresses the one-to-one correspondence between constituency and linear ordering. The problem is adverbial adjunction, again (cf. section 1.6.3). Consider the following sentence. The adverbial operator *waarschijnlijk* 'probably' can occur in each position marked by a dot.

(216) Toen heeft . een linguist uit Ter Apel . een boek van Chomsky . op internet . te koop aangeboden
then has . a linguist from Ter Apel. a book by Chomsky . on internet. for sale offered
'Then a linguist from Ter Apel offered a book by Chomsky for sale on internet'

In all cases of *waarschijnlijk* 'insertion', the adjunct must be related to a propositional domain – it is a sentential modifier – categorially headed by the finite auxiliary. At the same time, however, its scope with respect to other operators like the indefinite quantifiers has to be established (cf. Diesing and Jelinek 1995, De Hoop 1992). In a dcg, this would mean that for each possible position of *waarschijnlijk* in this sentence, its left and its right environment must be specified at constituent level in a separate rule of grammar. Although this would probably not push the grammar of Dutch completely outside the realm of finiteness, we clearly miss a generalization if every *vp*-design introduces a set of dcg rules to account for its internal distribution of adjunctive operators.

When one takes semantic reconstruction seriously, reversible grammar like dcg is not the right – nor most enlightening, most transparent, or most economic – instrument to tackle recursive discontinuity, the essence of syntax: discontinuity arising in structures embedded in discontinuous structures, like movement out of an extraposed or otherwise dislocated constituent. DELILAH does not entertain reversible grammar but operates two related, interdependent, yet distinct syntaxes for parsing and generation. The main difference between the two grammars is that generation must account for composition of segments that may turn out not to be adjacent in the final string. To put it directly: the rule that induces the unification of a verbal structure

and one of its arguments must result in a linearization structure that leaves room for an intervening verbal adjunct. That is: the resulting structure must acknowledge a well-ordered string segment in which an adjunct (or another intervener with a suitable category) can reside – whether that intervention actually 'occurs' or not. The generating rule format has to keep track of *possible* or 'future' linearizations.

The rule format for parsing has been discussed above, and mainly dealt with merging stacks of arguments under composition. Empty stacks – unless specified otherwise – behave as empty lists under append. The rule format for generation deals with segments of strings, since linearization has to be arranged. A category in a generation rule is a quadruple of strings *<WhString, LeftString, HeadString, RightString>.* The role of each (sub)string is indicated in its name. A generation rule is associated with a mode according to which two quadruples are compiled into a new one. The basic operation is a reordering of (sub) strings. To compare the formats, we repeat the parsing rule for mode */^open* (see section 1.6.2.2.3) here, and compare it to its generating or linearizing counterpart.

(217) PRIM \ u~PLA / _~[SEC^*open*|PRA] $\otimes_{\text{open}}$ SEC \ _~SLA/ _~[ ] →
PRIM \ LF~SLA+PLA / a~PRA

(218) <WHP,LEFTP, HEADP,RIGHTP> $+_{\text{open}}$ <WHS, LEFTS1+LEFTS2, HEADS, RIGHTS> →
<WHP+WHS, LEFTP+LEFTS1, HEADP, RIGHTP+LEFTS2+HEADS+RIGHTS>

In the generating rule, + stands for string concatenation. The generating rule is invoked by the primary category, according to an agenda (see section 1.9.2). At the moment of composition, the primary category must claim four fields: one for its *wh*-arguments, one for its other left arguments, one for its head, and one for its right-hand-side arguments; these form the quadruple of the primary category in the generating rule (218). The same holds for the secondary category, addressed by the agenda. At the moment of merge, the claims on the fields are just that: the final structure of each category and the saturation of its arguments may be unknown. The */^open* merge mode leaves the possibility that some left arguments of the secondary category are saturated before this particular merge, while others are absorbed after the merge. Therefore, the generating rule has to introduce a split of LeftS, LeftS1 and LeftS2; these lists occur at different positions in the final line-up, the line-up corresponding with the resulting category in (218).
The secondary head string *HeadS* has been integrated in the right field of the resulting quadruple. This is the counterpart in the concatenation algebra to

