Notice

This paper is the author's draft and has now been published officially as:

De Beule, Joachim (2012). A Formal Deconstruction of Fluid Construction Grammar. In Luc Steels (Ed.), *Computational Issues in Fluid Construction Grammar*, 215–238. Berlin: Springer.

BibTeX:

```
@incollection{debeule2012formal,
   Author = {De Beule, Joachim},
   Title = {A Formal Deconstruction of Fluid Construction Grammar},
   Pages = {215--238},
   Editor = {Steels, Luc},
   Booktitle = {Computational Issues in {Fluid Construction Grammar}},
   Publisher = {Springer},
   Series = {Lecture Notes in Computer Science},
   Volume = {7249},
   Address = {Berlin},
   Year = {2012}}
```

A Formal Deconstruction of Fluid Construction Grammar

Joachim De Beule

Artificial Intelligence Laboratory, Vrije Universiteit Brussel, Belgium

Abstract. Fluid construction grammar was primarily developed for supporting the on-line processing and learning of grammatical language in robotic language game setups, and with a focus on semantics and constructrion grammar. In contrast, many related formalisms were developed to support the formulation of static, primarily syntactic theories of natural language. As a result, many of FCG's features are absent in other formalisms, or take a somewhat different form. This can be confusing and give FCG a 'peculiar' status from the perspective of those more familiar with other formalisms. This chapter aims to clarify some of these peculiarities by providing a formal deconstruction of FCG based on a reconstruction of it's history.

1 Introduction

FCG was primarily developed for supporting the on-line processing and learning of grammatical language in robotic language game setups.[1] In contrast, many related formalisms were developed to support the formulation of static theories of natural language. As a result, many of FCG's features are absent in other formalisms, or take a somewhat different form. This can be confusing and give FCG a 'peculiar' status from the perspective of those more familiar with other formalisms. This chapter aims to clarify some of these peculiarities.

In the hope that this will help to accomplish this goal, our strategy will be to reconstruct (most of) FCG in a bottom up fashion. We start from the fact that the history and development of FCG are part of a longer tradition to make use of robotic and computational setups for investigating the emergence and evolution of language. As the complexity of these setups grew, the need arose to handle ever more complex aspects of language. Many of FCG's peculiarities today represent concrete solutions to specific problems encountered in this process. The strategy taken in this paper is therefore to give a deconstruction of FCG based on a reconstruction of its history.

It would also be possible to achieve clarification in another way, by providing a detailed account of the formal differences that exist between FCG and related formalisms. Although this is not the main approach taken here, some high level differences are touched upon in the next section. This will at least provide some

[1] In a language game setup, robotic agents interact in order to establish a communication system or *artificial language* [17].

dimensions along which such a comparison could take place, and helps to set the stage for the rest of the paper.

2 General Considerations

One of the most distinguishing and, perhaps, confusing aspects of FCG is that it lacks any notion of *well formedness*, at least not in the sense as is customary in other approaches. For instance, in most unification or constraint based formalizations of language, one is typically interested in the minimal consistent set of constraints that specifies, say, all of English (see e.g. [15].) In contrast, in FCG, the validity of a constraint or construction is not measured by its consistency or by how it restricts the set of well formed sentences. Rather, it is measured by how it enables communication. This calls for a *functional* approach to language rather than a *declarative* one.

The notion of "unification" in FCG is also somewhat peculiar. In general, unification is about finding conditions under which certain things can be done, that is, are "valid" or "well formed". Particularly, in other approaches, the set of well formed structured equals the the set of structures entailed by a given grammar through the process of unification. A particular sentence, in this view, specifies a number of specific constraints – the particular word forms in the sentence and their order. These carve out a subset of the set of all well formed sentences. As mentioned, determining this subset is what processing in most unification based approaches is about. In FCG, unification is used for somewhat different purposes, and therefore in different ways.

According to Construction Grammar (CxG), linguistic knowledge is organized in *constructions*. These are entrenched routines that are generally used in a speech community and that involve a pairing of meaning components to form elements [6]. In FCG, constructions are more specifically *routines for expressing a meaning and for understanding a form*. Like functions, FCG constructions transform meaning specifications to form specifications during production, and vice versa during parsing. In this view, a particular sentence specifies a number of '*seeds*' rather than constraints. Each 'seed' (e.g. a word or specific syntactic category) triggers constructions which in turn add more 'seeds' (e.g. the meaning or semantic category of the word). This is the reason why, in FCG, there are *two* sorts of "unification": one for determining the set of "valid" constructions (those that are triggered), and one that governs the actual application of activated constructions ("merge").

Another important distinction in FCG is the notion of *processing direction*. In the tradition of generative grammar, the two processing modes normally distinguished are *parsing* and *generation*. Parsing refers to bottom-up process by which a particular sentence is analyzed in terms of production rules. Generation refers to the top-down process of determining all possible sentences that can be generated with the production rules. The two processing modes in FCG, normally called parsing and *production*, have nothing to do with all this. In FCG, parsing refers to the process of determining the meaning of a given sentence,

and production refers to the process of determining a form by which a meaning can be expressed. The duality between FCG's processing modes is thus of a different nature than the one typically conceived of in the generative tradition. Some of FCG's peculiarities, like the fact that feature structures have two poles (a *semantic* and a *syntactic* pole), directly derive from this difference.

Finally, since FCG was developed for supporting *artificial* language processing, it makes the least possible amount of claims about the structure or nature of language.[2] Almost everything in FCG was introduced because it was required for solving a problem of language processing. For instance, certain generalizations require that it is possible to distinguish between specific word forms and the word classes they can belong to (e.g. their part of speech). FCG provides ways to make such kinds of distinctions, but it does not specify what particular distinctions *should* be made. There are no type hierarchies –at least none that can not be easilly circumvented or changed or extended; no intrinsic restrictions on the number or sort of features in feature structures; no restrictions on the values of features, etc.

In the following, we trace back the origins of these and other features of FCG by incrementally building up the machinery required to process an increasingly complex set of language phenomena. Throughout the paper, it will be useful to keep in mind a robotic agent that faces the task of parsing or producing an utterance. Producing an utterance amounts to transforming a given *meaning specification* into a *form specification*. Parsing amounts to transforming a given form specification into a meaning specification. This bi-directional problem structure is formalized throughout the chapter with the help of *production* and *parsing* functions, denoted as $g_{\rightarrow}$ and $g_{\leftarrow}$ respectively. The precise form of these functions, as well as of the meaning and form specifications upon which they operate, will gradually be changed and their complexity increased, mimicking the evolution of FCG itself. In the next section, we start by specifying the most basic elements of meaning and form specifications, called components, in more detail.

3 Meaning and Form Specifications

In general, with language processing we mean the transforming of meaning specifications into form specifications and vice versa. By definition, such specifications are built from primitive *meaning* and *form components* respectively. As is customary in FCG, such components will be represented as expressions in prefix-list notation. Furthermore, names of frames and frame relations will be used in components as specified in the on-line FrameNet database [2, 8]. For example, the following meaning component specifies the '`[line]`' frame:

```
(frame x [line]).
```

[2] Although recent advances, e.g. so calles "design patterns", can be seen as such.

The symbol '`x`' is a skolem constant: it represents a specific instance of the `[line]` frame. I will sometimes further refer to skolem constants in meaning components as *meaning constants*.

By definition, meaning specifications are collections of meaning components. For example, the meaning of the phrase "natural line" is

```
{(frame x [line]), (frame y [natural]),
            (fe-relation y [entity] x)}.
```

The second component in this set introduces '`y`' as an instance of the '`[natural]`' frame. The third component specifies that the '`[entity]`' frame element of this frame is the frame '`x`'.

