A Design Pattern For Phrasal Constructions

Luc Steels

This paper is the authors' draft and has now been officially published as:
Luc Steels (2011). A Design Pattern for Phrasal Constructions. In Luc Steels (Ed.), *Design Patterns in Fluid Construction Grammar*, 71–114. Amsterdam: John Benjamins.

Abstract

This chapter has two objectives. It discusses a design pattern for phrasal constructions and introduces the templates that can be used to instantiate this pattern in Fluid Construction Grammar, using as illustration nominal phrases such as "the green mouse" or "this mouse of mine". Phrasal constructions not only build phrases but also combine the meanings contributed by their constituents and possibly add meaning of their own. Phrasal constructions are interesting because they involve hierarchy, compositionality, recursion, agreement and percolation. The paper also illustrates how FCG uses templates to organise the grammar design process and to simplify the definition of the constructions relevant for a particular language.

1. Introduction

Human natural language grammars are extraordinarily complicated because aspects of meaning and form are tightly packaged to maximise the amount of information that can be transmitted with a minimal amount of signals. In this sense, human languages are like natural living systems in which the same components serve multiple purposes. The components have not been designed and put together in a strictly modular hierarchical fashion, the way one would design a machine, but evolved in a step-wise fashion, exploiting and building further on whatever was already there. The challenge for grammar designers (and for language learners) is to unpack this complexity without losing sight of the full richness of real grammar. Studying how grammars have historically evolved is often instructive because it shows how an additional layer of complexity (for example determiners) can be absent in one stage of the language and then gradually appear and become more complicated.

Fluid Construction Grammar is a general formalism for defining and applying constructions and it can be used to explore different approaches to linguistic theorizing, as long as the notion of a construction is accepted as the fundamental organizing principle. In our own work on grammar design, we have found it useful to start from a clear understanding of the grammar square (shown in figure 1), which is intended to illustrate that lexicons and grammar specify bi-directional relations between meaning and form, in the case of grammar by going through semantic and syntactic categorizations. Different constructions express different aspects of such bi-directional mappings.

The primary purpose of *lexical constructions* is to establish a direct mapping between meaning and form. Some parts of the meaning to be expressed are directly associated with a lexical item in the form of a string. It is therefore already possible to have a purely lexical communication system which consists of a set of individual words that each contribute some meaning to the meaning of the sentence as a whole.

Grammatical constructions come on top of this to serve two purposes:

1. Grammar helps to figure out how the meaning introduced by the different lexical items gets combined. If individual meanings are just introduced as isolated elements, it is up to the listeners to do so. They often can achieve this because human listeners know the context, can apply common sense knowledge, and can keep track of the communicative goals in the ongoing conversation. But there may still be occasions where all of this is not enough to allow the listener to reconstruct with certainty the interpretation the speaker desired or where it may require more cognitive effort than the listener is willing to supply. The speaker can therefore improve the chance of higher communicative success by expressing how meanings are to be combined through grammar.

2. Grammar is also used to express additional aspects of meaning by modulating the forms already supplied by the lexical items and thus package more information with the same materials. For example, many languages feature a grammatical system of tense and aspect to convey the temporal ordering and structuring of events, or a grammatical system of argument structure to express the roles of participants in events. Modulation takes many forms: the ordering of words, adding morphological markers (prefixes and suffixes), changing the basic word form (as from “spring” to “sprang”), using grammatical function words (like auxiliaries), imposing an intonation structure or stress pattern. Different languages make different choices in which aspects of meaning they express grammatically and how they do it.

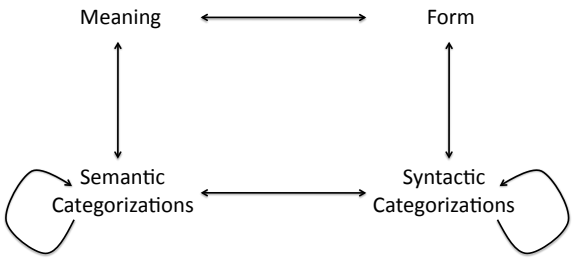

Figure 1. *The grammar square shows the kinds of bi-directional mappings between meaning and form that lexical and grammatical constructions establish. A language is said to be grammatical if these mappings go through the intermediary of syntactic and semantic categorizations.*

Human grammars have two characteristic features: First of all grammar operates through *semantic and syntactic categorizations*. The conceptual meanings are semantically re-categorized to fit with the conceptual patterns that a language has adopted, and these re-categorizations are known to be language specific (Talmy, 2000). For example, the participant in an event (for example the one pushing a block) is semantically re-categorized as agent, and then the grammar specifies how to express more generally the agent role rather than just the role of pusher. Or the temporal moment of an event is categorized with respect to the time of speaking in terms of past/present/future, and the grammar then expresses these temporal distinctions rather than other more fine-grained distinctions that could potentially be adopted (and are adopted in some languages). So semantic categorizations make it possible to achieve more abstract bi-directional mappings between meaning and form.

Syntactic categorizations have the same role, i.e. making grammatical mappings more abstract and therefore more widely applicable. They categorize the surface forms because they are often (but certainly not always) explicitly expressed using morpho-syntactic or phonological devices. Examples of syntactic categorizations are the distinctions between nominative and dative or masculine vs. feminine vs. neuter. How a syntactic categorization is explicitly expressed depends itself on the context, the form of the lexical item being modified, or on other grammatical

mappings, and all this makes it possible to package more information into the same marker than just the expression of a single syntactic category.

Second, grammar operates through *hierarchical structure*. Different lexical items are brought together into larger units, usually called *phrases*, which can then have their own syntactic and semantic categorizations. They in turn function as units which can again be combined with other ones to form new phrases. The meaning and properties of a phrase are based on combining the meanings and properties of the individual elements but a phrase can be more than the sum of its parts. For example, a nominal phrase like "the green mouse"[1] combines an article "the", an adjective "green", and a noun "mouse". The phrase as a whole can appear as such in different parts of the sentence, as in "I read a paper about the green mouse" or "The green mouse escaped the genetic engineering laboratory." It is possible that a phrase of a certain type appears as a constituent of a phrase of the same type, in which case we talk about *recursive hierarchical structure*. For example, the nominal phrase "the green mouse from the Japanese genetic engineering lab" has a nominal phrase "the Japanese genetic engineering lab" as one of its constituents.

In the design of a grammar, we want to treat different grammatical systems separately and introduce separate constructions to handle them, because that makes it more doable to cope with the complexity of human language. For example, it is desirable that there are separate constructions which focus on the expression of tense or aspect, others focus on argument structure, on information structure, etc. But of course in the final sentence everything has to be brought together again and it may therefore not always be so easy to tease apart different grammatical systems, particularly if they interact strongly. Moreover, language users most probably store recurring expressions and patterns as single complex constructions so that they can retrieve and apply them very fast. In that case, a division into different grammatical systems is no longer explicitly present and the stored pattern may start to behave as a construction it its own right, which is only partly influenced by the governing grammatical systems. This is indeed what happens when phrases become idiomatic and the meaning is no longer derivable in a compositional way from the meaning of the parts.

Phrase structures are generally regarded as forming the backbone skeleton of sentences and other grammatical systems (such as argument structure, aspect, modality, negation, etc.) operate over them, in the sense that they may add new constituents in phrases or modulate the form of some of their parts. Phrase struc-

1. This example is inspired by efforts in genetic engineering to make colored mammals, including a green fluorescent mouse (Masahito et al., 1999)

tures are therefore comparable to the basic architectural frame of a house, whereas other grammatical systems add more elements or embellish this frame to add more function and esthetic quality to the house. That is why we are going to study phrase structure first in this chapter, and later contributions in this book focus on how other grammatical systems then enhance or modulate this backbone or its various components, illustrated in particular for constructions dealing argument structure (van Trijp, 2011a).

2. Functional and Constituent Structure

There are two main traditions in linguistic investigations of phrase structure. The first tradition focuses on function. It considers that units are grouped together in a phrase because they play a certain role in that phrase. For example, "the" and "mouse" combine into the phrase "the mouse" because "the" is a determiner of "mouse". This perspective is explored in functional grammar (see for example (Dik, 1978; Siewierska, 1991), relational grammar (Perlmutter, 1983), and dependency grammar (See for example Mel'cuk (1988), Anderson (1971), Sgall & Panevova (1989)). The second tradition focuses on syntactic types (parts of speech or lexical categories for individual words, such as Noun, Verb, Article), and phrase types (such as Noun Phrase, Verb Phrase and the like). This perspective argues that units are grouped together because they belong to certain syntactic types. For example, "the cat" is a phrase because "the" is an article and "cat" is a "noun" and an article and a noun combine, by definition, into a noun phrase. This tradition has its roots in structural linguistics, particularly Bloomfieldian immediate constituent analysis (Bloomfield, 1993) and the Chomskyan generative tradition that starts with (Chomsky, 1957).

In this chapter, we will study a design pattern for handling phrasal structures that integrates both perspectives. The hierarchy itself (with units and subunits) is represented independently of whether it is based on functional or syntactic considerations. Phrases combine units if they satisfy various syntactic criteria, possibly including the syntactic type of the unit, and they impose specific functions on each unit. For example, a noun can function as the nominal of a phrase (which is therefore called a nominal phrase), as in "the mouse", and a nominal phrase can function as the subject of a sentence, as in "the mouse escapes".

In what follows, the term for denoting a phrase is chosen based on the syntactic function of the main constituent (usually called *head*) of a phrase. For example, the term *nominal phrase* designates a phrase that has as its main constituent a unit with the syntactic function nominal. In constituent structure traditions the same phrase-

type would be designated through the lexical category of the head of the phrase, as in *noun phrase*. Both terms are equivalent.

What makes grammar complicated is that there is not a one-to-one relation between syntactic type and syntactic function, in the sense that the same function can usually be achieved by many different syntactic types. For example, a nominal can also consist of an adjectival phrase and a noun as in "the green mouse" or a noun combined with another noun as in "the beach ball". Moreover the same syntactic type can have multiple syntactic functions. For example, a nominal phrase can act as the subject of a sentence but also as a direct object or indirect object, and even as a predicate, as in the phrase:

> "The mouse caught yesterday in a Roppongi night club is a green mouse of the kind that escaped last year from a genetic engineering lab."

The same lexical item can have multiple syntactic types and hence functions. For example, the word "left" can have an adjectival function in a nominal phrase in which case its syntactic type is adjective, as in "the left block", but it can also have an adverbial function, as in "she turned left after the traffic light", in which case its syntactic type is that of an adverb, modifying the verb.

Construction grammar strives for a tight integration of syntax and semantics. On the semantic side, we indeed find that syntactic functions are mirrored by semantic functions (although there is certainly not a simple one-to-one correspondence). Indeed one of the primary roles of syntactic functions is precisely that they help decide which semantic function a particular unit has. For example an adjectival function points to the semantic function of qualifier, an adverbial function to that of modifier, a determiner introduces the access-function (usually called reference) for the class of objects identified by the noun.

Semantic functions are denoted by terms like identifier, referring expression, qualifier, modifier, etc. and they have been studied most intensively by cognitive linguists (as for example (Langacker, 2008)) and by formal semanticists (as for example (Partee, 2003)). Which semantic function is chosen for specific meanings depends partly on the communicative goals of the speaker, but also on what lexical items are chosen to cover these meanings and on constraints imposed by the rest of the semantic and syntactic context. At the same time, the semantic functions that need to be expressed impose constraints on syntactic functions, constituent structure, and lexical items.