being cancelled in the algebra of types. One can lose types by cancellation, but one cannot get rid of strings combinatorially – the grammar of natural language is resource-sensitive. For that reason, the leftmost string of the resulting category must account for the *wh*-deliveries of both the primary and the secondary category, although at least one of them must end up empty. Which one, is to be determined by the generating \^*wh*-mode.
In the same vein, the generating rules cannot control the derivation, like requiring lists – string-fill recipes – to be empty or emptied, as is done in (217). Instead, (218) positions the secondary right field in such a way that it being claimed cannot influence the ordering of the other strings: it is peripheral in the resulting category.

When one does not use a reversible grammar as such, the question arises how to travel from one grammar – *e.g.* the parsing one – to the other. Hitherto, we have not implemented a general translation from parsing rules into generating rules, but constructed the generating grammar by hand out of the parsing grammar. For the sake of the computability of Dutch, however, it would be rewarding to have an algorithm performing such a translation. In order to achieve such an algorithm, we first have to formulate a very general relationship between categories and concatenated strings *instantiating* the category:

(219) C ::= w (*C is instantiated by w*) iff w is a string of category C;
[C^1, ..., C^n] ::= w^1+...+w^n *iff* C^i:=w^i and the list of categories is a left argument list;
[C^1, ..., C^n] ::= w^n+...+w^i *iff* C^i:=w^i and the list of categories is a right argument list;
[ ] ::= ε, where ε is the empty string and for all strings *w*, *w*+ε = ε+*w* = *w*;
H_~LA / _~RA ::= w^{WH}+ $w^{LA\text{-}}$+w^H+w^{RA}, where for some *XP^wh* in LA, XP^wh ::= w^{WH}, and $w^{LA\text{-}}$ instantiates the remainder of *LA.* The string w^{WH}+ $w^{LA\text{-}}$+w^H+w^{RA} is represented above and below as a quadruple <*WhString, LeftString, HeadString, RightString*>, with appropriate re-labeling.

In this instantiation, a category is associated with a quadruple of strings w^{WH}+w^{LA}+ w^H+w^{RA}. Only the string instantiating the head w^H is real, representing the phonological phrase to which that category is assigned. The other three strings are virtual, in the very same sense as the types in the argument list occur negatively until cancellation. Every category constructs a linearly ordered space of strings. Given this mapping from (lists of) categories to (concatenation of) strings, we can conjecture the declarative outline of a composition-to-concatenation translation for, *e.g.,* the rightward cancelling sort of rule, like (217). It must be noted, though, that CLG rules give rise to some degree of non-determinism, allocated in the input conditions of the cancellation modes, the flags (cf. section 1.6). Since flags are declared only at negative

occurrences of types (*i.e.* in argument lists) and not on heads, a certain context may comply with more than one cancellation mode – one cannot exclude this possibility. For that reason, the generating, concatenative rules must follow the pattern of the cancellation modes. With this starting point, here is a translation procedure for the class of rightward cancelling rules; the case for leftward rules is comparable. The result of the translation is a purely concatenative grammar.

(220) *Translation of recognizing grammar into producing grammar*

Given a rightward cancelling rule

PRIM \ Flag1~PLA / Flag2~[SEC^*mode*|PRA] $\otimes_{mode}$
SEC \ Flag3~SLA/ Flag4~SRA $\rightarrow$
PRIM \ Flag13~SLA$\oplus_{mode}$PLA / Flag24~PRA$\oplus_{mode}$SRA

create a generating concatenative rule
<WHP, LEFTP, HEADP, RIGHTP> $+_{mode}$
<WHS, LEFTS, HEADS, RIGHTS > $\rightarrow$
<WHPS, LEFTPS, HEADP, RIGHTPS>

where all the elements of the quadruples in the concatenative rule represent strings such that