Similarly, form specifications are collections of form components. The form of the phrase "natural line" for example is represented as below.

```
{(lexeme a "line"), (lexeme b "natural"), (meets a b)}.
```

The skolem constants '`a`' and '`b`' are *form constants*.

In the next section, we start investigating the processing of language in the case of the simplest meaning and form specifications possible: those consisting of a single meaning and form component only.

4 Holistic Language Processing

We first restrict the discussion to "holistic languages". In a holistic language, phrases are always 'atomic': they are not structured or made out of smaller components in any meaningful way. Formally, this corresponds to the case that meaning and form specifications specify exactly a single component each. In other words, we only consider *singleton* specifications in this section.

4.1 Constructions

The bi-directional processing of singleton meaning and form specifications calls for a bi-directional lookup table. Each entry in the table associates a meaning component with a form component, and in this sense is a construction. The following is an example construction that associates the meaning specification '`{(frame x [line])}`' with the form specification '`{(lexeme a "line")}`'.

$$c_1(x,a) = \boxed{\texttt{\{(frame x [line])\}} \leftrightarrow \texttt{\{(lexeme a "line")\}}}. \qquad (1)$$

This construction is not in full concordance with contemporary practices in FCG however, and we will change its form again later. Nevertheless, the convention of displaying constructions in a box will be maintained throughout the chapter. The meaning side is separated from the form side by a double arrow. The meaning and form sides of a construction 'c' will also be referred to as 'c^m' and 'c^f', respectively. For example, with c_1 as above:

$$c_1^m(x) = \texttt{\{(frame x [line])\}} \quad ; \quad c_1^f(a) = \texttt{\{(lexeme a "line")\}}.$$

The semantic part of a construction will generally be referred to as the construction's *semantic pole*. Likewise, the syntactic part is referred to as the construction's *syntactic pole*.

4.2 Variables and Bindings

When faced with the problem of expressing the meaning specification '`{(frame x [line])}`', an agent can retrieve the construction $c_1(x)$ from its lookup table and notice that it specifies the exact same meaning specification in its semantic pole. It can therefore conclude that this meaning is expressed by the construction's syntactic pole.

But when given instead, say, the specification '`{(frame y [line])}`', a problem arises. The problem is that the construction specifies the skolem constant '`x`', whereas the new specification specifies the constant '`y`'. In consequence, there will be no match during lookup. A proper construction, therefore, should specify a *variable* instead of a constant, so that it matches *all* instances of the '[line]' frame.

By convention, variables will be marked by a leading question mark, as in '`?x`'. Variables are implicitly and universally quantified over, so the introduction of variables is like the inverse of skolemization. A variable can take any (but only one) value. A specific assignment $[?x/x]$ of a variable $?x$ to a value x is called a *binding* or a *basic substitution*. A set of bindings B of variables $?x_1, ?x_2, ...$ to values $x_1, x_2, ...$ is represented as $B = [?x_1/x_1, ?x_2/x_2/, ...]$. Every set of bindings B induces a *substitution function* $\sigma_B(e)$, which replaces all variables that occur in an expression e by their value according to the bindings in B.

Replacing the skolem constants '`x`' and '`a`' in construction c_1 by variables and using the abbreviations

$$m_1(?x) = \texttt{(frame ?x [line])} \quad \text{and} \quad f_1(?a) = \texttt{(lexeme ?a "line"),}$$

thus gives the following more general definition of construction c_1:

$$c_1(?x, ?a) = \boxed{\{m_1(?x)\} \leftrightarrow \{f_1(?a)\}} \tag{2}$$

This construction associates *any* [line] frame with *any* "line" lexeme.

4.3 Component Matching

We now put everything together. Suppose that C is a constructicon –a collection of constructions, that is, a bi-directional lookup table between singleton meaning specifications and singleton form specifications– and suppose that some specification $\{m(x)\}$ needs to be expressed. For that, the agent goes through the constructions in C and compares their semantic poles to the given specification. The meaning and form components in constructions now contain variables. The operation with which components containing variables can be compared is called

component matching. We denote it by U_0. This function takes a pattern component p and a source component s and computes all sets of minimal substitutions σ_B that make the pattern identical to the source, that is, for which $\sigma_B(p) = s$.[3] An example of component matching is given below.

$$U_0(m_1(?x), m_1(x)) = \{[?x/x]\}.$$

We are now ready to give a first definition for the production and parsing functions. The production function $g_\rightarrow$ for now operates on a singleton meaning specification $\{m(x)\}$ and computes the set of form specifications that express it according to the constructicon C:

$$g_\rightarrow(m(x), C) = \{c^f : c \in C, U_0(c^m(?x), m(x)) \neq \emptyset\}.$$

Thus, this set contains the syntactic poles of all constructions of which the semantic side matches the given meaning component $m(x)$. For example, with c_1 as defined earlier:

$$g_\rightarrow(\{\texttt{(frame y [line])}\}, \{c_1\}) = \{\{\texttt{(lexeme ?a "line")}\}\}.$$

The parsing function $g_\leftarrow$ is defined in a similar way. It takes a singleton form specification and computes a set of singleton form specifications:

$$g_\leftarrow(\{f(a)\}, C) = \left\{c^m : c \in C, U_0(c^f(?a), f(b)) \neq \emptyset\right\}.$$

For example:

$$g_\leftarrow(\{\texttt{(lexeme b "line")}\}, \{c_1\}) = \{\{\texttt{(frame ?x [line])}\}\}.$$

It is possible that several constructions are available in a given constructicon that all express the same meaning (synonyms) or cover the same form (homonyms). In this case, the agent will have to select one among them. This could be done for instance on the basis of preference scores. We return to this issue in section 7.

This concludes our discussion of the processing of holistic language. In summary, it requires a bi-directional lookup table of constructions – associations between singleton meaning and form specifications containing variables –, and a component matching function for determining which constructions express a given singleton meaning specification or parse a given singleton form specification.

5 Compositionality

We now relax the assumptions that meaning and form specification may only contain a single component. This opens up the way to *compositional language*, in which phrases are structured and consist of several parts.

[3] A minimal substitution is one that does not specify bindings for variables that do not occur in the pattern or source. Component matching corresponds to the notion of standard unification in Artificial Intelligence and logic (See e.g. [13].)

5.1 Set Matching

Suppose again that C is a constructicon (a collection of constructions). As before, it is assumed that meaning and form constants are replaced with variables in constructions. Suppose further that the meaning specification that needs to be expressed is denoted by μ. As before, μ will have to be compared to the semantic poles of constructions in C. This time, however, it is possible that some construction only expresses *part* of μ. This is the case for those constructions of which the meaning pole is a *subset* of μ. The operation that checks whether one set of components is a subset of another is called *set matching*. It is denoted as U_1. This function takes a pattern set p and a source set s and computes the set of minimal substitutions σ_B that make the pattern set a subset of the source, that is, for which $\sigma_B(p) \subset s$. More formally, if μ and μ' are two meaning or form specifications, then:

$$U_1(\mu, \mu') = \{B : \sigma_B(\mu) \subset \mu'\} \tag{3}$$

New versions of the production and parsing functions that use this enhanced matching power can now be defined as follows:

$$g_{\rightarrow}(\mu, C) = \{c^f : c \in C, U_1(c^m, \mu) \neq \emptyset\},$$

$$g_{\leftarrow}(\phi, C) = \left\{c^m : c \in C, U_1(c^f, \phi) \neq \emptyset\right\}.$$

