Notice

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

Sierra, Josefina (2012). A Logic Programming Approach to Parsing and Production in Fluid Construction Grammar. In Luc Steels (Ed.), *Computational Issues in Fluid Construction Grammar*, 239–255. Berlin: Springer.

BibTeX:

```
@incollection{sierra2012logic,
   Author = {Sierra, Josefina},
   Title = {A Logic Programming Approach to Parsing and Production
            in Fluid Construction Grammar},
   Pages = {239--255},
   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 Logic Programming Approach to Parsing and Production in Fluid Construction Grammar

Josefina Sierra Santibáñez

Universidad Politécnica de Cataluña, Barcelona, Spain

Abstract. This paper presents a Logic Programming approach to parsing and production in Fluid Construction Grammar (FCG) [13]. It builds on previous work on the formalisation of FCG in terms of First Order Logic (FOL) concepts, more specifically on the definition of its core inference operations, *unification* and *merge,* in terms of FOL unification and search in the space of a particular set of FOL terms called *structure arrangements.* An implementation of such inference operations based on Logic Programming and Artificial Intelligence techniques such as unification and heuristic search is outlined.

1 Introduction

Fluid Construction Grammar (FCG) [10] is a grammatical formalism implemented in Lisp [6] which incorporates ideas from Construction Grammar [3] and Cognitive Grammar [5].

It has been used in a number of experiments [11, 15] investigating the *symbol grounding problem* [4] in populations of autonomous agents connected to their environment through sensors and actuators. These experiments focus on the study of the evolution and the acquisition of language [9] in particular on the acquisition of grammar and the role of grammar in language grounding, emphasising the communicative function of grammar as well as the relation between grammar and meaning [14].

FCG also draws inspiration from observations of language usage [18], which suggest that natural languages constantly adapt and evolve to cope with new meanings and variations in the behaviour of language users. The experiments themselves are designed to implement and test a constructivist approach to language development [16], in which grammatical constructions are acquired gradually, beginning with concrete linguistic structures based on particular words, from which they are progressively abstracted.

From a computational point of view, FCG is fully operational [1, 17] and it has been used in a considerable number of experiments. However its basic inference operations, unification and merge, are only defined intuitively in the linguistics literature. A formalisation of the unification and merge algorithms used in FCG has been proposed by its developers in [12]. Formal definitions of FCG concepts in terms of First Order Logic and Order-Sorted Feature Constraint Logic have also been presented in [8] and [2]. The present paper outlines an

approach to parsing and production in FCG based on Logic Programming and Artificial Intelligence techniques such as unification and heuristic search.

The rest of the paper is organised as follows. Section 1 describes the representation formalism used in FCG. Section 2 summarises the formal definition of *unification and merge* proposed in [8]. Section 3 illustrates the usefulness of such definition with two examples of construction application. Finally, section 4 presents an outline of an FCG-unification algorithm based on logic programming techniques through an extended example.

2 Representation Formalism

2.1 Semantic and Syntactic Structures

Linguistic and semantic information is represented using *syntactic* and *semantic structures* in FCG. A semantic or syntactic structure consists of a *set of units*, which correspond to lexical items or constituents such as noun phrases or relative clauses. A unit has a name and a number of feature-value pairs. In this paper, we will assume that *semantic units* contain the features *sem-subunits, referent, meaning* and *sem-cat,* in that order; and *syntactic units* the features *syn-subunits, utterance, form* and *syn-cat*. Feature values depend on the type of feature: *referent* and *utterance* have a *single value*, whereas the values of *sem-subunits* and *syn-subunits* are *sets of unit names*. The values of the rest of the features are *sets of facts* about different aspects of the components of a structure: *meaning* is a set of facts which can be used to identify the referent (e.g. its shape, colour, type of entity or event); *semantic categories* describe more abstract aspects of the referent (e.g. its role as the agent, object or recipient in an action); *form* and *syntactic categories* specify different aspects of the utterance, such as its number, part of speech, stem or grammatical role (e.g. subject, predicate or object). The set of facts which may be included in the values of these features is not restricted to those just mentioned but open ended.

We will use *lists* to represent those facts. For example, the fact that unit-2 is a *noun* will be represented by the list `(part-of-speech unit-2 noun)`. This notation allows using First Order Logic variables for syntactic and semantic categories [7]. In particular, we will use a *many-sorted language* with two types: list and atom[1]. We shall also assume that the elements of the lists representing facts are always variables or constants of type atom, and that any symbol preceded by a question mark is a variable.