HEADP represents the string to which the category
PRIM \ Flag1~PLA / Flag2~[SEC^*mode*|PRA] is assigned
HEADS represents the string to which the category
SEC \ Flag3~SLA/ Flag4~SRA is assigned
WHP is the string such that for some XP^wh in PLA, XP^wh ::= WHP and LEFTP instantiates the remainder of PLA
WHS is the string such that for some XP^wh in SLA, XP^wh ::= WHS and LEFTS instantiates the remainder of SLA
RIGHTP represents the string to be formed by concatenating the strings that introduce the types in PRA, and the empty string if PRA = [ ]
RIGHTS represents the string to be formed by concatenating the strings that introduce the types in SRA, and the empty string if SRA = [ ]
RIGHTFIELDS = RIGHTP + RIGHTS if PRA is the prefix of PRA$\oplus_{mode}$SRA
RIGHTFIELDS = RIGHTS + RIGHTP otherwise
LEFTFIELDS = LEFTP + LEFTS if PLA is the prefix of SLA$\oplus_{mode}$PLA
LEFTFIELDS = LEFTS + LEFTP otherwise

where + is simple concatenation, and furthermore

a. *if* SLA = [ ] *then* RIGHTPS = LEFTS+HEADS+RIGHTFIELDS *and*
LEFTPS = LEFTFIELDS
else if SLA ≠ [ ] and Flag3 = u *then*
RIGHTPS = HEADS+RIGHTFIELDS *and*
LEFTPS = LEFTFIELDS
otherwise for some suffix LEFTS2 of LEFTS,
RIGHTPS = LEFTS2+HEADS+RIGHTFIELDS *and*
LEFTPS = LEFTFIELDS

b. *if* FLAG1 = wh *then* WHPS = WHP
else if FLAG3 = wh *then* WHPS = WHS
otherwise WHPS = WHS+ WHP

Basically, the mapping, for each string-to-come defines in which region the string is bound to dwell; this region is defined by the given strings of the primary and secondary categories. A rightward cancelling rule *generates* if there is a generating concatenative rule that simultaneously meets the constraints above. This representation of a generating grammar is operationalized by specifying in each lexical template (complex symbol) not only a triple compositional category *a\b/c* but also a quadruple concatenative category *<w,x,y,z>*, and by having both categories related to proper parts of the template. The nature of this linking will be discussed in the chapter on the lexicon and unification. The way the quadruple is specified differs quintessentially from the way the triple is represented: the quadruple is derived post-derivationally by an operation *makestring/3* that is parametrized inside the template by unification of the strings and the modes to operate on. *makestring/*3 is defined in (221) as a Prolog predicate, operating on derivationally given quadruples to determine a new one – the quadruple holding the linearization of the template itself. The definition is followed by a scheme for a template in which it is functional; here, in quadruples *e* stands for the empty string.

(221)
```
makestring( OutString, [], OutString ) :- !.

makestring( In, [(FirstMode, FirstArgumentString)|Rest], OutString :-
       generating_rule(In ⊗FirstMode FirstArgumentString → Result ),
       makestring( Result, Rest, OutString ).
```

(222) *Template-scheme for triple and quadruple categories*

```
...
triple-category: a\_~[b^mode1] / _~[c^mode2]
quadruple-category: A
string: makestring( <e,e,Headstring,e>,  [(mode1, <W1,X1,Y1,Z1>),
                                          (mode2, <W2,X2,Y2,Z2>)],  A)
...
head: string: Headstring
...
argument(1):   type:b
               mode:mode1
               string:<W1,X1,Y1,Z1>
...
argument(2):   type:c
               mode:mode2
               string:<W2,X2,Y2,Z2>
```