These functions now operate on arbitrary meaning and form specifications instead of just singletons. For example, if $\mu_1(x) = \{m_1(x)\}$ and $\phi_1(a) = \{f_1(a)\}$, then:

$$U_1(\{m_1(?x)\}, \mu_1(x)) = \{[?x/x]\} \quad ; \quad U_1(\{f_1(?a)\}, \phi_1(a)) = \{[?a/a]\}$$

and, with $C = \{c_1(?x, ?a)\}$ (see equation 2),

$$g_{\rightarrow}(\mu_1(x), C) = \{\phi(?a)\} \quad ; \quad g_{\leftarrow}(\phi_1(a), C) = \{\mu_1(?x)\}$$

Now consider the following meaning specification:

$$\mu(x, y) = \{m_1(x), m_2(y)\} = \{\texttt{(frame x [line]), (frame y [natural])}\}.$$

It must be transformed into the following form specification:

$$\phi(?a, ?b) = \{f_1(?a), f_2(?b)\} = \{\texttt{(lexeme ?a "line") (lexeme ?b "natural")}.$$

This is achieved with the following construction h.

$$h = \boxed{\{m_1(?x), m_2(?y)\} \leftrightarrow \{f_1(?a), f_2(?b)\}}$$

Indeed, the semantic side of h matches μ:

$$U_1(h^m(?x, ?y), \mu(x, y)) = \{[?x/x, ?y/y]\}.$$

From this, and from the definition of $g_{\rightarrow}$, it follows that

$$g_{\rightarrow}(\mu, \{h\}) = \{\phi(?a, ?b))\}$$

In a sense this is a "holistic" production because the construction h maps both meaning components to two form components *in one go*. In contrast, a compositional encoding should involve two constructions: one for the 'line' part and one for the 'natural' part. Let us call these constructions c_1 and c_2. Then c_1 is as in equation (2) and c_2 is as below.

$$\begin{aligned} c_2(?y, ?b) &= \boxed{\{m_2(?y)\} \leftrightarrow \{f_2(?b)\}} \\ &= \boxed{\{\texttt{(frame ?y [natural])}\} \leftrightarrow \{\texttt{(lexeme ?b "natural")}\}}. \end{aligned} \quad (4)$$

The semantic sides of both constructions match the given specification $\mu(x, y)$, so that the "lookup" part of processing succeeds for both of them:

$$U_1(c_1^m(?x, ?a), \mu(x, y)) = \{[?x/x]\} \;\; ; \;\; U_1(c_2^m(?y, ?b), \mu(x, y)) = \{[?y/y]\}.$$

However, the production function $g_{\rightarrow}$ as defined above produces two incomplete productions instead of a single compositional production.

$$g_{\rightarrow}(\mu, C) = \left\{\{c_1^f\}, \{c_2^f\}\right\}.$$

Compositional language processing is thus more demanding than holistic language processing, beyond the fact that it requires a more general mode of unification (set matching). It also requires a *transient linguistic structure* for collecting intermediate results (partial meaning and form specifications).

5.2 The Transient Linguistic Structure and Merge

We introduce the transient linguistic structure for keeping track of intermediate processing results. Since this is needed in the case of compositional encoding regardless of the direction of processing (parsing or production), and since this will turn out to be useful in the following, we define the transient linguistic structure to be an association between a meaning and a form specification. As in the case of constructions, the semantic and syntactic poles of a transient structure s are denoted as s^m and s^f respectively.

The initial meaning and form specifications from which processing starts and that are given as input to the production and parsing functions can also conveniently be specified as transient linguistic structures by simply leaving the appropriate pole empty, that is, the syntactic pole in case of production and the semantic polle in case of parsing. Thus, the meaning specification μ which is given before production starts, corresponds to the transient structure s_μ below.

$$s_\mu = \boxed{\mu \leftrightarrow \emptyset}.$$

Now the semantic poles of the constructions c_1 and c_2 match the semantic pole of the transient structure s_μ:

$$U_1(c_1^m, s_\mu^m) = U_1(c_1^m(?x), \mu(x,y)) = \{[?x/x]\}$$

and

$$U_1(c_2^m, s_\mu^m) = U_1(c_2^m(?y), \mu(x,y)) = \{[?y/y]\}.$$

As before, this indicates that both constructions can be considered for further processing by the production function. The idea is that each of the constructions adds its own parts to the transient structure. This is done through *merging*.

In this section, we do not consider any other relationships between meaning or form components besides the fact that they appear together in meaning and form specifications. In this case, all that needs to happen during merging is that the missing constructional form components are added to the transient structure. In other words, for now we can define the merge operation to be the set theoretic union operation. More formally, we introduce the merge function $U_2(\phi, \phi')$ for merging two component sets ϕ and ϕ' into a single component set as:

$$U_2(\phi, \phi') = \phi \cup \phi'.$$

Clearly, the merge function applies both to meaning and form specifications, since these are all sets.

The notion of merge in FCG should not be confused with Chomsky's merge introduced in [4]. Rather, merging corresponds to unification in traditional unification and constraint based formalisms: both result in extra 'constraints' in the transient linguistic structure. As mentioned in the introduction, in FCG such 'constraints' can also be considered 'seed material'. This will become relevant when even more complex meaning and form specifications are considered below.

We are now in a position to redefine the production and parsing functions again such that they operate on transient linguistic structures and make use of the merge function U_2. Since these functions now should embody an iterative process of applying all applicable constructions one after the other, this is most easily done through induction. Thus, in case that the constructicon only specifies a single construction c, and with s a transient linguistic structure, we have that:

$$g_\rightarrow(s, \{c\}) = \begin{cases} \{\boxed{s^m \leftrightarrow U_2(s^f, s^f)}\} \text{ if } U_1(c^m, s^m) \neq \emptyset \\ \{s\} \text{ otherwise.} \end{cases} \tag{5}$$

In words, this says that the production function, when given a transient linguistic structure s and a singleton constructicon $\{c\}$ such that the semantic poles s^m and c^m match (according to U_1), produces a transient linguistic structure with a modified syntactic pole that is equal to the union of the syntactic poles of the given transient structure and the construction. This definition is easily extended to larger collections of construction through the following induction step on larger constructicons:

$$g_\rightarrow(s, \{c\} \cup C) = \{g_\rightarrow(s', C) : s' \in g_\rightarrow(s, \{c\})\}.$$

Note that this definition assumes that the order in which constructions are considered does not matter. If it did, then the constructions would not be independent and a search would be required over possible orderings. This possibility is further discussed in section 7.

Note also that, whereas the processing functions are defined by induction, this does not mean that they are also recursive functions. In recent years, there has been quite a debate over whether or not humans are the only species that possess the capacity for recursion, so this issue is a matter of some importance [7, 9, 11, 12]. However, there is an important difference between genuinely recursive functions and merely tail recursive representations of iterative functions [1]. Only genuinely recursive functions also require a recursive implementation for computation. Compositional language processing as defined in this section does not.

In any case, we now have that:

$$g_{\rightarrow}(s_{\mu}, \{c_1\}) = \left\{ \boxed{\mu \leftrightarrow \{f_1(?a)\}} \right\},$$
$$g_{\rightarrow}(s_{\mu}, \{c_2\}) = \left\{ \boxed{\mu \leftrightarrow \{f_2(?b)\}} \right\}.$$

And:

$$g_{\rightarrow}(s_{\mu}, \{c_1, c_2\}) = \left\{ \boxed{\mu \leftrightarrow \phi(?a, ?b)} \right\}.$$

This accomplishes a compositional encoding of μ into ϕ. The parsing function is defined in a similar fashion.