Unfortunately, the linguistic terminology for denoting syntactic types or syntactic and semantic functions is not standardised across the field. This fact can be

problematic for the novice, but in defense of the linguist it is hardly possible to standardize the terminology insofar as all linguistics categorizations appear language-specific and have a prototypical nature (Croft, 2001; Haspelmath, 2007). This is why the terminology of syntactic and semantic categories is not fixed in FCG. It is at the discretion of the grammar designer. At the same time, of course all terms need to be used consistently within the same grammar.

Given the representational power of feature structures, it is not difficult to represent notions of functional and constituent structure using feature structures. The hierarchy of units and subunits is represented using the subunits feature on the semantic and syntactic side (which makes it in principle possible to have a different structure on each side) and syntactic and semantic functions as well as syntactic types are represented as categorizations associated with units. For example, the unit for "green" in the phrase "the green mouse" would include in its `syn-cat` feature value information that its `lex-cat` is `adjective` and its `syn-function` `adjectival` and its `sem-cat` feature value would include the information that its `sem-function` is `qualifier`.

Defining constructions in such a way that they can support the linguistic decision-making process both for parsing and for producing phrase structures is a more complex matter, because decisions at different levels influence each other and often decisions need to be left open until more information becomes available. Three different types of constructions contribute to the decision-making process, and there is a template for each: `def-lex-cxn` builds lexical constructions, `def-fun-cxn` builds functional constructions, and `def-phrasal-cxn` builds phrasal constructions.

1. In addition to specifying the lexical stem and meaning of a word (more precisely a lexical stem), *lexical constructions* can contribute information on the lexical category and on semantic categorizations that are helpful to decide which semantic function can be realized by this item. For example, the lexical construction for "mouse" would include information that this word is a noun and that its meaning designates a class of objects. Lexical constructions are therefore defined using two templates (as explained already earlier in this book, (Steels, 2011b)). A template called `def-lex-skeleton` introduces the meaning and the string, and a template called `def-lex-cat` introduces the semantic and syntactic categorizations.

2. *Functional constructions* specify (potential) syntactic and semantic functions based on the properties of lexical items. For example, there could be a construction specifying that a noun can be the nominal (head) of a nominal phrase in which case they introduce the identifier of a referring expression. Functional constructions are a

specific case of *categorizing constructions* that introduce new semantic or syntactic categorizations given existing ones. Functional constructions are built with a template called `def-fun-skeleton` which defines the syntactic and semantic functions that go together and the phrase-types that are required.

3. *Phrasal constructions* do the horsework for combining different units (whether lexical items or phrases) into phrasal units. The core objective of a phrasal construction is to take care of syntactic compositionality. Given units with specific syntactic and semantic properties, the construction should define how they can be combined into a unit with a new phrase-type and new semantic function. This is defined using a template called `def-phrasal-skeleton`. But phrasal constructions must typically do a lot more:

1. They must specify also how the meaning of the unit as a whole is built based on the meanings of the individual components by detailing the *linking* of the arguments. This is done with a template called `def-phrasal-linking`. Argument linking can become quite complicated and versatile not only in the case of sentences but also adjectival or nominal phrases. In the case of sentences, linking issues are usually lifted out of the phrasal construction and delegated to a separate set of constructions, called `argument-structure constructions`. How FCG deals with argument-structure constructions is discussed in detail in a later chapter of this (van Trijp, 2011a).

2. Construction grammarians argue that the constructions that build new phrases can also add *new meaning* to a phrase, which is more than the sum of the meanings provided by individual constitutents. Phrasal constructions therefore use another template called `def-phrasal-require` to achieve this function. This template can impose new meanings but also new semantic categorizations to a phrase that may be relevant for triggering other constructions.

3. And then there are of course the specific *form constraints* that the construction has to look out for or impose. For example, there are typically ordering relations between the different units, or there may be a particular stress pattern imposed on the units. These are defined in the template called `def-phrasal-require`.

4. Human languages not only use word order but also syntactic and semantic *agreement* relations to constrain which constituents can be combined. For example, an article and a noun have to agree for number to be considered part of

the same nominal phrase in English. "a" can be combined with "house" forming "a house" because both of them are singular, whereas "a houses" would be ungrammatical. Moreover often a phrase adopts some of the syntactic or semantic categorizations of one of its constituents, a phenomenon known as *percolation*. For example, definiteness percolates from the determiner to its nominal phrase or number percolates from the noun to the nominal phrase as a whole. The agreement and percolation imposed by a particular phrasal construction are defined using a template called `def-phrasal-agreement`.

The remainder of this chapter provides more detail on these various templates and how the constructions they build interact with each other. The chapter has a tutorial character and sticks to simplified examples, not worrying yet about the fact that usually the same linguistic element can have multiple uses. A companion paper (Bleys et al., 2011) illustrates in more detail how the different types of constructions introduced here apply and it provides more information on the underlying search process. Later chapters introduce not only more complex examples of phrasal constructions but also techniques to deal with more sophisticated agreement and percolation phenomena based on feature matrices (van Trijp, 2011b) and with open-ended choices about syntactic and semantic functions by using potentials (Spranger & Loetzsch, 2011). This chapter assumes that the reader has had an introduction to FCG by studying the previous chapter of this book "A First Encounter with FCG" (Steels, 2011a). Occasionally the full expansion of a template is provided, which the computational linguist might find relevant to understand precisely how a construction is operationalized in FCG. However these expansions need not be understood in detail by the reader who just wants to grasp the general approach to phrase structure commonly practiced in FCG.

The next section introduces first how meaning and form are going to be represented, with subsequent sections examining lexical constructions, functional constructions, and phrasal constructions.

3. Representing meaning and form

Language users must be able to map meanings to forms in speaking and forms to meanings in comprehension. Designing a lexicon and grammar that captures the knowledge needed to achieve these mappings for a particular fragment of language therefore starts by considering how meanings and forms are represented.

3.1. Representing meaning

Fluid Construction Grammar is not dogmatic about what approach is used for representing the meaning of utterances. Some researchers use variants of first order predicate calculus, others use frame semantics (Micelli et al., 2009), and still others use grounded procedural semantics (Spranger & Loetzsch, 2011). In this paper I will use predicate calculus expressions, which can be interpreted against a fact base as in PROLOG style semantics. In what follows, the domain of discourse consists of *individuals* (such as 'a mouse'), which is denoted by unique names consisting of a symbol and an index, as in `mouse-1`, `mouse-2`, ..., etc., and *sets of individuals*, such as { `green-mouse-1`, `green-mouse-2`, ... } also denoted by unique names consisting of a symbol and an index, as in `green-mice-1`, or `light-green-mice-45`. The indices have no meaning, except to distinguish between different instances.

Predicates are semantically treated as relations between sets. A predicate relates a source set of elements to produce a target set containing the elements in the source set that satisfy a particular condition. For example, the predicate `mouse` determines how far the elements in a source-set satisfy the image schema of a mouse. We use prefix-notation to represent a primitive fact, consisting of a predicate and its arguments, as in:

```
(mouse mice-33 source-set-67)
```

The indices have no particular meaning. `source-set-67` is a set provided by further context. `mice-33` is a set of mice within that source-set. The source-set is provided as an important part of a predicate. For example, a box could be called "blue" in the context of green boxes because it is the box that is most blue, but the same box could be called "green" when it is the greenest box in the context of other blue boxes.

Predicates may also have individuals as arguments. For example, a predicate may occasionally pick-out a single member from a set, typically the best representative at that point. The operation of picking out the referent can be postponed until enough information is available to do it. For example, in the sentence "the child that was brought to school by her mother yesterday", the referent of the child can only be computed when the meaning of "yesterday" has become available.

Finally, every element (individuals, predicates, sets) can be bound to a variable, which consists of a question mark followed by a symbol and possibly an index as in: `?set-1`, `?individual-5`, `?predicate-24`, ... As before the names of variables have only meaning for us and are chosen to be as clear as possible, they have no function in the system itself. We could just as well have used `?s-1`, `?i-5`, `?p-24`.

Given that sentences may involve dozens of variables it is clear that unmotivated names would make it much harder to follow what is going on.

Using this approach, the meaning of a phrase like "the green mouse" can be expressed with the following list of predicate-argument expressions representing a conjunction of facts. The names of the predicates or variables is chosen entirely for this occasion:

```
((context ?context)
    ; ?context is the set of all objects in the context
(mouse ?mouse-set ?context)
    ; ?mouse-set represents all elements in the context which are mice
(green ?green-mouse-set ?mouse-set)
    ; ?green-mouse-set represents the subset of mice that are green
(unique-definite ?the-mouse ?green-mouse-set))
    ; ?the-mouse gets bound to the unique individual that
    ; remains in the singleton set ?green-mouse-set.
```

The aim of this paper is to design a lexicon and grammar that maps such meanings to phrases and vice-versa. I use English-like nominal phrases, although there is no effort made to come close to a complete coverage.

3.2. Representing form

Similar to other unification-based formalisms, the form of an utterance is represented in FCG using a set of predicates that define constraints on what the utterance should look like. This has numerous advantages, not only because every relevant form aspect can be taken into account, but also because constraints on form can be gradually assembled by many different constructions without having to go through complex manipulations of tree structures. The form constraints are translated in a concrete utterance by the renderer or reconstructed from an utterance by the de-renderer. It is perfectly possible that the constraints are incomplete, in which case the renderer makes random decisions. For example, if the ordering is not fully specified, some of the words may appear anywhere in the utterance. Issues of morphology, intonation, etc. are not addressed in this paper, so only two form-predicates are needed: `string` and `meets`, as illustrated in the following examples:

```
(string unit-1 "green")
```

states that a particular unit (here `unit-1`) covers the string "green".

```
(meets unit-a unit-b)
```

states that the relation between two units, in this case `unit-a` and `unit-b`, requires the second unit immediately follows the first unit of the utterance.

Using these predicates, the form of the utterance "the green mouse" is described with the following set of predicate-argument expressions:

```
((string the-unit "the")
 (string green-unit "green")
 (string mouse-unit "mouse")
 (meets the-unit unit-green)
 (meets green-unit unit-mouse))
```

The names of the units are of course arbitrary chosen here. A set of constructions can now be developed that is able to map from the kind of meanings shown in section 3.1. to ordered sequences of strings, and back. Each step progressively makes the grammar or lexicon more complex in order to handle the various challenges discussed earlier.

4. Lexical constructions

The natural way to start is with lexical constructions. We have already seen examples of these in an earlier chapter (Steels, 2011). A lexical construction is built with the template `def-lex-cxn` that introduces a name for the lexical construction and then evokes a set of subtemplates to progressively build the construction as a whole. The first subtemplate simply defines the meaning, i.e. a set of predicate-argument expressions, and the word-string. So the definition of a construction using a template would typically take the following form:

```
(def-lex-cxn cx-name
  (def-lex-skeleton cx-name
    :meaning ...
    :string ...)
  )
```

The `def-lex-skeleton` template has a slot for the meaning called `:meaning` and a slot for the string called `:string`. Here is an example to define the lexical construction for "mouse" using the `def-lex-cxn` template:

```
(def-lex-cxn mouse-cxn
  (def-lex-skeleton mouse-cxn
    :meaning (== (mouse ?mouse-set ?base-set))
    :string "mouse"))
```

The filler of the `:meaning` slot specifies that the meaning covered includes, i.e. `==`, the predicate-argument expression `(mouse ?mouse-set ?base-set)`. The filler of the `:string` slot specifies the word string "mouse".