[1] There is a binary function symbol `cons: Atom` $\times$ `List` $\longrightarrow$ `List` and a constant symbol `NIL` of type `list`, which allow constructing terms of type `list`. For example, the term `(part-of-speech unit-2 noun)` is an abbreviation for the first order logic term `(cons part-of-speech (cons unit-2 (cons noun NIL)))`, where `part-of-speech, unit-2` and `noun` are constant symbols of type `atom`.

2.2 Constructions

In FCG inference is performed applying *constructions* to *source structures.* A *construction* is a pair <*left-pole*> ⇔ <*right-pole*> of *pattern structures* which usually associates a *syntactic pattern* with a *semantic pattern* (see figures 2 and 1). Constructions play therefore the role of *grammar rules* in *construction grammars* [3]. However they not only relate syntactic patterns to semantic ones but also supply information required for parsing and generation which is not included in lexical items, making it possible to construct sentences whose meaning is more than the sum of the meanings of their parts.

Source structures (i.e. semantic and syntactic structures of the type we have described before) constitute the input to parsing and production processes in FCG, whereas constructions are used to add semantic and syntactic information to source structures, that is, to complete missing aspects in these structures, such as the identity of the agent in an event or the subject of a verb.

Formally, the *application of a construction* is a combination of two operations: *Unification,* which is used to check whether a construction is compatible with a source structure; and *merge,* which extends the structure with information contained in the construction [12].

3 Unification and Merge

3.1 Feature-value Unification

Unification for feature-values depends on the type of feature. *First Order Logic unification* can be used to compute *the most general unifier (mgu)* of two features whose values are single terms. However, when feature values are sets of terms (unit names or facts represented by lists of atoms), a number of issues must be taken into account before First Order Logic unification can be applied.

Feature-value Arrangement Let $s = \{t_1, \ldots, t_n\}$ be a feature value of type set of terms (unit names or facts represented by lists of atoms) of a semantic or syntactic source unit. An *m-arrangement* of s is a list $v = (t_{i_1}, \ldots, t_{i_m})$, where $t_{i_j} \in s$ for $j = 1 \ldots m$, and t_{i_j} are all distinct. An m-arrangement is thus a list in which m distinct elements of s are organised in a particular order.

Feature-value Unification Let $s = \{a_1, \ldots, a_n\}$ be a feature value of type set of terms of a *source* unit and $p = (== b_1, \ldots, b_m)$ a feature value of type set of terms of a *pattern* unit[2]. We say that s and p are *FCG-unifiable* if there is an m-arrangement s' of s such that the First Order Logic terms s' and p of type list of terms are unifiable, and we call the most general unifier σ of s' and p an FCG-unifier of s and p.

[2] Note that we use *list* notation (round brackets), rather than *set* notation (curly brackets), for specifying *pattern* feature values of type set of terms. The reason is that we need not consider the m-arrangements of pattern values.

The symbol == in the pattern feature value is used in FCG to indicate that the source feature value should include the pattern feature value, but that they need not be exactly the same set. If the symbol == is omitted from the list representing the pattern feature value, then it is understood that both sets, the source and the pattern, must be equal after unification.

Feature values of type set of terms in FCG patterns may take thus one of the following forms:

1. $(== t_1, \ldots, t_m)$, specifying a set of terms which should be included in the value of a particular feature in a source structure; or
2. $(t_1, \ldots, t_m)$, specifying a set of terms which should be equal to the value of a particular feature in a source structure.

Note that there can be several FCG-unifiers for a pair (s, p) of source and pattern feature values. Because two different arrangements s_1 and s_2 of s might be such that p and s_1 are unifiable, and so are p and s_2, but s_1 and s_2 are not.

3.2 Feature-value Merge

Merge is used to extend a semantic or syntactic source structure with additional information contained in a pattern structure.

If the pattern and source feature values are FCG-unifiable, *merge* is equivalent to unification[3]. However, when the pattern and source are not FCG-unifiable, the source feature value is minimally extended so that its extension and the pattern are unifiable. The source feature value is extended only if this can be done without introducing inconsistencies. Consider the following example:

```
p: ((?unit (sem-cat (== (agent ?e ?a) (entity-type ?a human)))))
s: {(unit (sem-cat {(agent e a) (event-type e motion)}))}
```

The values of the feature `sem-cat` are not unifiable. But if we add the fact `(entity-type ?a human)` to the source value, both feature values can be unified yielding the following extended value, which is the result of merging s with p.