This concludes our discussion of bi-directional processing in the case that meaning and form specifications are sets of 'independent' meaning or form components. The fact that the components are 'independent' means for instance that they are not allowed to share any of their skolem constants. In summary, in this case a *set* unification function U_1 is required for matching sets of components and for testing which constructions apply. This corresponds to the 'lookup' step in processing. Furthermore, actually applying matching constructions in a compositional fashion amounts to adding their constructional components to a transient linguistic structure through a merge operation. The transient linguistic structure functions to keep 'intermediate results' (partial productions or parses) during processing.

6 Constituent Structure and Hierarchy

So far, we have been considering the English example phrase "natural line" without bothering about the fact that the two lexemes in it are ordered. Indeed, consider again the form specification $\phi(?a, ?b)$ from the previous section, which is the result of applying the production function to the meaning $\mu(x, y)$:

$\phi(?a, ?b) = \{f_1(?a), f_2(?b)\} = \{$`(lexeme ?a "line")`, `(lexeme ?b "natural")`$\}$

Something is missing in this specification. In fact, it fails to specify the order of lexemes, so that both "natural line" and "line natural" are consistent with it. As mentioned already in section 3, a more correct specification should specify this order, like ψ below.

$$\begin{aligned}\psi &= \{f_1(?a), f_2(?b), f_3(?b, ?a)\} \\ &= \{\texttt{(lexeme ?a "line"), (lexeme ?b "natural), (meets ?b ?a)}\}.\end{aligned}$$

Here, the component $f_3(?b, ?a)$ or '`(meets ?b ?a)`' represents a further constraint on the variables $?a$ and $?b$ besides the fact that they stand for the lexemes '`line`' and '`natural`'. This component specifies that these lexemes should be rendered in a particular order. Only the phrase "natural line" is consistent with this specification, whereas the alternative "line natural" is ruled out.

Similarly, nothing in the meaning specification $\mu(x, y) = \{m_1(x), m_2(y)\}$ specifies that it is actually the line (represented by x) that has the property of being natural or, in FrameNet terminology, that fulfills the '`[entity]`' Frame Element relation of the '[natural]' frame. The phrase "natural line" therefore expresses not $\mu(x, y)$, but the more elaborate meaning specification η below.

$$\begin{aligned}\eta &= \{m_1(x), m_2(y), m_3(x, y)\} \\ &= \{\texttt{(frame x [line]), (frame y [natural]),} \\ &\qquad \texttt{(fe-relation y x [entity])}\}.\end{aligned} \tag{6}$$

As on the form side in ψ, components are no longer independent in η: some of the skolem constants are shared between them. In the remainder of this section we investigate what modifications need to be made to the machinery developed so far so that it becomes possible to deal with such dependencies.

We start by recalling that, in English, the order of words in the example phrase "natural line" is licensed by a modifier-head construction for combining adjectives with nouns. On the meaning side, this construction links the corresponding meaning components by specifying an `[entity]` frame element relation. How could such a construction be defined with the representational tools developed so far? The answer, unfortunately, is that it can not.

To see this, consider that it should apply to the transient structure obtained from applying the lexical constructions c_1 and c_2 to the extended meaning specification η. Since

$$U_1(c_1^m(?x, ?a), \eta(x, y)) = \{[?x/x]\} \quad \text{and} \quad U_1(c_2^m(?y, ?b), \eta(x, y)) = \{[?y/y]\},$$

this transient structure looks as follows:

$$g_{\rightarrow}(\eta, \{c_1, c_2\}) = \left\{ \boxed{\begin{array}{c}\{m_1(x), m_2(y), m_3(x, y)\} \\ \leftrightarrow \\ \{f_1(?a), f_2(?b)\}\end{array}} \right\}.$$

The correspondence that exists between the meaning component 'm_1' and the form component 'f_1' (as it is captured in construction c_1) is not reflected in the

above transient linguistic structure. Similarly, nothing in the structure shown above indicates that there is a connection between the meaning component m_2 and the form component f_2. However, the modifier head construction needs access to this information.

Once again we will extend our representational toolkit. PArticularly, we will make the poles of constructions and of the transient linguistic structure to be *feature structures* instead of plain sets. In other words, constructions (and the transient structure) now become *coupled feature structures* (or *cfs*'s in short).

6.1 Coupled Feature Structures

In general, a feature structure is a set of named *units*. Each unit represents a lexical or phrasal constituent. In order to find the form components that are associated with the meaning components in a unit, it suffices to look in the corresponding unit (the unit with the same name) on the syntactic side. Units are further organized into features. The MEANING feature holds the unit's meaning specification. The FORM feature holds its form specification. Furthermore, like constituents, one unit may be part of another one. This is encoded in the SUB-UNITS feature. Meaning and form constants are kept in the REFERENT feature. This can be summarized as follows using BNF notation:

```
cfs              ::= sem-unit-structure <--> syn-unit-structure
unit-structure   ::= {regular-unit*}
regular-unit     ::= <unit-name,{regular-feature*}>
regular-feature  ::= <feature-name,feature-value>
feature-value    ::= feature-value-element
                     | {feature-value-element*}.
```

The asterisk represents the Kleene star operation, specifying that one or more elements of the type marked with it are possible. So '`regular-unit*`' stands for zero or more regular units. The notation $\langle . \rangle$ denotes an ordered list of elements, i.e. a sequence. Sets, which are unordered lists of elements, are denoted with curly brackets as usual.

Here is an example. The cfs that holds the meaning specification η and nothing else appears as follows:

$$s_\eta = \boxed{\{\langle u_0, \langle \text{MEANING}, \eta \rangle\rangle\} \leftrightarrow \{\}} \tag{7}$$

This cfs has one unit named u_0 (not to be confused with the component unification function U_0). In turn, this unit has the specification η as value for its MEANING feature.

By using feature structures instead of plain sets for representing the poles of constructions and of the transient structure, it becomes possible to capture hierarchical constituent structure relations in them. For instance, as will be explained shortly, the transient linguistic structure after applying the constructions

c_1 and c_2 to the coupled feature structure s_η above will look as below:

$$s_{12} = g_{\rightarrow}(s_\eta, \{c_1, c_2\})$$

$$= \boxed{\left\{\begin{array}{ll} \langle u_0, & \langle \textsc{subunits}, \{u_1, u_2\}\rangle, \\ & \langle \textsc{meaning}, \{m_3(x,y)\}\rangle, \\ \langle u_1, & \langle \textsc{meaning}, \{m_1(x)\}\rangle, \\ & \langle \textsc{referent}, x\rangle\rangle, \\ \langle u_2, & \langle \textsc{meaning}, \{m_2(y)\}\rangle, \\ & \langle \textsc{referent}, y\rangle\rangle \end{array}\right\} \leftrightarrow \left\{\begin{array}{ll} \langle u_0, & \langle \textsc{subunits}, \{u_1, u_2\}\rangle, \\ & \langle \textsc{form}, \{f_3(?b, ?a)\}\rangle, \\ \langle u_1, & \langle \textsc{form}, \{f_1(?a)\}\rangle, \\ & \langle \textsc{referent}, ?a\rangle\rangle, \\ \langle u_2, & \langle \textsc{form}, \{f_2(?b)\}\rangle, \\ & \langle \textsc{referent}, ?b\rangle\rangle \end{array}\right\}} \quad (8)$$

Among other things, this cfs captures precisely the fact that the components m_1 and f_1 'belong together' since they are part of the same unit u_1 (although they are in different poles). Importantly, the constituent structure in this cfs is a *result* of applying the constructions c_1 and c_2 to the initial linguistic structure s_η (which itself only has a flat constituent structure). So it is indeed the construction c_1 that defines the meaning-form pair $m_1(?x) \leftrightarrow f_1(?a)$ as a constituent, but now this is also reflected in the transient linguistic structure.