For the interested reader, I give here the equivalent operational construction when the template is expanded. The elements that are specified as fillers of slots in the template are in bold, the rest is filled in by the template itself.

```
(def-cxn  mouse-cxn ()
  ((?top-unit
    (tag ?meaning
         (meaning  (== (mouse ?mouse-set ?base-set))))
    (footprints (==0 mouse-cxn)))
   ((J ?word-mouse ?top-unit)
    ?meaning
    (footprints (==1 mouse-cxn))))
  <-->
 ((?top-unit
    (tag ?form
         (form (== (string ?word-mouse  "mouse"))))
    (footprints (==0 mouse-cxn)))
  ((J ?word-mouse ?top-unit)
    ?form
    (footprints (==1 mouse-cxn)))))
```

It is important to note how the covered meaning and the covered form are tagged and moved to a new unit bound to `?word-mouse`. This unit is created by the J-operator, and from then on it functions as the representation of the word "mouse". A footprint `mouse-cxn` is added and tested so that the construction does not keep applying indefinitely. Footprints are by convention equal to the name of the construction that adds the footprint.

Here is the definition of another lexical construction for "the" using the `def-lex-cxn` template:

```
(def-lex-cxn the-cxn
  (def-lex-skeleton the-cxn
    :meaning (== (unique-definite ?indiv ?base-set))
    :string "the"))
```

and one for "green":

```
(def-lex-cxn green-cxn
  (def-lex-skeleton green-cxn
    :meaning (== (green ?green-set ?base-set))
    :string "green")))
```

When these various lexical constructions are applied to “the green mouse” we obtain the transient structure shown in Figure 2. A similar transient structure would

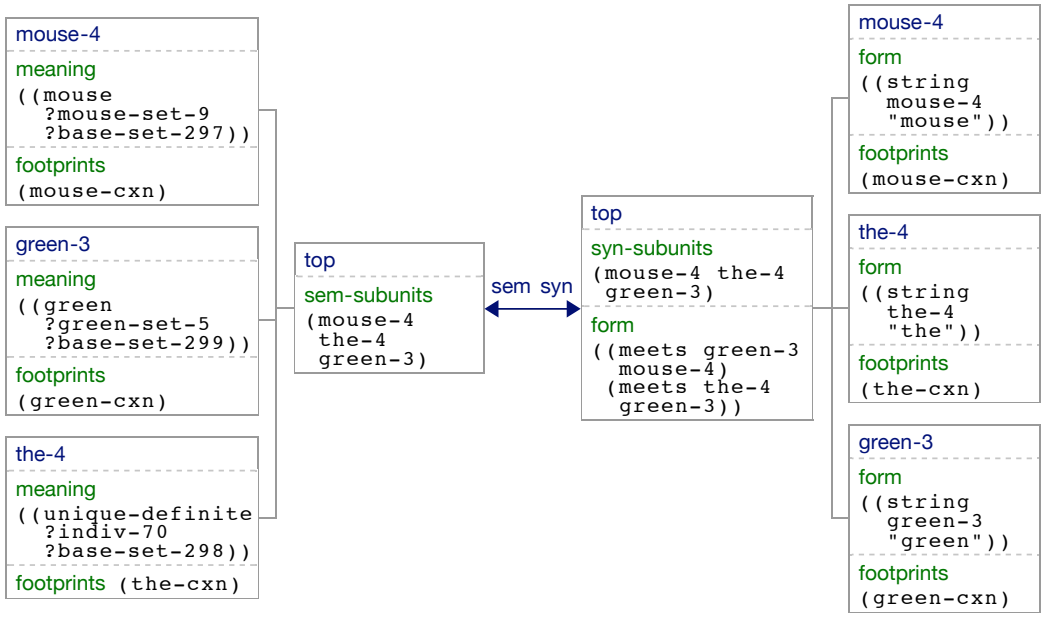

Figure 2. *Applying only the lexical constructions gets us already quite some distance in parsing and producing utterances. This figure shows the result of parsing “the green mouse”.*

be built in production given as input the following meaning:

```
((unique-definite indiv-mouse-1 green-1)
 (green green-1 mouse-1)
 (mouse mouse-1 context-1)
 (context context-1))
```

The utterance being produced would be “mouse the green” or “green the mouse”, or some other permutation, because the transient structure does not contain any ordering constraints. This illustrates that a set of lexical constructions is in

itself already enough to achieve some form of communication even though it relies entirely on the listener to connect the different meanings.

How can the job of communication be done better? Let us look first at the outcome of parsing. The meaning assembled by taking the union of the meanings of all units in the transient structure in Figure 2 looks like this:

```
((unique-definite ?indiv-70 ?base-set-298)
 (green ?green-set-5 ?base-set-299)
 (mouse ?mouse-set-9 ?base-set-297))
```

This can be paraphrased as: There is a unique individual member `?indiv-70` out of a base-set `?base-set-298`. There is a green set of things `?green-set-5` all belonging to the set `?base-set-299`, and there is a set of mice `?mouse-set-9` all belonging to the set `?base-set-297`. These formulations are all correct but incomplete. The linking between the base-set used for picking an individual, the set of green objects and the set of mice is not stated explicitly. Hence, the individual can be a member of any set and the set of green things is not necessarily a subset of the set of mice. Moreover the general context of this phrase, i.e. `?base-set-297`, is not grounded to the current context. We therefore need additional (grammatical) constructions to combine the meanings provided by the lexicon.

The job of production is also not yet complete. The sentence produced here consists of the right set of words, but there is no ordering specified among the words. Consequently the renderer might happen to produce the correct order, but it might just as well render the words in another order. We need grammar to fix this, and this grammar needs to be compositional. We should be able to do “the mouse” but also “the green mouse” or “the very green mouse” or “the slightly blue green mouse”.

Before developing the necessary grammatical constructions to achieve this, it is worthwhile to point to the advantages of using only lexical constructions as the first step both in parsing and producing. Despite the meanings obtained from lexical parsing being incomplete, nevertheless an interpretation process that has access to a world model would in fact already be able to come up with a very plausible interpretation. This possibility is good news for achieving robust parsing or for building learning systems that may not yet have acquired complete grammar. Furthermore, even if some aspects of form are missing, such as correct word order or morphology, a purely lexicon-based production process might already be able to produce sentences fragments that are interpretable by a human listener, who might then correct the sentence and thus generate a learning opportunity for the speaker.

5. Functional constructions

Next we need constructions whose primary role is to decide on the syntactic and semantic functions of the lexical items. Therefore, they are called *functional constructions*. For example, a noun can have the syntactic function of nominal, and, if it has this function, its semantic function is to identify the class of objects that is used in a referring phrase. This information gets packaged into a functional construction that associates syntactic types (like noun) with syntactic functions (such as nominal) and at the same time associates semantic categorizations (like introducing a class of objects) with semantic functions (such as identifier). This step is necessary because the same syntactic or semantic types can be used in many different functions, or they can even be coerced into functions that are not yet conventionalized. These bi-directional mappings come as a complete package. When one of these mappings is blocked for one reason or another then the other relation is blocked as well. For example, if the semantic function is identifier because a word introduces a class of objects, but the word itself does not belong to the lexical class of nouns, then the whole mapping is blocked.

To define these relations, we will use a template called `def-fun-cxn` that has only one skeletal template `def-fun-skeleton`. It has a slot `:sem-cat` for the relevant semantic categorizations, a slot `:sem-function` for the semantic function, a slot `:syn-cat` for the relevant syntactic categorizations and a slot `:syn-function` for the relevant syntactic function. The syntactic and semantic categorizations can be as complex as is necessary, and the template may include other slots, but for now we only need these ones. Below is an example of the use of this template:

```
(def-fun-cxn noun-nominal-cxn
  (def-fun-skeleton noun-nominal-cxn
   :sem-cat (==1 (class object))
   :sem-function identifier
   :syn-cat (==1 (lex-cat noun))
   :syn-function nominal))
```

This construction relates the semantic category class of objects with the semantic function `identifier` and the part of speech noun `(lex-cat noun)` with the syntactic function `nominal`. Syntactic and semantic aspects are always considered at the same time, and the mapping is blocked when either side shows a conflict. For example, even though a unit identifies a class of objects, it may not be categorized as a noun if the part of speech of the word being used is not a noun.

For the interested reader, the expansion of the noun-nominal-cxn definition using templates into an operational definition is given below. The elements that have been explicitly supplied by the template are in bold. Everything else is automatically added by the template itself:

```
(def-cxn noun-nominal-cxn ()
  ((?top-unit
    (sem-subunits (== ?nominal-unit)))
   (?nominal-unit
    (footprints (==0 noun-nominal-cxn))
    (sem-cat (==1 (class object))))
   ((J ?nominal-unit)
    (sem-cat
     (==1 (sem-function identifier)))
    (footprints (==1 noun-nominal-cxn))))
  <->
  ((?top-unit (syn-subunits (== ?nominal-unit)))
   (?nominal-unit
    (footprints (==0 noun-nominal-cxn))
    (syn-cat (==1 (lex-cat noun))))
   ((J ?nominal-unit)
    (syn-cat (==1 (syn-function nominal)))
    (footprints (==1 noun-nominal-cxn)))))
```

We see that this functional construction does not create a new unit of its own but uses the J-operator to add more information to an existing unit (bound to `?unit-name`). The footprint `noun-nominal-cxn` is added to avoid infinite application of this construction and to trace its use. Strictly speaking, we do not need this footprint because the construction could also test whether the change it wants to make has already been made and, if so, block another application on the same unit. At the same time, uniform usage of footprints in all constructions avoids errors.