Under such formalisms, the generator can be adapted into an excellent test apparatus for the parser if every cancelling rule generates. In Functional Grammar, exploring two-way grammars has been good practice in the computational enterprise from the very start (e.g. Kay 1981). This, however, does not imply that parsing and generation are themselves reversible processes. (222) shows that there is a homomorphism from the cancellation algebra into concatenation, but the mapping is not bijective. The cancellation algebra essentially computes types to get the lambda conversions right, checking linearization in order to avoid ambiguity or invalid unification. The concatenation algebra focuses on linearization only. Because linearization comes with discontinuity – dismantling constituenthood and adjacency, the prerequisites for categorial composition according to Steedman (1990) – strings cannot be the sole basis for appropriate lambda conversions. Thus, there is no homomorphism from concatenation into cancellation. The grammar is two-sided, translatable and computable, but not reversible. As a matter of fact, the triple categories steer generation in a way that is described in the next section.

When, for the sake of parsing, one needs cancellation rules that do not generate, these rules are at least suspicious. They endanger the testability of the grammar, and question the idea that we have a homogeneous grammar for each language. One such rule might be the one dealing with the recovery of so-called *r*-pronouns in Dutch (see also (23)). It allows for an 'anywhere' appearance of the category *rnp* and has to be defined recursively.

(223) *rnp-special rule*

```
PRIM\ L / _RF1~R ⊗rnp rnp \u~[] /u~[] → PRIM\ L / a~RR if
      append( PRA, [pp^6|SRA], R ) and
      rnp\u~[]/u~[] ⊗isl pp\_LF1~[rnp^isl]/PRA →
                    pp\a~[]/PRAP and
      append( PRA, [pp^rnp| SRA], RR )
```

The rule states that an *rnp* can be cancelled in any position if the primary category gives rise to a *pp*-argument that licenses the *rnp*, and that the *pp* has to be marked for being saturated with respect to that *rnp*. This rule is meant to be exceptional as a parsing rule, and it does not generate, because its context-free rule format is conditioned. We are not aware of any grammatical analysis for the phenomenon that could help us out of this impasse. Interestingly, the situation is less complex from a linearization point of view: whenever a *pp* looking for an *rnp* is absorbed, the *rnp*-string is integrated with the left field of the absorbing category. The problem with the linearization rule, though, is that it cannot be translated from (223). As a consequence, we are forced to leave open the possibility that some 'abnormal' parsing rules must be manually turned into linearizing and generating rules. Because this linearization procedure is completely opportunistic and *ad hoc*, we will not try to specify it here.

1.9.2 Categories as agendas

Reiter and Dale (1995) define generation for applications by means of a layered architecture involving at least three stages: text planning, sentence planning and linguistic realization. The generation we describe here is, of course, not an application. Yet, it may be useful to locate the strategies of the DELILAH generator as living in the interface of sentence planning and linguistic realization. Our generator creates a meaningful sentence by following a syntactical strategy. It may have as a goal to produce a sentence with certain detailed semantic properties or even expressing a certain proposition. This goal, however, is achieved by piecemeal, unification-oriented construal. In chapter 2, on semantics, a procedure will be given to generate from fully-specified logic in this manner.

Here, we concentrate on the use of combinatory categories in the generation process, for any purpose and for any input. The internal structure of these categories sets off a sequence of lexical lookup, unification of complex symbols and update of the agenda.

In the remainder of this section, we will offer a shortcut through a generation process by numbered statements. Below is the generation scheme, focusing

on agenda management. The templates are represented by suggestion, rather than by detail. The indexation of arguments is considerably simplified.

(224) *Concise overview of a generation procedure*

(a) initialization: determine the categorial hypothesis: `s`
(b) determine a lexical template with type: `s\_/_`
(c) found (randomly): *wenst* 'wish'

T1 = `[...HEAD:CONCEPT:`*wish*`...SEM:`***f***`(..wish..)...`
`TYPE:` *s\u~[np^wh#1]/u~[np^isl#2]*`...`
`ARG(1):...`
`ARG(2):...PHON:`*makestring(..wenst..)*`]`

(d) fix agenda (by cancellation and merge):
`given: [s]`
`to_find: [np^wh#2, np^isl#1]`
(e) retrieve a lexical template with type `np\_/_` and properties as determined by the value for the feature `ARG(2)` in the `s` template
(f) found (randomly): *wettig* 'lawful'