In FCG, the correspondence between constructions and constituents is not merely one-to-one. In particular, FCG expects that constructions *explicitly specify the way in which they change the constituent structure of the transient linguistic structure* (i.e. add new units and/or move components around etc.) Thus, the modifier head construction we aim to define will have to specify that a new unit can be created that groups an existing adjective unit with an existing noun unit in a specific order. In the next section, it is explained how such manipulations of the transient structure can be specified in constructions through the usage of TAGs and 'J-units'.

6.2 Tags and J-units

Following the developments in the previous section, we now define constructions as an extension over coupled feature structures as follows:

```
cxn          ::= sem-pole <--> syn-pole
pole         ::= {unit*}
unit         ::= regular-unit | J-unit |
                 <unit-name, {[regular-feature | tag]*}>
tag          ::= (TAG [tag-variable regular-feature]*)
J-unit       ::= <(J . ?focus ?parent {?child*}),
                           {[tag-variable | regular-feature]*}>
```

All coupled feature structures are thus constructions, but not the other way around: constructions, in addition, may contain TAGs and *J-units*. J-units are easily recognized by the fact that they do not have a proper name. Instead their name is a list of the form '(J . ?focus-unit ?parent-unit child*)'. The first element in this list is called the 'J-operator'. The dot after the J-operator in

the definitions above indicates that the arguments that follow the operator are optional.

J-units are ignored during the matching or 'lookup' process. The workings of the J-operator during merging is most easily explained with an example. Consider therefore again the lexical construction c_1 that couples the meaning component m_1 to the form component f_1. In our new notation it looks as follows:

$$c_1 = \boxed{\begin{array}{l} \left\{ \begin{array}{ll} \langle ?u, & (\text{TAG } \texttt{?m } \langle \text{MEANING}, \{m_1(?x)\}\rangle)\rangle \\ \langle (\texttt{J ?new } ?u), & \{\texttt{?m}, \langle \text{REFERENT}, ?x\rangle\}\rangle \end{array} \right\} \\ \qquad\qquad \leftrightarrow \\ \left\{ \begin{array}{ll} \langle ?u, & (\text{TAG } \texttt{?f } \langle \text{FORM}, \{f_1(?a)\}\rangle)\rangle \\ \langle (\texttt{J ?new } ?u), & \{\texttt{?m}, \langle \text{REFERENT}, ?a\rangle\}\rangle \end{array} \right\} \end{array}} \tag{9}$$

This construction specifies one J-unit in both poles. The focus of the J-units is the variable '?new'. Their parent is the variable '?u', which also is the name of a regular unit in the construction. The fact that the focus variable does not refer to a regular unit indicates that a new subunit is introduced by this construction in the transient linguistic structure during merge. This unit will be made a subunit of the parent unit. On the semantic side, the new unit will have a REFERENT feature with value (the binding value of) $?x$. It will also have a MEANING feature with value $\{m_1(?x)\}$, as specified by the TAG variable $?m$. On the syntactic side, it will have a REFERENT feature with value $?a$ and a FORM feature with value $\{f_1(?a)$. The latter is specified by the TAG variable $?f$. By definition, when meaning and form components are tagged and specified in J-units as described, they will be removed from the unit in which they originally occurred. This explains why, in structure (8) above, which is the result of applying constructions c_1 and c_2 to the initial structure (7), the meaning and form components covered by these two constructions only occur in the newly created units u_1 and u_2. The process is explained in more detail in the following.

6.3 Unit structure matching and merging

We now specify the process explained in the previous section in more detail. We first consider the matching or comparing of unit structures for lookup.

Recall the definition of coupled feature structures in BNF notation. In particular, notice that a unit structure is a *set* of units. It follows that unit structures can be matched with the set matching function U_1, provided that the matching of the set elements, which are units now, is properly defined. Two units match when their names match and when their *sets* of features match. The latter is a case of set matching again, and calls for a proper definition of feature matching. It is clear that, continuing in this way and following the BNF definition of coupled feature structure, matching of unit structures can fully be defined in terms of symbol matching (for the names of units and regular features) and the set and component matching functions U_1 and U_0.

It remains to specify how J-units and TAGs affect the matching process. As mentioned, J-units are simply ignored during matching. Their purpose is to specify how a construction modifies constituent structure, *given* that the (appropriate

pole of) the construction matches with the (appropriate pole of) the transient linguistic structure.

We now turn to tags. On the one hand, the purpose of tags is to mark parts of the transient structure for later reference. Thus, when the semantic pole of construction c_1 is matched against the initial unit structure s_η, the tag variable $?m$ receives the binding $\langle \text{MEANING}, \{m_1(x)\}\rangle$. On the other hand, their purpose is to allow to move components between units during merge. So we turn to the merging of unit structure. Ignoring the details having to do with the bookkeeping of names of units and features etc., merging regular parts of unit structures boils down to taking their union. J-units additionally change constituent structure and extract tagged components. For simplicity, we continue to use the symbol 'U_2' for the extended merge operation that accomplishes all this. Below is given the result of merging the semantic pole of construction c_1 with the initial structure s_η^m:

$$U_2\left(c_1^m, s_\eta^m\right) = \left\{ \begin{array}{l} \langle u_0, \langle \text{MEANING}, \{m_2(y), m_3(x,y)\}\rangle\rangle \\ \langle u_1, \langle \text{MEANING}, \{m_1(x)\}\rangle\rangle \end{array} \right\} \tag{10}$$

The resulting structure has two units instead of just one (with the new unit arbitrarily named u_1). As explained, this is due to the J-operator in c_1^m. The tagged part of meaning $m_1(x)$ was extracted from the original unit and moved to the new unit.

We can now define new production and parsing functions that work with coupled feature structures. The main difference with the previous versions is that now merging should occur for both poles, regardless of the processing mode. The reason for this is that both constructional poles may contain J-units, implying that both sides of the transient linguistic structure may change. Thus we have:

$$g_\rightarrow(t, \{c\}) = \begin{cases} U_2(t^m, c^m) \leftrightarrow U_2(t^f, c^f) \text{ if } U_1(c^m, t^m) \neq \emptyset \\ t \text{ otherwise.} \end{cases}$$

and

$$g_\rightarrow(t, C \cup \{c\}) = g_\rightarrow(g_\rightarrow(t, \{c\}), C).$$

A new parse function is readily defined in a similar way. Note that, as before, these definitions assume that the order in which constructions are applied is arbitrary. This is not a valid assumption in general, and the consequences of relaxing this assumption are investigated in section 7. Note also that, since it is now possible that constructions specify changes to the transient linguistic structure on the side that was previously only matched, we have entered the domain of context sensitive language processing.

6.4 Grammatical Constructions

We now finish the discussion on constituent structure and hierarchy by working towards a modifier-head construction for processing the phrase "natural line".