Here is another example of a functional construction that relates articles to determiners with the semantic function reference:

```
(def-fun-cxn article-determiner-cxn
  (def-fun-skeleton article-determiner-cxn
   :sem-cat (==1 (determination ?definiteness))
   :sem-function reference
   :syn-cat (==1 (lex-cat article))
   :syn-function determiner))
```

Before these functional constructions can be used, we need to extend lexical constructions with the necessary syntactic and semantic categorizations. This extension can be done easily by the `def-lex-cat` template that adds semantic and syntactic categorizations (`:sem-cat` and `:syn-cat`) to the skeleton of a lexical construction, so that for the noun " mouse" we get the following definition:

```
(def-lex-cxn mouse-cxn
  (def-lex-skeleton
    :meaning (== (mouse ?mouse-set ?base-set))
    :string "mouse")
  (def-lex-cat
    :sem-cat (==1 (class object))
    :syn-cat (==1 (lex-cat noun))))
```

For the article " the" we get:

```
(def-lex-cxn the-cxn
  (def-lex-skeleton
   :meaning (== (unique-definite ?indiv ?context))
   :string "the")
  (def-lex-cat
   :syn-cat (==1 (lex-cat article))
   :sem-cat (==1 (determination definite))))
```

When both of these lexical and functional constructions are available, and we give "mouse" as input, i.e. the value of the form feature in the top is `(string mouse-6 ''mouse'')`, we get the structure shown in Figure 3. A similar structure is obtained if we would start a production process with a target meaning like `((mouse mouse-set-14 base-set-5))`

6. Phrasal constructions

A phrasal construction is developed in steps, just as the lexical construction discussed earlier. Each step is captured in a template that operates on the result of the previous step. The first step is to combine constituents based on their syntactic and semantic functions by the `def-phrasal-skeleton` template, then to handle agreement and percolation (`def-phrasal-agreement`), next to achieve argument linking (`def-phrasal-linking`), and finally to specify how the construction adds

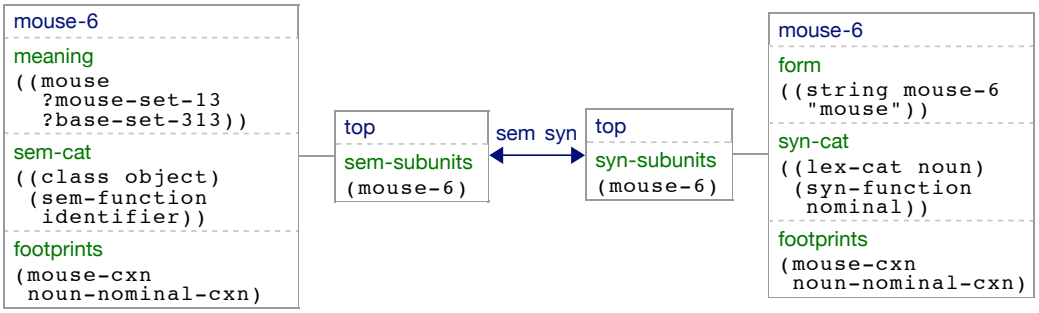

Figure 3. *Transient structure obtained after applying a construction that adds the syntactic function nominal and the semantic function identifier to words characterized as nouns.*

its own additional constructional meaning or form using (`def-phrasal-require`). The order in which these templates operate is irrelevant but the skeleton has to come first because it introduces the different constituents involved in the construction. The `determiner-nominal-phrase` construction serves as a primary example, with more complex examples provided in the next section. The determiner-nominal-phrase construction combines a nominal and a determiner to build a nominal phrase, such as "the mouse".

All phrasal constructions are grouped with a template `def-phrasal-cxn`, which does not much more than group all the different steps in building an operational construction. It is of the form

```
(def-phrasal-cxn cxn-name
  (def-phrasal-skeleton cxn-name
       ...)
  ...)
```

6.1. Combining constituents

We begin by focusing on compositionality handled by the `def-phrasal-skeleton` template. For the determiner-nominal construction, this means the following:

1. On the semantic side, the construction requires a unit `?determiner-unit` with the semantic function of `reference` and a unit `?nominal-unit` with the function `identifier`, and it then constructs a new unit `?nominal-phrase` with as semantic function `referring`.

2. On the syntactic side, the construction requires a unit `?determiner-unit` with the syntactic function `determiner` and a unit `?nominal-unit` with the syntactic function `nominal`, and it should construct a new unit `?nominal-phrase` categorized syntactically as a `nominal-phrase`.

This suggests the beginning of a template for phrasal constructions, which is called `def-phrasal-skeleton`. It has two slots: one is called `:phrase` for defining semantic and syntactic categorizations of the new phrase, and one is called `:constituents` for defining the various constituents, where each constituent is defined in terms of what semantic and syntactic function to expect and possibly a phrasal type. The template introduces variables for the different units involved. These variables provide motivated names when symbols for units need to be created in production or parsing and, more importantly, they make it possible later to formulate additional parts of the construction. The template handles any number of possible constituents, and the order in which they are defined does not play a role.

Here is an example of the use of this template for defining the `determiner-nominal-phrase` construction:

```
(def-phrasal-skeleton determiner-nominal-phrase-cxn
  :phrase
  (?nominal-phrase
    :sem-function referring
    :phrase-type nominal-phrase)
  :constituents
  ((?determiner-unit
     :sem-function reference
     :syn-function determiner)
   (?nominal-unit
     :sem-function identifier
     :syn-function nominal)))
```

The nominal phrase requires a determiner and a nominal, which are here only defined in terms of what syntactic and semantic functions they should have. The

phrase as a whole is given a semantic function (referring) and a phrase type (nominal-phrase).

This definition is not to be confused with a generative rewrite rule of the sort

```
NounPhrase -->  Article Noun
```

First of all the construction operates on the basis of syntactic and semantic functions, rather than syntactic types although we could have added more constraints on the syntactic type of constituents as well. More importantly, the determiner-nominal-phrase construction *combines* a determiner and a nominal into a nominal phrase both in parsing *and* in production. The constituents have to be there already from the application of earlier constructions and they have to satisfy all the constraints defined here in order to be combined. It is never the case that the nominal phrase exists as a unit and is then 'rewritten' with two new constituents. FCG is not designed to support generation (as in generative grammar). Instead it focuses on production, in the sense of mapping meaning to form, and parsing.

For the interested reader, I provide here the operational construction for the example given. The information provided explicitly is in bold. All the rest is added by the template itself:

```
(def-cxn determiner-nominal-cxn ()
  ((?top-unit
    (footprints (==0 determiner-nominal-cxn))
    (sem-subunits (== ?determiner-unit ?nominal-unit)))
   (?determiner-unit
    (sem-cat (==1 (sem-function reference))))
   (?nominal-unit
    (sem-cat (==1 (sem-function identifier))))
   ((J ?nominal-phrase ?top-unit
     (?determiner-unit ?nominal-unit))
    (sem-cat (==1 (sem-function referring)))
    (footprints (==1 determiner-nominal-cxn)))))
```

⟷

```
 ((?top-unit
    (footprints (==0 determiner-nominal-cxn))
    (syn-subunits (== ?determiner-unit ?nominal-unit)))
  (?determiner-unit
   (syn-cat (==1 (syn-function determiner))))
  (?nominal-unit
```

```
   (syn-cat (==1 (syn-function nominal))))
  ((J ?nominal-phrase ?top-unit
      (?determiner-unit ?nominal-unit))
   (syn-cat (==1 (phrase-type nominal-phrase)))
   (footprints (==1 determiner-nominal-cxn)))))
```

The sem- and syn-functions appear in the sem-cat and syn-cat unit-features of the relevant constituents. The J-operator creates the nominal phrase both on the semantic and syntactic side with the determiner and nominal units as subunits. It adds information about the phrase as a whole (the phrase-type on the syntactic side and the sem-function on the syntactic side). There are footprints added on both sides to control and report the application of the construction and these footprints are tested in the `?top-unit`.

Using this phrasal construction and the lexical and categorization constructions given earlier, we obtain the semantic and syntactic pole of the transient structure shown respectively in Figure 5 and Figure 4 after parsing "the mouse".

An entirely similar structure is obtained when we start a production process with the input meaning:

```
((unique-definite individual-1 mouse-1)
 (mouse mouse-1 context-1))
```

6.2. Agreement and percolation

The `phrasal-skeleton` template is a good start, but phrasal constructions must do a lot more. Let us focus first on handling agreement and percolation. Agreement means that certain syntactic or semantic features of one unit are shared with that of another unit. For example, determiner and nominal have to agree with respect to number. Interestingly, agreement often flows in both directions. It is possible that we do not know the number of the determiner for sure (as in the case of "the") and then the nominal determines it (as in "the mouse") or it may be that we do not know the number of the nominal for sure (as in the case of "sheep") and then the determiner might determine it (as in "a sheep"). It is even possible that neither the determiner nor the nominal allow a decision to be made, as would be the case for "the sheep". It is only in a larger context that number can be decided, as in "the sheep is eating grass" where the auxiliary "is" has to agree in number with the subject.

All this suggests that we should think about agreement in terms of constraint propagation. It is handled in unification-based grammars by using variables for the

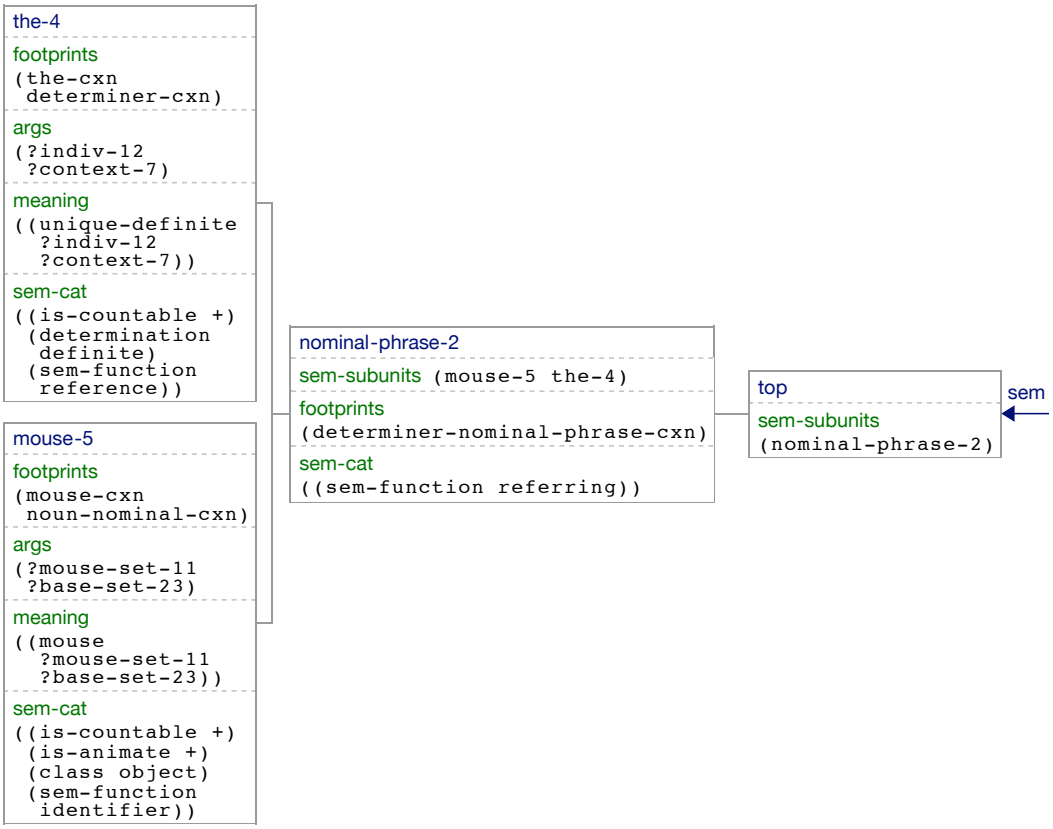

Figure 4. *The semantic pole of the transient structure obtained after applying the determinal-nominal-phrase construction when parsing "the mouse". A unit for the nominal phrase has been constructed with the semantic function referring.*

relevant syntactic or semantic features. They get bound in one place, and then used in other places. It is not necessary to specify where the variables get bound and where they are used. This approach is used also in FCG. (See a later chapter by van Trijp, 2011b, for a more sophisticated way of dealing with agreement.)