T2 = `[...HEAD:CONCEPT:`*lawful*`...SEM:`***g***`(..lawful..)...`
`TYPE: np\u~[]/u~[n^isl#3]...ARG(3):...]`

(g) fix agenda (by cancellation, merge and unification T1 ⊔ T2):
`given: [s]`
`to_find: [n^isl#3, np^isl#1]`
(h) retrieve a lexical template with type `n\_/_` and properties as determined by the value for the feature `ARG(3)` in the `s` template
(i) found (randomly): *goud* 'gold'

T3 = `[...HEAD:CONCEPT:`*gold*` ...SEM:`***h***`(..gold..)...`
`TYPE:` *n\u~[]/u~[]* `...]`

(j) fix agenda (by cancellation, merge and unification (T1 ⊔ T2) ⊔ T3):
`given: [s]`
`to_find: [np^isl#1]`
(k) retrieve a lexical template with type `np\_/_` and properties as determined by the value for the feature `ARG(1)` in the `s` template
(l) found (randomly): *langzaam* 'slow(ly)'

T4 = `[...HEAD:CONCEPT:`*slow*`...SEM:`***k***`(..slow..)...`
`TYPE:` *np\u~[]/u~[n^isl#3]*`...ARG(4):...]`

(m) fix agenda (by cancellation, merge and ((T1 ⊔ T2) ⊔ T3) ⊔ T4):
`given: [s]`
`to_find: [n^isl#4]`
(n) retrieve a lexical template with type `n\_/_` and properties as determined by the value for the feature `ARG(4)` in the `s` template
(o) found (randomly): *water* 'water'

T5 = `[...HEAD:CONCEPT:`*water*` ...SEM:`***m***`(..water..)...`
`TYPE:` *n\u~[]/u~[]* `...]`

(p) fix agenda (by cancellation, merge and (((T1 ⊔ T2) ⊔ T3) ⊔ T4) ⊔ T5:
`given: [s]`
`to_find: []`

(q) apply `makestring/3`: `wettig goud wenst langzaam water`
'lawful gold wishes slow water'

apply **f∘g∘h∘k∘m**

T = `[... HEAD:CONCEPT:`*wish*`, HEAD:PHON:`*wenst*`...`
`SEM:` *quant(A,gen).[water(A) & slow(A) &*
quant(B,some).[gold(B) & lawful(B) &
quant(C,some).[wish(C) & state(C) &
experiencer_of(C,B) & theme_of(C,A) &
attime(C,D) & tense(C,pres)]]]...
`TYPE:`*s\a~[]/a~[]*`...`
`ARG(1):PHON:`*langzaam*`...`
`ARG(2):PHON:`*wettig*`...`
`ARG(3):PHON:`*goud*`...`
`ARG(4):PHON:`*water*`... ]`

The yield of the generation procedure is a template that contains at least linearization of phonological units and composition of semantic functions. The linearization and the composition are the main results, and they are fed by incremental unification of templates during the procedure. Chapter 3 elaborates on this unification process. It is important to stress here that the incremental unification is steered by the agenda which is cast in terms of the output categories of the syntactic composition rules. The 'agendafication' of an output category is straightforward. For any category (225) that is the result of a cancel-and-merge rule called for in the derivation, the agenda it imposes on the subsequent generation procedure is (226).

(225) `Head\_~L1` $\oplus_i$ `L2/_~R1` $\oplus_i$ `R2`
(226) given: `Head`
to_find: `L1` $\oplus_i$ `L2 and R1` $\oplus_i$ `R2`

Technically, the agenda is added to the existing one under cancellation of the *given*-value from that agenda, if possible. If not, both the head and the appended argument lists are added to the *given* and *to find* values, respectively. Below is the generating algorithm, amounting to an agenda regime; first, the well-known notion of logical *backtrack* is defined procedurally.