Consider again construction c_1:

$$c_1 = \boxed{\begin{array}{c} \left\{\begin{array}{ll} \langle ?u, & (\text{TAG } \texttt{?m} \ \langle \text{MEANING}, \{m_1(?x)\}\rangle)\rangle \\ \langle (\texttt{J ?new } ?u), & \{\texttt{?m}, \langle \text{REFERENT}, ?x\rangle\}\rangle \end{array}\right\} \\ \leftrightarrow \\ \left\{\begin{array}{ll} \langle ?u, & (\text{TAG } \texttt{?f} \ \langle \text{FORM}, \{f_1(?a)\}\rangle)\rangle \\ \langle (\texttt{J ?new } ?u), & \{\texttt{?m}, \langle \text{REFERENT}, ?a\rangle\}\rangle \end{array}\right\} \end{array}} \tag{11}$$

Its semantic side matches that of the initial cfs s_η repeated below.

$$s_\eta = \boxed{\{\langle u_0, \langle \text{MEANING}, \{m_1(x), m_2(y), m_3(x,y)\}\rangle\} \leftrightarrow \{\}} \tag{12}$$

Indeed:

$$U_1(c_1^m, s_\eta^m) = [?u/u_0, ?x/x, ?m/\langle \text{MEANING}, \{m_1(x)\}\rangle]. \tag{13}$$

The focus variable '?new' of the J-unit in the construction c_1 does not receive a binding according to the above expression. As explained, merging this construction into the structure s_η therefore results in an additional unit u_1:

$$s_1 = g_\rightarrow(s_\eta, \{c_1\}) = \boxed{\begin{array}{c} \left\{\begin{array}{ll} \langle u_0, & \langle \text{SUBUNITS}, \{u_1\}\rangle, \\ & \langle \text{MEANING}, \{m_2(y), m_3(x,y)\}\rangle, \\ \langle u_1, & \langle \text{MEANING}, \{m_1(x)\}\rangle, \\ & \langle \text{REFERENT}, x\rangle\rangle \end{array}\right\} \\ \leftrightarrow \\ \left\{\begin{array}{ll} \langle u_0, & \langle \text{SUBUNITS}, \{u_1\}\rangle\rangle, \\ \langle u_1, & \langle \text{FORM}, \{f_1(?a)\}\rangle, \\ & \langle \text{REFERENT}, ?a\rangle\rangle \end{array}\right\} \end{array}} \tag{14}$$

The additional unit contains the tagged part of meaning that was previously in the top unit u_0. Now consider c_2 below.

$$c_2 = \boxed{\begin{array}{c} \left\{\begin{array}{ll} \langle ?u, & (\text{TAG } \texttt{?m} \ \langle \text{MEANING}, \{m_2(?y)\}\rangle)\rangle \\ \langle (\texttt{J ?new } ?u), & \{\texttt{?m}, \langle \text{REFERENT}, ?y\rangle\}\rangle \end{array}\right\} \\ \leftrightarrow \\ \left\{\begin{array}{ll} \langle ?u, & (\text{TAG } \texttt{?f} \ \langle \text{FORM}, \{f_2(?b)\}\rangle)\rangle \\ \langle (\texttt{J ?new } ?u), & \{\texttt{?m}, \langle \text{REFERENT}, ?b\rangle\}\rangle \end{array}\right\} \end{array}} \tag{15}$$

We have that

$$U_1(c_2^m, s_1) = \{[?u/u_0, ?y/y, ?m/\langle \text{MEANING}, \{m_2(y)\}\rangle]\},$$

and

$$U_2(c_2^m, s_1^m) = s_{12}^m \quad , \quad U_2(c_2^f, s_1^f) = s_{12}^f,$$

with s_{12} as before:

$$s_{12} = g_{\rightarrow}(s_{\eta}, \{c_1, c_2\})$$

$$= \boxed{\left\{\begin{array}{ll} \langle u_0, & \langle \textsc{subunits}, \{u_1, u_2\}\rangle, \\ & \langle \textsc{meaning}, \{m_3(x,y)\}\rangle, \\ \langle u_1, & \langle \textsc{meaning}, \{m_1(x)\}\rangle, \\ & \langle \textsc{referent}, x\rangle\rangle, \\ \langle u_2, & \langle \textsc{meaning}, \{m_2(y)\}\rangle, \\ & \langle \textsc{referent}, y\rangle\rangle \end{array}\right\} \leftrightarrow \left\{\begin{array}{ll} \langle u_0, & \langle \textsc{subunits}, \{u_1, u_2\}\rangle, \\ & \langle \textsc{form}, \{f_3(?b, ?a)\}\rangle, \\ \langle u_1, & \langle \textsc{form}, \{f_1(?a)\}\rangle, \\ & \langle \textsc{referent}, ?a\rangle\rangle, \\ \langle u_2, & \langle \textsc{form}, \{f_2(?b)\}\rangle, \\ & \langle \textsc{referent}, ?b\rangle\rangle \end{array}\right\}} \quad (16)$$

This cfs explicitly couples the meanings $m_1(x)$ and $m_2(y)$ to the forms $f_1(?a)$ and $f_2(?b)$ respectively, and this way provides the necessary information required to express the remaining component $m_3(x, y)$. We are finally in a position to specify the modifier head construction c_3 that expresses this component:

$$c_3 = \boxed{\begin{array}{c} \left\{\begin{array}{ll} \langle ?u_0, & \langle \textsc{subunits}, \{?u_1, ?u_2\}\rangle, \\ & (\textsc{tag}\ \texttt{?m}\ \langle \textsc{meaning}, \{m_3(?x, ?y)\}\rangle), \\ \langle ?u_1, & \langle \textsc{referent}, ?x\rangle\rangle, \\ \langle ?u_2, & \langle \textsc{referent}, ?y\rangle\rangle, \\ \langle (\texttt{J}\ ?u_2\ ?u1), & \emptyset\rangle, \\ \langle (\texttt{J}\ ?u_1), & \{\texttt{?m}\}\rangle \end{array}\right\} \\ \leftrightarrow \\ \left\{\begin{array}{ll} \langle ?u_0, & \langle \textsc{subunits}, \{?u_1, ?u_2\}\rangle, \\ & (\textsc{tag}\ \texttt{?f}\ \langle \textsc{form}, \{f_3(?b, ?a)\}\rangle), \\ \langle ?u_1, & \langle \textsc{referent}, ?a\rangle\rangle, \\ \langle ?u_2, & \langle \textsc{referent}, ?b\rangle\rangle, \\ \langle (\texttt{J}\ ?u_2\ ?u1), & \emptyset\rangle, \\ \langle (\texttt{J}\ ?u_1), & \{\texttt{?f}\}\rangle \end{array}\right\} \end{array}}$$

This construction further transforms structure s_{12} to structure s_{123}:

$$s_{123} = g_{\rightarrow}(s_{\eta}, \{c_1, c_2, c_3\})$$

$$= \boxed{\begin{array}{l} \left\{\begin{array}{ll} \langle u_0, & \langle \textsc{subunits}, \{u_1\}\rangle \\ \langle u_1, & \langle \textsc{subunits}, \{u_2\}\rangle \\ & \langle \textsc{meaning}, \{m_1(?x), m_3(?x, ?y)\}\rangle \\ & \langle \textsc{referent}, ?x\rangle\rangle \\ \langle u_2, & \langle \textsc{meaning}, \{m_2(?y)\}\rangle \\ & \langle \textsc{referent}, ?y\rangle\rangle \end{array}\right\} \\ \leftrightarrow \left\{\begin{array}{ll} \langle u_0, & \langle \textsc{subunits}, \{u_1\}\rangle \\ \langle u_1, & \langle \textsc{subunits}, \{u_2\}\rangle \\ & \langle \textsc{form}, \{f_1(?a), f_3(?b, ?a)\}\rangle \\ & \langle \textsc{referent}, ?a\rangle\rangle \\ \langle u_2, & \langle \textsc{form}, \{f_2(?b)\}\rangle \\ & \langle \textsc{referent}, ?b\rangle\rangle \end{array}\right\} \end{array}} \quad (17)$$

The construction c_3 enforces a head-complement relation between its head and complement units $?u_1$ and $?u_2$ respectively. This is accomplished by the first J-units on both sides, which make unit $?u_2$ a subunit of $?u_1$. The second J-units moves the covered meaning and form components m_3 and f_3 to the head unit. Thus, together, constructions c_1, c_2 and c_3 successfully parse and produce the form specification ψ into the meaning specification η. In other words, we have finally accomplished the parsing and production of the English phrase "natural line".