Percolation means that a newly constructed phrase obtains some of its syntactic or semantic features from its constituents. For example, `definiteness` is usually expressed in English with the determiner ("the" versus "a"), but then it becomes a property of the nominal phrase as a whole, as with "the table" which is considered

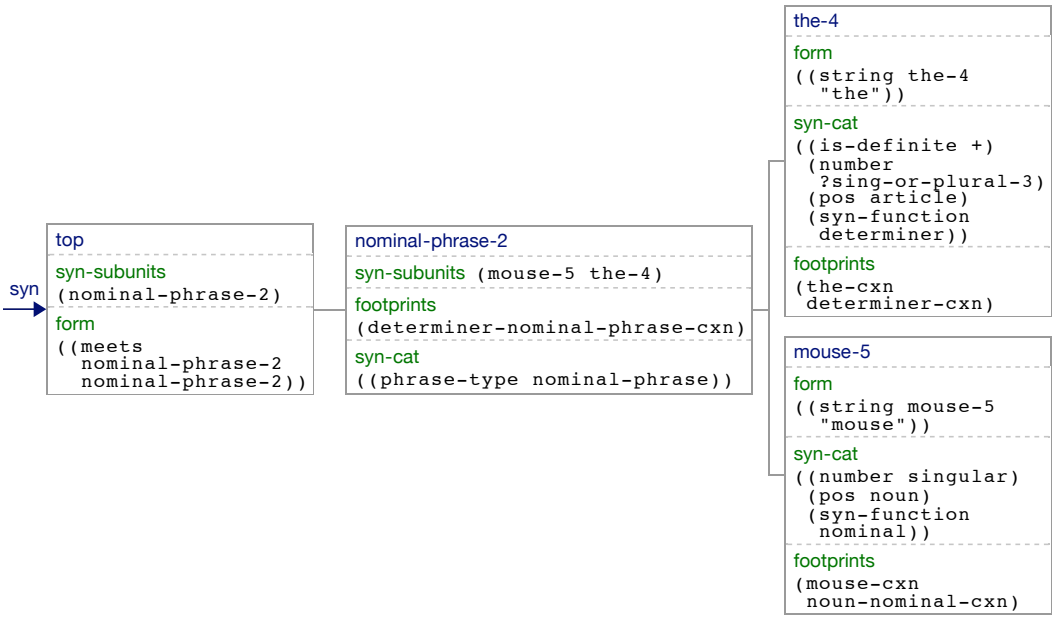

Figure 5. *Syntactic pole of the transient structure obtained after applying the determiner-nominal-phrase construction. The new phrase has the phrase-type nominal-phrase.*

a definite nominal phrase. Percolation is computationally the same as agreement. Both processes establish which features are shared between units, whether this is horizontal (between constituents in agreement) or vertical (between phrases and their constituents in percolation). They are therefore handled with the same template called `def-phrasal-agreement`.

The `def-phrasal-agreement` takes as arguments the name of the construction it is building and a list of agreement specifications for each of the units in the phrasal skeleton, where each specification has one slot for which syntactic features have to agree (called `:syn-cat`) and one for which semantic features have to agree (called `:sem-cat`). The general structure is therefore as follows:

```
(def-phrasal-agreement cxn-name
  ( unit-variable
   :syn-cat features
   :sem-cat features))
  ... )
```

The unit-variables used to specify which unit we are talking about are those that were used to define the original phrasal skeleton in the first place. The use of this template is illustrated in the following example which builds further on the skeleton of `determiner-nominal-phrase-cxn` defined earlier:

```
(def-phrasal-agreement determiner-nominal-phrase-cxn
  (?nominal-phrase
    :syn-cat
      (==1 (is-definite ?definiteness)
           (number ?number)))
  (?determiner-unit
     :sem-cat
       (==1 (is-countable ?countable))
     :syn-cat
       (==1 (is-definite ?definiteness)
            (number ?number)))
   (?nominal-unit
     :sem-cat
       (==1 (is-countable ?countable))
     :syn-cat
       (==1 (number ?number))))
```

`Number` appears in all units, ensuring that there is agreement for number between determiner and nominal and that number percolates to the nominal phrase as a whole. `is-definite` percolates up from the `?determiner-unit` to the `?nominal-phrase` but is not mentioned with the nominal because (in English) this information is not marked or associated with nouns. The determiner-nominal also shows an example of semantic agreement for the feature `is-countable` between determiner and nominal. The phrase “a milk” is ungrammatical because “milk” is a mass noun, i.e. uncountable, and therefore cannot be combined with the article “a” which signals countability. (Although we get some unusual exceptions when milk is coerced into meaning “a bottle of milk”, as in “I ordered a milk”.)

In regards to the operational definition, below is the worked out determiner-nominal-cxn after these agreement constraints have been added. The additions are shown in bold. The features that percolate are found in the J-units and the ones that have to agree are found in the other units. Using the same variable (e.g. ?countable) in the determiner-unit and the nominal-unit ensures that the variables are either unknown but from now on their values will be considered equal or they have bindings and then the bindings have to be equal.

```
(def-cxn determiner-nominal-phrase-cxn ()
 ((?top-unit
    (footprints (==0 determiner-nominal-cxn))
    (sem-subunits
        (== ?determiner-unit ?nominal-unit)))
  (?determiner-unit
    (sem-cat  (==1 (is-countable ?countable))))
  (?nominal-unit
    (sem-cat  (==1 (is-countable ?countable))))
  ((J ?nominal-phrase ?top-unit
      (?determiner-unit ?nominal-unit))
    (footprints (==1 determiner-nominal-cxn))
    (sem-cat (==1 (sem-function referring)))))
<-->
 ((?top-unit
    (footprints (==0 determiner-nominal-cxn))
    (syn-subunits
        (== ?determiner-unit ?nominal-unit)))
  (?determiner-unit
    (syn-cat
      (==1 (is-definite ?definiteness)
           (number ?number))))
  (?nominal-unit
     (syn-cat  (==1 (number ?number))))
  ((J ?nominal-phrase ?top-unit
      (?determiner-unit ?nominal-unit))
     (syn-cat
       (==1 (is-definite ?definiteness)
            (number ?number)))
     (footprints (==1 determiner-nominal-cxn)))))
```

An example of the application of this construction follows shortly (see Figures 6 and 7).

6.3. Linking variables

Next, we want the phrasal construction to combine the meanings contributed by the different constituents, which can be done by linking their variables. However, we cannot specify directly in a construction what kind of linking should be done because the construction has to be general and apply with any kind of lexical item that satisfies the functional and agreement constraints. Thus, linking goes through arguments that are explicitly declared for this purpose.

First the lexical constructions for "the" and "mouse" are extended to include declarations of the arguments that are available for linking (additions in bold). Rather than adding another template, we do this with an additional slot called `:args` for the `def-lex-skeleton` template:

```
(def-lex-cxn the-cxn
  (def-lex-skeleton the-cxn
   :meaning (== (unique-definite ?indiv ?context))
  :args (?indiv ?context)
  :string "the")
  (def-lex-cat the-cxn
   :sem-cat (==1 (is-countable +)
                 (determination definite))
   :syn-cat (==1 (lex-cat article)
                 (number ?sing-or-plural)
                 (is-definite +))))
```

```
(def-lex-cxn mouse-cxn
  (def-lex-skeleton mouse-cxn
    :meaning (== (mouse ?mouse-set ?base-set))
    :args (?mouse-set ?base-set)
    :string "mouse")
  (def-lex-cat mouse-cxn
    :sem-cat (==1 (is-animate +)
                  (is-countable +)
                  (class object))
    :syn-cat (==1 (lex-cat noun)
                  (number singular))))
```

What this means for example for "mouse" is that the variables `?mouse-set` and `?base-set` are available to link the meaning of "mouse" to meanings provided by other lexical items.

Another template called def-phrasal-linking is introduced to add the linking of variables between the different units to a phrasal construction already set up with def-phrasal-skeleton. Here is an example of its use:

```
(def-phrasal-linking determiner-nominal-cxn
  (?nominal-phrase
     :args (?referent ?context))
  (?determiner-unit
     :args (?referent ?nominal-referent))
  (?nominal-unit
     :args (?nominal-referent ?context)))
```

The ?referent variable is shared between the ?nominal-phrase and the ?determiner-unit and the ?context variable between the ?nominal-phrase and the ?nominal-unit. The variable ?nominal-referent links the meaning supplied by the determiner to that supplied by the nominal.

The above specification expands into the following full construction, with additions by def-phrasal-linking in bold. The additions consist of feature values for the args feature in the semantic pole. The args feature of the phrase is part of the J-unit because it is to be added by the construction.

```
(def-cxn determiner-nominal-phrase-cxn ()
 ((?top-unit
    (footprints (==0 determiner-nominal))
    (sem-subunits (== ?determiner-unit ?nominal-unit)))
  (?determiner-unit
     (args (?referent ?nominal-referent))
    (sem-cat (==1 (is-countable ?countable))))
  (?nominal-unit
     (args (?nominal-referent ?context))
    (sem-cat (==1 (is-countable ?countable))))
  ((J ?nominal-phrase ?top-unit
      (?determiner-unit ?nominal-unit))
    (args (?referent ?context))
    (footprints (==1 determiner-nominal))
    (sem-cat (==1 (sem-function referring)))))
 <-->
 ((?top-unit
    (footprints (==0 determiner-nominal))
    (syn-subunits (== ?determiner-unit ?nominal-unit)))
```

```
(?determiner-unit
  (syn-cat (==1 (is-definite ?definiteness)
                (number ?number))))
(?nominal-unit
  (syn-cat (==1 (number ?number))))
((J ?nominal-phrase ?top-unit
    (?determiner-unit ?nominal-unit))
  (syn-cat
     (==1 (is-definite ?definiteness)
          (number ?number)))
  (footprints (==1 determiner-nominal)))))
```

This example demonstrates how templates help to deal with complexity. Things are beginning to look fairly complicated even though the `determinal-nominal-cxn` is still highly simplified.

6.4. Constructional Form and Meaning

Finally a phrasal construction should obviously be able to impose additional form constraints, such as the word order of the constituents. Moreover, one of the key tenets of construction grammar is that a construction can also contribute novel meanings and semantic categorizations to a phrase or its constituents. New forms and meanings could also be added to any of the constituents as well. For example, a construction could determine the syntactic and semantic functions of one of its constituents.