(227) *Backtrack*
the statement *backtrack* p_1, ...p_{n-1} *until* p_n means:

find a valuation V *such that* for all i, V(pi) is true
if such V exists *then* proceed to the next statement
else regress to previous statement

(228) *Generating algorithm*

a. *input* **hypothesis** H
set **agenda** *to* <given: [ ], to_find: [H]>
set **structure** *to* [ ]
b. *backtrack*
c. *if* **agenda** = <given: [ ], to_find: [Only]> *then*
select a template T^{only} *with* category(T^{only}) = `Only\L/R`
set **structure** *to* [T^{only}]
set **agenda** *to* <given: [Only], to_find: L+R >
endif
d. *if* **agenda** = <given: Given, to_find: ToFind> *then*
e. *backtrack*
f. *if* Given = [First | Rest] *then*
extract a T^{First} *from* **structure**
select a template T *such that for some rule*
`category(T) ⊗ First\L/R → New\LNew/RNew`
set T^{New} *to* T ⊔ T^{First}
endif
g. *if* Rest =/= [ ] *then*
for some R ∈ Rest *determine some rule*
`New\LNew/RNew ⊗ R\L7/R7 → New\LN2/RN2`
extract a T^{R} *from* **structure**
set T^{New} *to* T^{New} ⊔ T^{R}
replace in **structure** T^{First} *by* T^{New}
remove T^{R} *from* **structure**
set **agenda** *to* <given: Rest-[R]+[New],
to_find: LN2+RN2+ToFind>
else
replace in **structure** T^{First} *by* T^{New}
set **agenda** *to* <given: [New],
to_find: LNew+RNew+ToFind>
endif
until Given = [H] % end backtrack (e)
h. *backtrack*
if ToFind = [First | Rest] *then*
select a template T *with* category(T) = `First\L/R`
% by now, structure is a singleton set
extract T^{Only} *from* **structure**
set **structure** *to* [T^{Only} ⊔ T]
set **agenda** *to* <given: Given,
to_find: L+R+Rest>
endif
until ToFind = [ ] % end backtrack (h)
i. *endif* % end *if* (d)
j. *until* **agenda** = <given: [H], to_find: [ ]> % end backtrack (b)
k. *output* **structure**

Basically, the algorithm tries to find templates and to unify them according to an agenda which is set by an initial hypothesis and updated by applying combinatory categorial rules. The agenda consists of two parts: *given*, corresponding with complex symbols already adopted, and *to_find*, corresponding with structures still to be checked. The result of successfully unifying complex symbols according to the agenda is accumulated in a store *structure*. The procedure succeeds if the *to_find* part of the agenda can be discarded, and the *given* section contains only the initial hypothesis. The complex symbol in *structure* is the yield of the procedure. The repeat-loop (228)b-j does the job. It checks the agenda and acts accordingly. Firstly, it tries to reduce the *given* part to a single occurrence of the hypothesis, in (228)e-g. After that, it looks for complex symbols matching the *to_find* agenda, in (228)h. In practice, every step of the algorithm can be sugared with additional directives. For example, in steps (228)f and (228)g it might be desirable to have the algorithm select templates with categories headed by the hypothesis. In general, the *select* subprocedure can be regimented by shortcuts respecting the agenda. As a matter of fact, the procedure we propose here amounts to posing additional restrictions on *select*. In particular, *select* will be restricted by conceptual conditions derived from logical form. These conditions are discussed in chapter 2.

Halting is not an intrinsic property of the procedure: the length and the complexity of the expression to be produced are not fixed. For practical purposes, however, we may impose restrictions on the length of the sentences to be produced. Suppose that is the case. Given such a restriction, we must investigate the soundness and completeness of the procedure in the following sense:

(229) *Soundness*
for every template T used in the production of a sentence S, there is a parse of S using T

(230) *Completeness*
for every template T used in the parse of a sentence S, there is a production of S using T

The soundness of (228) depends on the soundness of the translation (220) and the completeness of the parser. The generator cannot be better than the parser. We assume that by the translation and unification procedure underlying both parsing and generation, the generated sentences are among the parsable ones. This is an empirical matter, however. The generator can always be tuned or even made sound in this respect by having its output tested on the parser. Note that this does not invalidate the idea that the generator can be used to test and cure the parsing grammar.