Note that the usefulness of the REFERENT feature now becomes apparent as well: without it, it would not have been possible to formulate a modifier-head construction c_3, unless through a reference to the particular meaning components that are being combined. In our case these are m_1 and m_2, but a different modifier-head construction would be needed for a different pair of components. This would undermine the whole purpose of the modifier-head construction, which is to separate the expression of a linking relation *between* components from the expression of the components themselves.

This almost concludes our discussion of constituent structure and hierarchy. For completeness, we mention that the processing of natural language in general requires many more refinements to the machinery introduced so far. Consider for instance that our modifier-head construction c_3 for pairing an adjective an a noun actually does not refer to the notions of "adjective" or "noun". In French, and in many other languages, there are even different *types* of adjectives and nouns. For instance, in the phrase "le grand ballon rouge", both "grand" and "rouge" are adjectives ("big" and "red" in English respectively), but they combine differently with the noun "ballon". These distinctions are clearly important for language and language processing, and it should be possible to deal with them in FCG. Luckily, the bulk of the machinery required for this is already in place, and no further extensions need to be introduced besides new types of unit features and new types of operators.

In FCG, semantic and syntactic types and categories are usually specified in the SEM-CAT and SYN-CAT features. Next to the MEANING, FORM and REFERENT features, these are the most commonly used features in FCG, although the set of possible features is open ended. Testing for the semantic or syntactic category of a constituent is easily performed during matching by including the type description in the corresponding regular unit in a construction. Most grammatical languages also involve more refined sorts of restrictions on semantic and syntactic categories. Consider for example the sentence (a) below, which is an instance of the Caused Motion construction.

(a) "Joe broke the vase onto the floor."

Golberg notes that the Caused Motion construction only applies if the cause is *not* an instrument [10], rendering sentence (b) below ungrammatical.

(b) *"The hammer broke the vase onto the floor"

As is discussed at length elsewhere, such and other types of restrictions can be specified with special operators.

7 Constructional Dependencies and Search

So far we have ignored any intricacies that might arise from dependencies between constructions. Such dependencies may interfere with processing. To see this, consider once more the example phrase "natural line". It does not matter in what order the lexical constructions c_1 and c_2 for "natural" and "line" are applied, the resulting structure will be the same. But the modifier-head construction only applies after them. In other words, there is a *dependency* between these constructions. It is also possible that two constructions are *conflicting*, for instance synonyms which both cover the same (set of) form components. In this case there will be several possible parses or productions. Finally if, as in the context of language game experiments, constructions are marked with conventionality or preference scores, then some analysis will be more preferable than others. All of these things imply that processing involves search.

7.1 General aspects of processing as search

In general, search involves the exploration of a *search space* of possible analysis (parses or productions) with the goal to find a suitable one. The search space is not directly accessible however: it needs to be constructed through the actual application of constructions, starting from the initial linguistic structure. All constructions that apply to the initial structure give rise to a new (partial) analysis or state in the search space. At each moment during processing, it therefore needs to be decided which analysis to consider further and, if it triggers several constructions, which construction to actually apply to it. This process of repeatedly selecting a previously obtained transient linguistic structure and selecting and applying a construction to it in order to get a modified transient structure is what search is all about. Different search strategies, like breadth first or depth first, merely correspond to different selection strategies, that is, to different strategies for exploring the search space.

In FCG, we are not primarily interested in *all possible* analysis (i.e. the set of all well formed feature structures compatible with the input initial structure). Rather, processing is goal-driven, and the primary aim is to find a *suitable* analysis as fast as possible. What is a suitable analysis depends on the task at hand. During production, one will typically want that all meaning components are expressed, or that the produced phrase is unambiguous. If there are several alternatives, e.g. synonyms, then the most conventional one is preferred etc. During parsing, all form components should be processed, and the parsed meaning specification should 'make sense', e.g. unambiguously determine a referent in the discourse context.

It is often the case that several analysis (parses or productions) are possible that all meet these primary criteria. For instance, an idiom might express a meaning more concisely than a regular analysis. This situation is somewhat similar to the one that arises when both a holistic and a compositional analysis are possible (see sections 4 and 5). Selecting among such alternative analysis requires additional criteria. In human language, such criteria are often related

to frequency and alignment effects: words and constructions that are more common or were used in the nearby past are typically preferred over others. In the following section it is discussed in more detail how the availability of frequency or related constructional scores can be used to guide search such that more desirable analysis are found first before others.

7.2 Optimal Processing

The problem investigated in this section is how to select among a set of available partial analysis for further expansion during processing. To be precise, an analysis consists of an ordered set of constructions that, when applied one after the other to the initial linguistic structure (the input to processing), leads to a modified transient linguistic structure. Obviously, if one of the modified structures meets a given set of primary processing criteria (see section 7.1), then processing stops (a 'solution' is found). If not, however, one of the available analysis and a construction that applies to it are selected. Applying the construction then gives rise to a new, additional analysis, and the process is repeated.

If constructions are marked with a "preference score", for instance reflecting the frequency of usage, then a global score can be calculated for each analysis by combining the scores of the constructions in it. One possibility is to sum all constructional preference scores. However, this introduces a bias towards analysis with many constructions, that is, towards compositional rather than holistic or idiomatic analysis. An alternative is to average over all constructional preference scores. Neither of these approaches take into account that some analysis might be less complete than others (e.g. leave more meaning or form components unanalyzed). As such, an analysis that covers only a small part of all components with highly scored constructions will be preferred over one that covers all components, but with only moderately scored constructions.

The question arises whether there is an objective or otherwise 'optimal' way of combining constructional scores. The answer in general depends on the precise nature of these scores. If they reflect the probability that the construction will "get the message through", that is, is shared between the interlocutors, then one possibility is to optimize the expected communicative success. This makes sense particularly in the context of language evolution experiments, where the aim is precisely to simulate the evolution of an efficient communication system or language between robots, and scores typically reflect a degree of "conventionality" or "sharedness".[4]

So let $\alpha(s_0, c)$ denote the fraction of components in the initial structure s_0 that is covered by a construction c. For example, if $\eta = \{m_1(x), m_2(y), m_3(x, y)\}$ and c_1, c_2 and c_3 are as before, then each of the constructions covers one third of the components in η. The holistic construction h of section 5.1 covers two thirds. Furthermore, let $\beta(c)$ denote the (positive) preference score of the construction c,

[4] Note however that not just any scoring mechanism allows an interpretation in terms of probabilities. For one thing, probabilities must be positive and are subject to normalization constraints etc.

and let A be a (partial) analysis (an ordered set of constructions). An optimistic (or *admissible*) yet informative heuristic score of the analysis A is the *best still achievable score* $\gamma(A)$ defined as follows:

$$\gamma(A) = \sum_{c \in A} \alpha(s_0, c)\beta(c) + (1 - \sum_{c \in A} \alpha(s_0, c))$$

The first term in the above expression is the contribution of the constructions in A, that is, in the set of constructions that already applied, modulated by their proper preference scores. The second part is an optimistic estimate of what can still be achieved for the remaining components not yet covered in the analysis. This term assumes that it will be possible to cover the remaining components with constructions of maximum preference score 1. Considering constructions in order of preference and adjusting the still achievable score accordingly may considerably improve the accuracy of the heuristic, and hence the time and memory required for processing.[5]

In summary, given that a constructional score can be interpreted as the probability that the construction is shared, optimal processing optimizes the expected communicative success. This is particularly relevant within the context of language game experiments, where the goal is to optimize communicative success amongst a population of (simulated) language users, but might also be relevant in relation to frequency effects observed in human language.