To be able to specify this information, we employ a new template: `def-phrasal-require`. It specifies what the construction itself requires (when it is used in matching) or imposes (when it is used in merging) on the respective units. The constructional form constraint concerns here concerns only that the determiner and the nominal have to follow each other, and the constructional meaning introduces the context of the referring expression:

```
(def-phrasal-require determiner-nominal-cxn
    (?nominal-phrase
     :cxn-form (== (meets ?determiner-unit ?nominal-unit))
     :cxn-meaning (== (context ?context))))
```

The fully expanded `determiner-nominal` construction now looks as follows (with additions in bold). This is now the complete definition of the construction.

```
(def-cxn determiner-nominal-phrase-cxn ()
 ((?top-unit
    (tag ?meaning (meaning (== (context ?context))))
    (sem-subunits
         (== ?determiner-unit ?nominal-unit))
    (footprints (==0 determiner-nominal-cxn)))
  (?determiner-unit
    (sem-cat (==1 (is-countable ?countable)
                  (determiner ?determiner)))
    (args (?referent ?nominal-referent)))
  (?nominal-unit
    (args (?nominal-referent ?context))
    (sem-cat (==1 (is-countable ?countable)
                  (sem-function identifier))))
  ((J ?nominal-phrase ?top-unit
      (?determiner-unit ?nominal-unit))
    ?meaning
    (args (?referent ?context))
    (sem-cat (==1 (sem-function referring)))
    (footprints (==1 determiner-nominal-cxn))))
<->
 ((?top-unit
    (footprints (==0 determiner-nominal-cxn))
    (syn-subunits (== ?determiner-unit ?nominal-unit))
     (tag ?form
       (form
         (== (meets ?determiner-unit ?nominal-unit)))))
  (?determiner-unit
    (syn-cat (==1 (syn-function determiner)
                  (number ?number)
                  (is-definite ?definiteness))))
  (?nominal-unit
    (syn-cat (==1 (number ?number)
                  (syn-function nominal))))
  ((J ?nominal-phrase ?top-unit
     (?determiner-unit ?nominal-unit))
    ?form
    (footprints (==1 determiner-nominal-cxn))
    (syn-cat (==1 (phrase-type nominal-phrase)
```

```
(is-definite ?definiteness)
(number ?number))))))
```

Note how the semantic and syntactic pole of the `?top-unit` specifies respectively the form and meaning that the construction requires or imposes. The feature-values are tagged with `?meaning` and `?form` respectively and their bindings are then moved to the phrasal-unit, just like lexical constructions tag meanings and forms and move them to the lexical units they create.

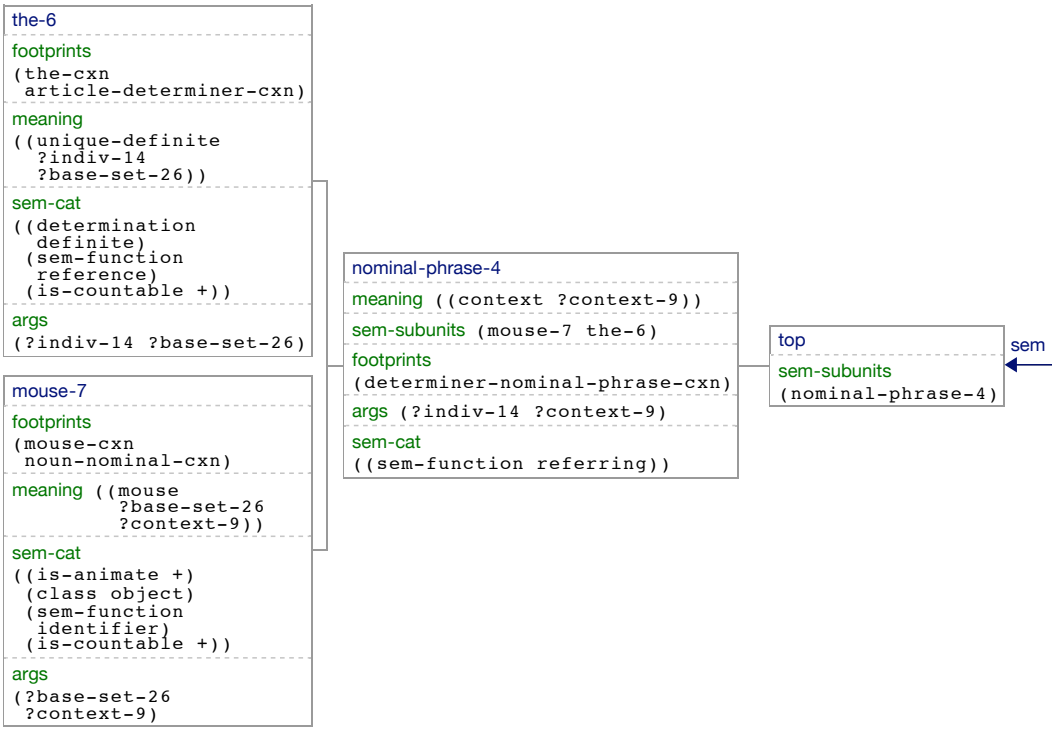

Figure 6. *Semantic pole of transient structure after application of the determiner-nominal phrasal construction when parsing "the mouse".*

The application of the complete phrasal construction is illustrated in Figure 6 and Figure 7 and is an example of parsing with the phrase "the mouse" as input. We now see that the variables have all been chosen so as to link the predicates supplied by the constituent meanings. The nominal phrase introduces the context-variable `?base-set-332`, which is the one that is taken by the predicate `mouse` in the noun unit to come up with a set of mice, bound to `?context-737`. This noun is then used

by the determiner unit to pick out the unique individual `?indiv-285`. Suppose in production a meaning like the following one is given as input:

```
((unique-definite indiv-mouse-1 mouse-1)
 (mouse mouse-1 context-1)
 (context context-1))
```

Then the same set of constructions produces a transient structure which is entirely similar. All constructions are perfectly reversible.

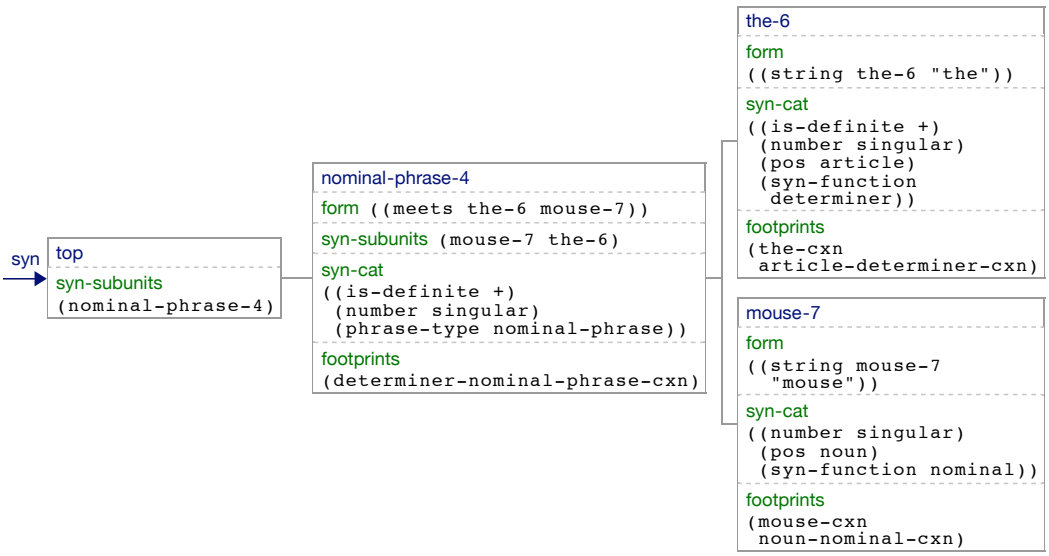

Figure 7. *Syntactic pole of transient structure after parsing "the mouse".*

7. Other examples of phrasal constructions

These templates are now exercised with three more examples: an adjectival phrase built from an adverbial and an adjectival, a nominal built from an adjectival and a nominal, and a postposed genitive, such as "this friend of mine", which combines a nominal phrase and a genitive into a new nominal phrase.

7.1. Building adjectival phrases

An adjectival phrase combines an adverbial and an adjectival, as in "very green". Let us first extend the lexicon with an example of an adverb and an adjective:

```
(def-lex-cxn green-cxn
  (def-lex-skeleton green-cxn
    :meaning (== (green ?green-set ?context))
    :args (?green-set ?context)
    :string "green")
  (def-lex-cat green-cxn
    :sem-cat (==1 (category hue))
    :syn-cat (==1 (lex-cat adjective))))
```

and

```
(def-lex-cxn very-cxn
  (def-lex-skeleton
    :meaning (== (very ?very-set ?very-base-set))
    :args (?very-set ?very-base-set)
    :string "very")
  (def-lex-cat
   :sem-cat (== (similarity prototype))
   :syn-cat (==1 (lex-cat adverb))))
```

Next we need functional constructions for adjectivals and adverbials all defined using templates:

```
(def-fun-cxn adjectival
  (def-fun-skeleton
   :sem-cat (==1 (sem-function qualifier))
   :sem-function qualifier
   :syn-cat (==1 (lex-cat adjective))
   :syn-function adjectival))
```

```
(def-fun-cxn adverbial
  (def-fun-skeleton
   :sem-cat (==1 (sem-function modifier))
   :sem-function modifier
   :syn-cat (==1 (lex-cat adverb))
   :syn-function adverbial))
```

And finally we define a phrasal construction combining adverbials and adjectivals into an adjectival phrase. There are no agreement relations, so we only need to specify the phrasal skeleton and the linking of variables:

```
(def-phrasal-cxn adverbial-adjectival-cxn
  (def-phrasal-skeleton adverbial-adjectival-cxn
    :phrase
    (?adjective-phrase
     :sem-function qualifier
     :syn-function adjectival
     :phrase-type adjectival-phrase)
    :constituents
    ((?adverbial-unit
      :sem-function modifier
      :syn-function adverbial)
     (?adjectival-unit
      :sem-function qualifier
      :syn-function adjectival)))
  (def-phrasal-linking adverbial-adjectival-cxn
    (?adjective-phrase
     :args (?adverbial-referent ?base-set))
    (?adverbial-unit
      :args (?adverbial-referent ?adjectival-referent))
    (?adjectival-unit
      :args (?adjectival-referent ?base-set)))
  (def-phrasal-require adverbial-adjectival-cxn
    (?adjective-phrase
      :cxn-form (== (meets ?adverbial-unit ?adjectival-unit)))))
```

These constructions can parse phrases such as “very green”. They can also be used to illustrate how FCG deals with recursion. When the adjectival construction applies, it not only constructs an adjectival-phrase, but it also assigns the sem-function `qualifier` and the syn-function `adjectival` to this new phrase, so that the newly constructed phrasal unit can itself be used again as a component of an adjectival-phrase. Here is such an example for production. Processing starts from the following initial meaning:

```
((very very-set-2 very-set-1)
 (very very-set-1 green-set-1)
 (green green-set-1 context-1))
```

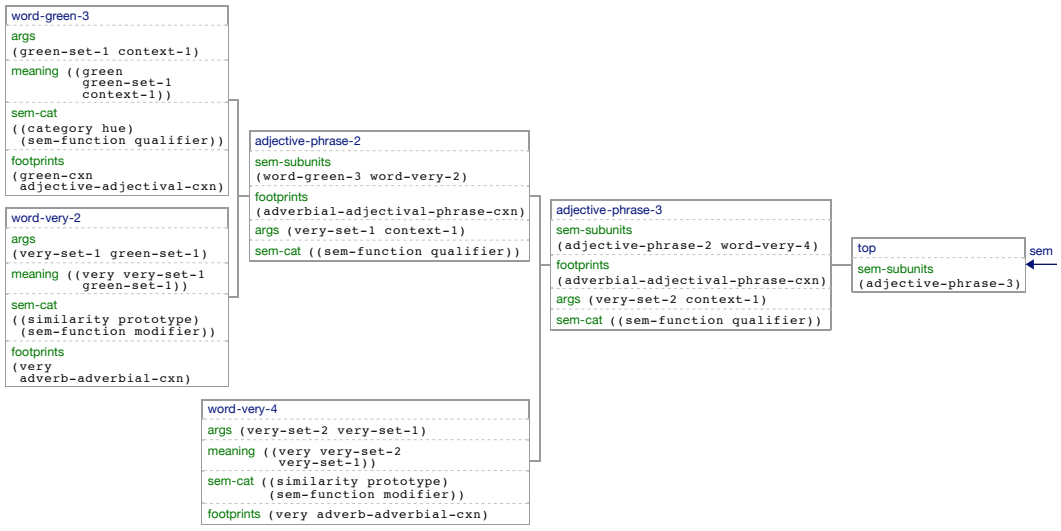

Figure 8. *Semantic pole of the transient structure after a recursive application of the adjectival-phrase-cxn.*

It produces the result shown in Figure 8 and Figure 9. First an adjectival phrase is built for the lexical items “very” and “green” that cover

```
(very very-set-1 green-set-1)
(green green-set-1 context-1)
```

Then the resulting unit is combined with a second lexical unit for “very” that covers

```
(very very-set-1 green-set-1)
```

The same constructions work perfectly well in parsing and would produce a similar transient structure. It is worthwhile to point out that processing recursive language in FCG is not handled by a separate stack mechanism that acts as an additional memory (as you would typically find in context-free grammar parsers). The transient structure itself acts as a memory and the standard mechanisms for applying constructions based on the matching and merging operations apply.