The completeness of (228) is a much more complex affair. The question is whether we can prove that every grammatical expression – within the limits of the parsing grammar – can be produced, *i.e.* can be part of a successful production. The burden of proof is on the design of the selection procedure. In favour of completeness are the following considerations:

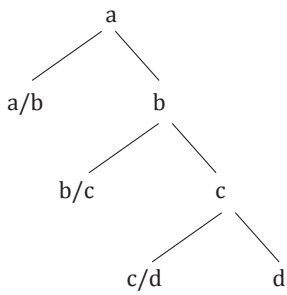
- the selection for a particular agenda can be organized as trial-and-error by pop-up without return from a randomized stack;
- the stack on which selection operates is finite by definition and thus backtracking is enhanced;
- there are no inhibitions for the introduction of complex categories, including recursion.

The last argument *pro* completeness, however, is dangerous. Under the assumptions of limited length, the complexity of categories must be controlled somehow. To make this work, we have to assume that for every type X (with a possible exception for the hypotheses) representatives with zero categorial complexity are available, *i.e.* templates of the category X\u~[ ]/u~[ ], for only selection of these arguments reduces the arity of the *to find*-agenda. This requirement is met by the parsing grammar, however. Under the principles of count invariancy (47) and antisymmetry (cf. section 1.5.3) of the combinatorial rules, the parser can recognize sentences only if a few of these 'zero' categories exist lexically. If we – additionally but perfectly reasonably – assume that every instance of a complex category can be reduced by some cancellation, the requirement that zero argument categories are available is met – actually, by virtue of some normality condition on the grammar. At the same time, categories of a certain complexity may be the only solution to the agenda problem. The situation of (228)g can be solved only if there are categories which have at least two arguments (complexity ^{3} 2). The existence of such categories cannot be assumed trivially. Consider sequences like

(231) a/b d b/c c\d → a
a/b b/c c/d d → a

These sequences are designed perfectly, without containing any category of a higher complexity than 1. Is there any reason to believe that these sequences do not or could not occur in natural language? There is nothing wrong in natural-language analysis with a binary branching left-headed tree like (232).

(232)



It might be the format of a simple transitive sentence with object *c* and subject *a/b*, for example. It might even be the dreamt configuration for antisymmetrical syntax with lambda annotation. Moreover, if the complexity of categories were below 2, agenda management at the *to_find*-side would be considerably simplified: this part of the agenda could never increase in complexity – it would be at most 1, by definition. Therefore, it is ill-advised to require the generating syntax to be such that it contains categories of complexity 2 or higher. Rather, we would have the algorithm enriched with a check on the maximal complexity of categories meeting certain conditions, and have these checks abort the present search and force backtracking if the required complexity is no longer available in the lexicon.

Alternatively, we may consider how to avoid the situation covered by (228) g that the *given* agenda contains more than one type. In our experience, this situation occurs when a set of unrelated side-conditions to the generation is processed initially. If one of them cannot be passed to the actual *to_find*-agenda by cancellation, the category associated with the side-condition is by necessity passed on to the *given* agenda, which is already inhabited by the hypothesis, at least. It takes brute force at the control side to avoid such a situation, passing the account of side-conditions on to the *to_find* agenda under the regime of (228)h. To the extent that one succeeds in this task, backtracking and termination for the generation process are enhanced, independently of accidental features of the lexicon and the grammar. By the general design of selection sketched above, randomization and backtracking ensure the completeness of the generation procedure.

Of course, completeness (230) does not imply that every parsable sentence will be generated at some moment by the procedure. It ensures, however, that the procedure can be adopted in an enumeration automaton to this effect. Thus, DELILAH's grammar embodies a generator that mirrors the parser.