7.3 Efficiency Issues and Specialized Techniques

Many other specialized techniques exist for optimizing symbolic language processing. However, many of them rely on assumptions that do not hold in FCG. For instance, a well known optimization technique is chart parsing. In computer science terms, this is a *memoization* technique, which ensures that the same subphrase is never analyzed more than once. Instead, the result is calculated once and 'memoized', so that it can be retrieved again later if needed. Most chart parsing techniques rely on pre-defined and often linear representations of form. FCG adopts a more flexible representation of form that may involve a tree, or even a graph of form components. Nevertheless, some degree of memoization can still be achieved by checking for similarities between different analysis. This can still greatly reduce processing load, particularly during lexical processing where the order in which constructions are applied often does not matter.

Another optimization technique is the usage of *ranks* or *independent construction sets*. For some constructions it is known that they will not apply unless certain other constructions applied (consider again the modifier-head construction for instance), so it makes sense to put them in different constructicons. Considering that increasing the size of the constructicon can quickly lead to

[5] This is not a trivial matter however, since in general it cannot be excluded that a highly scored construction is only triggered at a later stage, for instance because it depends on other constructions.

a combinatorial explosion of the search space, such a 'divide and conquer' approach might considerably reduce processing load. More generally, we can keep a detailed record of all dependencies that exist between constructions and compile them into a constructional dependency network. This might eliminate search altogether. However, if the language changes, constructional dependencies change too, so that a 'one time compilation' approach is not possible and more dynamic programming techniques are required. These issues are further discussed elsewhere, e.g. [18] or [5].

8 Discussion and Conclusion

FCG was developed as a computational framework to support artificial language processing in robotic setups. From this perspective, it is actually surprising that it has grown into a formalism in which even complex phenomena of natural language can be captured. This in itself, in my opinion, makes it worthwhile to compare FCG with other formalisms that were developed for capturing natural language from the start.

Thus FCG, like HPSG [14] or ECG [3], is feature structure and unification based. However, despite these similarities, a detailed comparison is not easily made due to fact that these techniques are sometimes employed for different reasons. As such, whereas unification in HPSG is seen as a falsifiability problem that determines the set of well formed structures, in FCG it is sometimes used as a way to perform a "lookup" of constructions during processing (match), and sometimes, in a different form, as an active step in processing itself (merge). This reflects the fact that the development of FCG was driven primarily by the problem of *language processing* rather than that of *language representation*.

Another distinguishing feature of FCG is that linguistic knowledge is organized into constructions. As in construction grammar, these are meaning form pairings. In FCG, constructions are furthermore routines for (partially) transforming meaning specifications into form specifications during production, and vice versa during parsing. There is thus an inherent duality between two processing modes –parsing and production– that is not usually found in other formalisms. This duality should not be confused with the one that exists between parsing and generation in generative approaches.

With this chapter, we aimed to clarify some of these and other 'peculiarities' by tracing them back in the history of FCG. This history more or less follows the development of processing machinery for processing increasingly complex forms of language. For purely holistic language processing, a simple bi-directional lookup table between meaning and form specifications suffices. Matching then amounts to a simple comparison of patterns (component matching) and merging is not even relevant. For compositional language processing of independent components, a slightly more involved form of matching is required, namely set-matching. Merging then amounts to taking the union of sets. It also becomes necessary to introduce the notion of a transient linguistic structure for capturing intermediate processing results. When components are no longer independent

but form a network, additionally relations between meaning and form components as captured in constructions need to be represented in the transient linguistic structure as well. This is the reason for introducing feature structures in FCG. Matching now becomes a kind of pattern-matching between feature structures. Merging now formally corresponds to the notion of unification of feature structures in other unification-based language formalisms, augmented with special machinery for specifying manipulations to the constituent structure of the transient linguistic structure. Chapters [16] and [5] can be consulted for a further entanglement of differences and commonalities between FCG and other approaches to unification and constraint based language processing.

Acknowledgements

This work was mainly funded by the FWO through the European ComplexityNet project "EvoSym". The author wishes to thank Luc Steels for making the development of FCG possible and for helping to improve this paper.

Bibliography

[1] Abelson, H., Sussman, G.J.: Structure and Interpretation of Computer Programs, Second Edition. MIT Press (1996)
[2] Baker, C.F., Fillmore, C.J., Lowe, J.B.: The Berkeley FrameNet Project. In: Proceedings of the 17th international conference on Computational linguistics. Association for Computational Linguistics, Morristown, NJ, USA (1998)
[3] Bergen, B., Chang, N.: Embodied Construction Grammar in simulation-based language understanding. In: Östman, J.O., Fried, M. (eds.) Construction Grammar(s): Cognitive and Cross-Language Dimensions. Johns Benjamins (2005)
[4] Chomsky, N.: The Minimalist Program. MIT press, Cambridge Ma. (1995)
[5] Ciortuz, L., Saveluc, V.: Fluid Construction Grammar and feature constraints logics. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[6] Croft, W.: Logical and typological arguments for radical construction grammar. In: Ostman, J., Fried, M. (eds.) Construction Grammars: Cognitive grounding and theoretical extensions. John Benjamin Publ Cy., Amsterdam (2005)
[7] De Beule, J.: Compositionality, Hierarchy and Recursion in Language, a Case Study in Fluid Construction Grammar. Ph.D. thesis, VUB Artificial Intelligence Lab (2007)
[8] Fillmore, C.J.: Frame semantics. In: Linguistics in the Morning Calm. pp. 111–137. Seoul (1982)
[9] Gentner, T.Q., Fenn, K.M., Margoliash, D., Nusbaum, H.C.: Recursive syntactic pattern learning by songbirds. Nature 0 (2006)

[10] Goldberg, A.: Constructions: A Construction grammar approach to argument structure. University of Chicago Press, Chicago (1995)
[11] Hauser, M.D., Chomsky, N., Fitch, W.T.: The faculty of language: What is it, who has it, and how did it evolve? Science 298, 1569–1579 (11 2002)
[12] Jackendoff, R., Pinker, S.: The nature of the language faculty and its implications for evolution of language (reply to fitch, hauser, and chomsky). Cognition (September 2005)
[13] Norvig, P.: Paradigms of AI: Case studies in Common Lisp. PWS Publishing Company (1992)
[14] Pollard, C., Sag, I.: Head-driven phrase structure grammar. University of Chicago Press, Center for the Study of Language and Information of Stanford University, Chicago (1994)
[15] Sag, I.A., Wasow, T., Bender, E.M.: Syntactic Theory, A Formal Introduction. CSLI Publications, Leland Stanford Junior University, United States, 2nd edn. (2003)
[16] Sierra, J.: A logic programming approach to parsing and production in Fluid Construction Grammar. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[17] Steels, L., Vogt, P.: Grounding adaptive language games in robotic agents. In: Husbands, P., Harvey, I. (eds.) Proceedings of the Fourth European Conference on Artificial Life (ECAL'97), Complex Adaptive Systems. The MIT Press, Cambridge, MA (1997)
[18] Wellens, P.: Organizing constructions in networks. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)