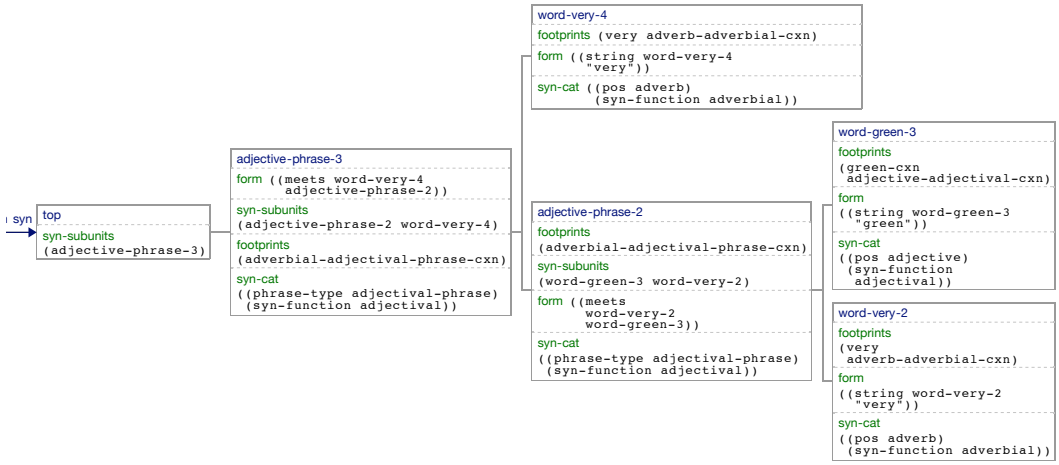

Figure 9. *Syntactic pole of the transient structure after recursive application of the adjectival phrase construction.*

7.2. Building nominals

By using syntactic and semantic functions for constraining whether an item or phrase can be part of another one, we get the unbounded compositionality and recursivity required in natural language processing. Here is another example to illustrate this feature.

A second way to get a nominal is by combining an adjectival with a nominal, as in “green mouse” or “very green mouse”. The two units have to follow each other sequentially, and the arguments have to be linked in particular ways. Moreover, there are semantic and syntactic categorizations that have to be satisfied by each. If that is the case, a new adjectival-nominal unit can be constructed that is still a nominal and functions as an identifier:

```
(def-phrasal-cxn adjectival-nominal-cxn
 (def-phrasal-skeleton adjectival-nominal-cxn
   :phrase
   (?adjectival-nominal
     :sem-function identifier
     :syn-function nominal
     :phrase-type adjectival-nominal)
   :constituents
   ((?adjectival-unit
     :sem-function qualifier
     :syn-function adjectival)
    (?nominal-unit
      :sem-function identifier
      :syn-function nominal)))
  (def-phrasal-require adjectival-nominal-cxn
    (?adjectival-nominal
      :cxn-form
         (== (meets ?adjectival-unit ?nominal-unit))))
  (def-phrasal-agreement adjectival-nominal-cxn
    (?adjectival-nominal
      :sem-cat (==1 (is-countable ?countable))
      :syn-cat (==1 (number ?number)))
    (?nominal-unit
      :sem-cat (==1 (is-countable ?countable))
      :syn-cat (==1 (number ?number))))
  (def-phrasal-linking  adjectival-nominal-cxn
    (?adjectival-nominal
      :args (?adjectival-referent ?context))
    (?adjectival-unit
      :args (?adjectival-referent ?nominal-referent))
    (?nominal-unit
      :args (?nominal-referent ?context))))
```

Based on this definition we can now parse and produce a phrase like “the green mouse” but also “the very green mouse”. The semantic pole at the end of the process for parsing the latter sentence is shown in Figure 10. It shows that meanings contributed by individual words have all been linked together properly and that all agreement and percolation requirements are satisfied.

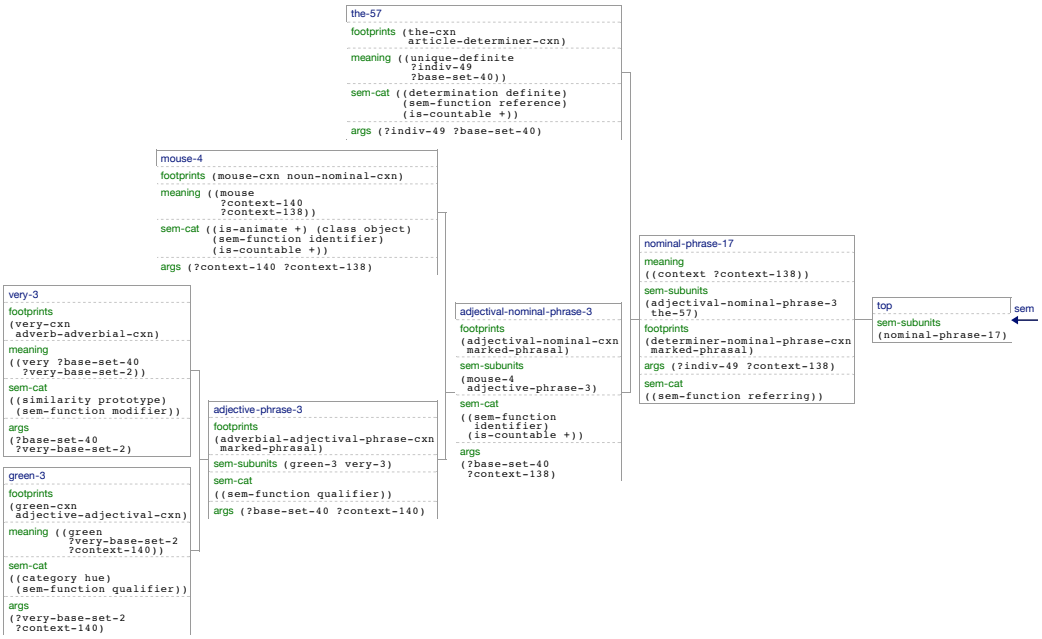

Figure 10. *Semantic pole after compositional application of different lexical and phrasal constructions progressively constructing a complex nominal phrase for parsing "the very green mouse".*

Using exactly the same constructions, we can produce the same phrase starting from the following initial meaning:

```
((unique-definite indiv-mouse-1 very-1)
 (context context-1)
 (very very-1 green-1)
 (green green-1 mouse-1)
 (mouse mouse-1 context-1))
```

After all relevant constructions are applied, the syntactic pole is shown in Figure 11. Individual words and ordering constraints have all been properly added.

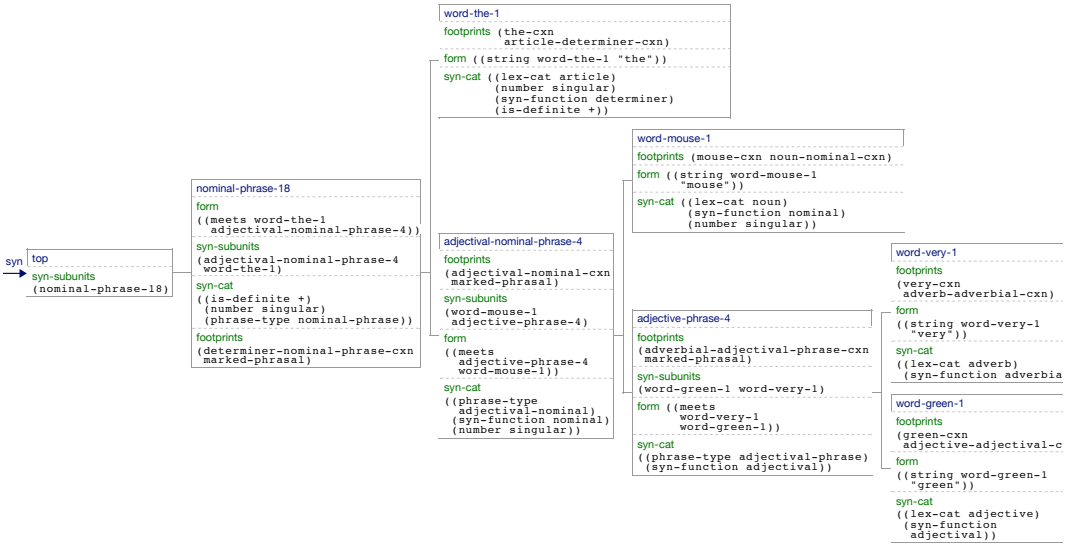

Figure 11. *Compositional application of phrasal constructions in producing "the very green mouse". This figure shows the syntactic pole at the end of the process.*

7.3. The postposed-genitive

To further illustrate the templates proposed here, let us look at the so-called `postposed-genitive` construction, seen in phrases such as "this dog of mine", "these silly ideas of yours". It consists of a nominal phrase followed by the preposition "of", which has here a purely grammatical function, followed by a genitive. The genitive can be a proper name (as in "this dog of John's") or a pronoun in the genitive case ("mine", "yours", "theirs") (Lyons, 1985). The focus is on the pronoun case, with "of" treated as a purely grammatical function word.