```
s': {(unit (sem-cat {(agent e a) (event-type e motion)
                     (entity-type a human)}))}
```

The steps involved in merging the feature value in source structure s with the feature value in pattern structure p above are:

1. Finding a minimal subset $p_c = \{$(entity-type ?a human)$\}$ of p such that $s \bigcup p_c$ and p are FCG-unifiable, and an FCG-unifier σ of p and $s \bigcup p_c$.
2. Applying $\sigma = \{?a = a, ?e = e\}$ to $s \bigcup p_c$ in order to obtain the extended source feature value $(s \bigcup p_c)\sigma$, which is the result of merging s with p.

In general the result of merging a source feature value s with a pattern feature value p is not unique, because there might be different minimal extensions s' of s such that s' and p are unifiable; and there might be as well different FCG-unifiers for a given extension s' and the pattern p.

[3] In this case, the source feature value is not extended as a result of merging it with the pattern feature value, although some of its variables might be instantiated when an FCG-unifier of both feature values is applied to it.

The Set of Facts Consistency Condition The first step above requires further clarification. Let us consider another example, where s and p denote the source and pattern structures to be merged respectively.

```
p: ((?unit (form  (== (string ?unit car)))
           (syn-cat (== (number ?unit singular)))))
s: {(unit (form  {(string unit cars)})
          (syn-cat {(number unit plural)}))}
```

In this case, s should not be merged with p, because neither the values of the `form` feature nor those of the `syn-cat` feature are consistent with each other. The value of the source feature `syn-cat` and the value of the same feature in the pattern are not unifiable. But the union of the minimal subset of the pattern feature value $p_c = \{$(number ?unit singular)$\}$ and the source feature value $s = \{$(number unit plural)$\}$ leads to a contradiction, once the most general unifier $\sigma = \{$?unit = unit$\}$ is applied to it: the number of a unit cannot be singular and plural at the same time.

$(s \bigcup p_c)\sigma =$ (syn-cat {(number unit plural) (number unit singular)})

In fact, the pattern and source structures above cannot be merged. The minimal subset of the pattern feature value p_c such that $s \bigcup p_c$ and p are FCG-unifiable must satisfy an additional condition which we will call *the set of facts consistency:* the extended source feature value resulting from merging the source with the pattern should not contain any pair of facts $(f\ a_1 \ldots a_n\ u)$ and $(f\ a_1 \ldots a_n\ v)$ such that their elements are all equal but for the last one $(u \neq v)$. The reason for imposing this condition is that a function cannot assign different values to a single tuple of elements, and we are assuming that a fact described by a list such as $(f\ a_1 \ldots a_n\ v)$ represents a statement of the form $f(a_1, \ldots, a_n) = v$, where f denotes a function symbol, $a_1, \ldots, a_n$ its arguments, and v the value that f assigns to $(a_1, \ldots, a_n)$.

In FCG the symbol =1 is used in pattern feature values of type set of terms to indicate that no repetitions are allowed. For example, the pattern of the previous example should be specified as follows in FCG:

```
p: ((?unit (form  (=1 (string ?unit car)))
           (syn-cat (=1 (number ?unit singular)))))
```

We need not use =1, because we assume a *functional interpretation* of lists representing facts and *the set of facts consistency* condition. The reader should be warned that lists representing facts in FCG are interpreted *relationally* and that FCG does not make the set of facts consistency assumption.

Feature-value Merge Let s be a source feature value of type set of terms, p a pattern feature value of the same type, p_c a minimal subset of p such that $s \bigcup p_c$ and p are FCG-unifiable[4], and σ an FCG-unifier of $s \bigcup p_c$ and p. If $(s \bigcup p_c)\sigma$

[4] A subset p_c of a pattern feature-value p is *minimal* with respect to a source feature-value s if no subset p_t of p satisfies that: (1) $p_t \subset p_c$; (2) p and $s \bigcup p_t$ are FCG-unifiable; and (3) $(s \bigcup p_t)\sigma$ is fact set consistent.

satisfies *the set of facts consistency condition,* then the extended feature value $(s \bigcup p_c)\sigma$ is a valid result of merging s with p.

3.3 Unification and Merge for Units

Let $p = (p_{name}\ (f_1\ \bar{u}_1)\ (f_2\ u_2)\ (f_3\ \bar{u}_3)\ (f_4\ \bar{u}_4))$ be a pattern unit, where $f_1, \ldots, f_4$ are the feature names *sem-subunits, referent, meaning* and *sem-cat*, if p is a semantic unit; or the feature names *syn-subunits, utterance, form* and *syn-cat*, if p is a syntactic unit.

Unit Arrangement A *p-arrangement* of a source unit s is a first order logic term of the form $(s_{name}\ (f_1\ \bar{v}_1)\ (f_2\ v_2)\ (f_3\ \bar{v}_3)\ (f_4\bar{v}_4))$, where s_{name} is the name of s; n_1, n_3 and n_4 are the number of elements in $\bar{u}_1, \bar{u}_3$ and $\bar{u}_4$; $\bar{v}_1, \bar{v}_3, \bar{v}_4$ are n_1, n_3 and n_4-arrangements of the values of features f_1, f_3 and f_4 in s, respectively; and v_2 is the value of feature f_2 in s.

A *p-arrangement* of a source unit s is thus a unit obtained from s substituting each of its feature values for arrangements of such feature values with respect to the corresponding feature values in the pattern unit p.

Unit Unification Let p be a pattern unit and s a source unit. We say that s and p are *FCG-unifiable* if there is a p-arrangement s' of s such that the first order logic terms p and s' are unifiable. The most general unifier σ of s' and p is an FCG-unifier of s and p.

Unit Merge Let s be a source unit $(s_{name}\ (f_1\ sv_1)\ (f_2\ sv_2)\ (f_3\ sv_3)\ (f_4\ sv_4))$; p a pattern unit $(p_{name}\ (f_1\ pv_1)\ (f_2\ pv_2)\ (f_3\ pv_3)\ (f_4\ pv_4))$; pv_1^c, pv_3^c and pv_4^c minimal subsets of pv_1, pv_3 and pv_4 such that the extended unit $s^e = (s_{name}\ (f_1\ \ sv_1 \bigcup pv_1^c)\ (f_2\ \ sv_2)\ (f_3\ \ sv_3 \bigcup pv_3^c)\ (f_4\ \ sv_4 \bigcup pv_4^c))$ and the pattern unit p are FCG-unifiable; and σ an FCG-unifier of s^e and p. If every feature value in $s^e\sigma$ satisfies the *set of facts consistency* condition, then $s^e\sigma$ is a valid result of merging s with p.

3.4 Unification and Merge for Structures

Structure arrangement Let $s = \{u_1 \ldots u_n\}$ be a source structure and $p = (== v_1 \ldots v_m)$ a pattern structure. An *m-arrangement* of s is a list of m-units $s' = (u_{i_1}^{v_1}, \ldots, u_{i_m}^{v_m})$, where each $u_{i_j}^{v_j}$ is a v_j-arrangement of some $u_{i_j} \in s$ for $j = 1 \ldots m$, and the u_{i_j} are all distinct.

Structure Unification Let $s = \{u_1 \ldots u_n\}$ be a source structure and $p = (== v_1 \ldots v_m)$ a pattern structure. We say that s and p are *FCG-unifiable* if there is an m-arrangement s' of s such that the first order logic terms s' and p are unifiable. The most general unifier σ of s' and p is an FCG-unifier of s and p.

Structure Merge Let s be a source structure; p a pattern structure ($== v_1 \dots v_m$); p_c a minimal subset of p such that for each unit $u_i \in s \bigcup p_c$ $i = 1 \dots n$ there is a unit u_i^e which is either equal to u_i or an extension of u_i with respect to a unique unit $v_j \in p$, such that the extended structure $s^e = \{u_1^e, \dots, u_n^e\}$ and the pattern structure p are FCG-unifiable; and σ an FCG-unifier of s^e and p. If every feature-value in $s^e\sigma$ satisfies the set of facts consistency condition, then $s^e\sigma$ is a valid result of merging s with p.