Firstly, more lexical constructions are defined, using the same `def-lex-cxn` template as before. The word "this" is treated as a determiner so that the determiner-nominal construction can be reused. This step is of course a short-cut to keep the examples discussed here as simple as possible.

```
(def-lex-cxn this-cxn
  (def-lex-skeleton this-cxn
    :meaning (== (proximal-reference ?indiv ?context))
    :args (?indiv ?context)
    :string "this")
  (def-lex-cat this-cxn
   :sem-cat (==1 (is-countable +)
                 (determination definite))
   :syn-cat (==1 (lex-cat article)
                 (number singular)
                 (is-definite +))))
```

The word “mine” is treated as a genitive pronoun. “Mine” refers to the speaker in the dialogue. It is given a possessive semantic function.

```
(def-lex-cxn mine-cxn
  (def-lex-skeleton mine-cxn
    :meaning
       (== (dialogue-participant ?indiv speaker))
    :args (?indiv)
    :string "mine")
  (def-lex-cat mine-cxn
    :sem-cat (==1 (sem-function possessive))
    :syn-cat (==1 (lex-cat pronoun)
                  (person 1st)
                  (number singular)
                  (case genitive))))
```

The postposed-genitive construction is defined using the `def-phrasal-cxn` template again. It needs to construct a new nominal phrase based on two constituents: a nominal phrase and a pronominal. The construction introduces the `possessive` meaning and adds form constraints to its constituents, namely constituent ordering and the use of the grammatical function word “of”. Arguments must get properly linked to express the possessive relation.

```
(def-phrasal-cxn postposed-genitive-cxn
  (def-phrasal-skeleton postposed-genitive-cxn
    :phrase
    (?possessive-nominal-phrase
      :sem-function referring
      :phrase-type nominal-phrase)
    :constituents
    ((?nominal-unit
       :sem-function referring
       :phrase-type nominal-phrase)
     (?pronominal-unit
       :sem-function possessive
       :lex-cat pronoun
       :syn-cat (==1 (case genitive)))))
  (def-phrasal-require postposed-genitive-cxn
    (?possessive-nominal-phrase
      :cxn-meaning
         (== (possessive ?referent-nominal
                         ?referent-pronominal))
      :cxn-form (== (meets ?nominal-phrase ?word-of)
                    (string ?word-of "of")
                    (meets ?word-of ?pronominal-unit))))
  (def-phrasal-agreement postposed-genitive-cxn
    (?possessive-nominal-phrase
      :syn-cat (==1 (number ?number)
                    (is-definite ?definiteness)))
    (?nominal-unit
      :syn-cat  (==1 (is-definite ?definiteness)
                     (number ?number))))
  (def-phrasal-linking postposed-genitive-cxn
    (?possessive-nominal-phrase
      :args (?referent-nominal))
    (?nominal-unit
      :args (?referent-pronominal))
    (?pronominal-unit
      :args (?referent-pronominal))))
```

The semantic pole after processing the various lexical, categorial and phrasal constructions for the sentence “this mouse of mine” is shown in Figure 12. Notice how the possessive meaning has been added to the top unit and how all the different arguments have been correctly linked. The syntactic pole created after processing the

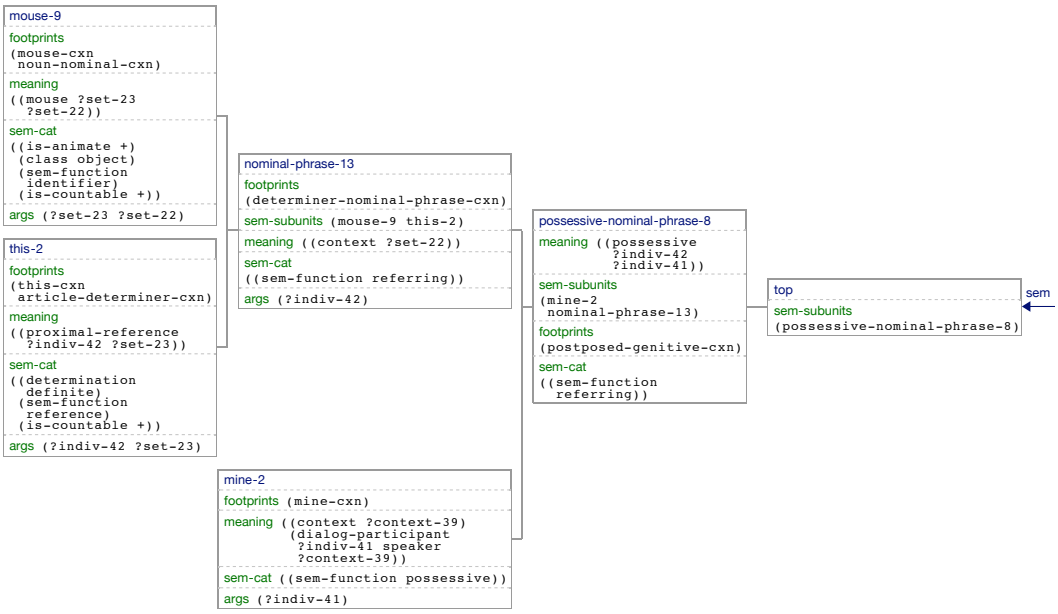

Figure 12. *An example of parsing using the postposed-genitive phrasal construction. Only the semantic pole is shown.*

following meaning is shown in Figure 13.

```
((context context-1)
 (mouse mouse-set-1 context-1)
 (proximal-reference indiv-1 mouse-set-1)
 (context context-2)
 (dialogue-participant indiv-2 speaker context-2)
 (possessive indiv-1 indiv-2))
```

The word “of” functions here purely as a grammatical word that does not have its own lexical unit. It is simply part of the postposed-genetive.

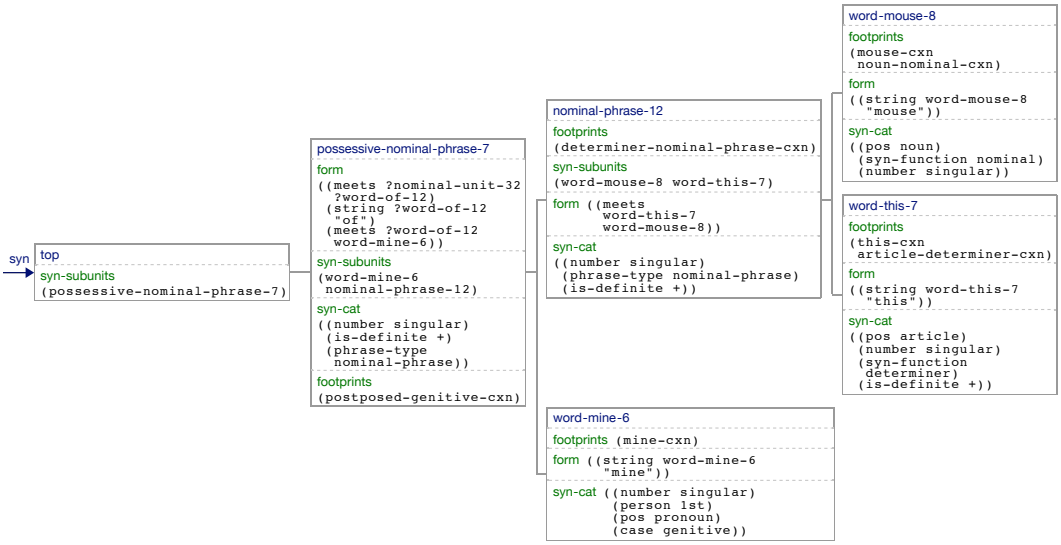

Figure 13. *An example of production using the postposed-genitive phrasal construction. Only the syntactic pole is shown.*

8. Conclusions

This paper has considered examples of phrasal constructions and how they are handled in Fluid Construction Grammar. More concretely, it shows how compositionality, hierarchy, recursion, percolation, agreement and constructional meaning or form can be defined and processed. The examples were deliberately simplified so that there would be no search or other problems in dealing with multi-functionality, ambiguity or indeterminacy. These topics are discussed in other chapters in this book.

The paper also illustrates how templates are used to simplify the writing of lexicons and grammars. Templates make it much easier to write constructions and they help to avoid errors in the definition of a grammar. Moreover these templates are also useful to start thinking about learning operators in the sense that the elements that can fill the various slots in a template are a heuristic guide on what needs to be learned in the acquisition of phrasal constructions. It is important nevertheless to understand the behavior of the fully expanded operational definition of constructions in order to understand how transient structures get built, and to pos-

sibly extend or develop new templates that give better support for issues raised in a particular language.

Acknowledgements

The research reported here was conducted at the Sony Computer Science Laboratory in Paris and the Artificial Intelligence Laboratory of the Free University of Brussels (VUB). and the EU-FP7 project ALEAR.

References

Anderson, John (1971). Dependency and grammatical functions. *Foundations of Language*, 7, 30–37.

Bleys, Joris, Kevin Stadler, Joachim De Beule (2011). Search in linguistic processing. In Luc Steels (Ed.), *Design Patterns in Fluid Construction Grammar*. Amsterdam: John Benjamins.

Bloomfield, Leonard (1993). *Language*. New York: Henry Holt.

Chomsky, Noam (1957). *Syntactic Structures*. Berlin: Mouton de Gruyter.

Croft, William (2001). *Radical Construction Grammar: Syntactic Theory in Typological Perspective*. Oxford: Oxford UP.

Dik, Simon (1978). *Functional Grammar*. London: Academic Press.

Haspelmath, Martin (2007). Pre-established categories don't exist. *Linguistic Typology*, 11(1), 119–132.

Langacker, Ron (2008). *Cognitive Grammar. A Basic Introduction*. Oxford: Oxford University Press.

Lyons, Christopher (1985). The syntax of english genitive constructions. *Linguistics*, 12, 123–143.

Masahito, Ikawa, Shuichi Yamada, Tomoko Nakanishi, Masani Okabe (1999). Green fluorescent protein (gfp) as a vital marker in mammals. *Curr Top Dev Biol.*, 44, 1–20.

Mel'cuk, Igor (1988). *Dependency syntax: Theory and Practice.* Albany NY: State University Press of New York.

Micelli, Vanessa, Remi van Trijp, Joachim De Beule (2009). Framing fluid construction grammar. In N.A. Taatgen, H. van Rijn (Eds.), *Proceedings of the 31th Annual Conference of the Cognitive Science Society*, 3023–3027. Cognitive Science Society.

Partee, Barbara (2003). *Compositionality in Formal Semantics: Selected Papers of Barbara Partee.* Oxford: Blackwell Publishers.

Perlmutter, David (1983). *Studies in Relational Grammar*. Chicago: Chicago University Press.

Sgall, Peter, Jarmila Panevova (1989). Dependency syntax - a challenge. *Theoretical Linguistics*, 15, 30–37.

Siewierska, Anna (1991). *Functional Grammar*. London: Routledge.

Spranger, Michael, Martin Loetzsch (2011). Syntactic indeterminacy and semantic ambiguity: A case study for German spatial phrases. In Luc Steels (Ed.), *Design Patterns in Fluid Construction Grammar*. Amsterdam: John Benjamins.

Steels, Luc (2011a). A first encounter with Fluid Construction Grammar. In Luc Steels (Ed.), *Design Patterns in Fluid Construction Grammar*. Amsterdam: John Benjamins.

Steels, Luc (2011b). Introducing Fluid Construction Grammar. In Luc Steels (Ed.), *Design Patterns in Fluid Construction Grammar*. Amsterdam: John Benjamins.

Talmy, Leonard (2000). *Toward a Cognitive Semantics, Concept Structuring Systems*, vol. 1. Cambridge, Mass: MIT Press.

van Trijp, Remi (2011a). A design pattern for argument structure constructions. In Luc Steels (Ed.), *Design Patterns in Fluid Construction Grammar*. Amsterdam: John Benjamins.

van Trijp, Remi (2011b). Feature matrices and agreement: A case study for German case. In Luc Steels (Ed.), *Design Patterns in Fluid Construction Grammar*. Amsterdam: John Benjamins.