4 Examples of Construction Application

The semantic and syntactic source structures constructed at an intermediate stage during the parsing process of the sentence *John slides blocks to Mary* are shown in figure 1. These structures result from applying morphological, lexical, semantic categorisation and phrase structure rules (constructions) to an initial structure containing just the words that make up the sentence. *Morphological rules* decompose words into a stem and a set of syntactic categories (e.g. "slides" into a stem "slide" and the categories *verb* and *singular*). *Number* is *grammatical* as opposed to *natural,* because it does not contribute to meaning. *Lexical rules* map the stem of a lexical item into a set of facts specifying its meaning, and natural syntactic categories (e.g. *number* for nouns) into additional meaning. *Semantic categorisation rules* add semantic categories to the semantic structure (e.g. the arguments of "slide" can be mapped into the semantic roles *agent, object* and *recipient* in a *transfer-to-recipient (tr)* event). Finally, *phrase structure rules* relate structural properties of a sentence, such as word order, to syntactic categories, such as *subject, direct object* or *indirect object.*

Note that the variables associated with the referents of semantic units *jo, bl* and *ma,* which represent *John, blocks* and *Mary* respectively, are different from those associated with the roles in the *transfer-to-recipient* event in unit *sl.* Figure 2 shows an example of a construction whose purpose is to ensure that the variables associated with the roles *agent, object* and *recipient* (*ag, obj* and *rec* in unit *?eu*) in a *transfer-to-recipient* (*tr*) event in a semantic structure become equal to the variables associated with the referents of semantic units `?au, ?ou` and `?ru,` which represent the participants in such an event.

Let us see how the construction shown in figure 2 can be applied to the syntactic and semantic structures associated with the sentence *John slides blocks to Mary,* in order to make the variables representing the roles in the *transfer-to-recipient* event equal to those associated with the referents of the units for John, blocks and Mary.

From a computational point of view, *construction application* is a combination of two operations: unification and merge. *Unification* is used to check whether a construction is compatible with a source structure; and *merge* to extend the structure with information contained in the construction.

In our example unification is first applied to the syntactic pattern of the construction and the syntactic source structure, to determine whether the construction can be used to extend the structure. In this case both structures are

unifiable. Then the unifier built during this process {?su=u, ?eu=sl, ?au=jo, ?tu=t, ?ou=bl, ?ru=ma} is applied to the semantic pattern of the construction and the semantic source structure. Next, the semantic source structure is merged with the semantic pattern of the construction.

If the semantic source structure and the pattern are unifiable, merge is equivalent to applying one of their FCG-unifiers to the semantic source structure. In our example they are unifiable. Therefore the result of merging the semantic source structure with the semantic pattern is obtained applying the unifier τ = {?s=?e, ?j=?a, ?b=?o, ?m=?r} to the semantic structure. As a consequence of this, the variables associated with the roles agent, object and recipient in the transfer to recipient event become equal to those associated with the referents of the units representing John, blocks and Mary in the semantic structure.

```
{(u   (sem-sub {sl jo bl ma}))

 (sl  (referent ?s)
      (meaning {(act ?s slide) (arg1 ?s ?a)
                (arg2 ?s ?o) (arg3 ?s ?r)})
      (sem-cat {(ev-type ?s tr) (ag ?s ?a)
                (obj ?s ?o) (rec ?s ?r)}))

 (jo  (referent ?j)
      (meaning {(entity-type ?j person)
                (count ?j one)}))

 (bl  (referent ?b)
      (meaning {(entity-type ?b block)
                (count ?b several)}))

 (ma  (referent ?m)
      (meaning {(entity-type ?m person)
                (count ?m one)})) }
```

```
{(u   (syn-sub {sl jo bl t ma})
      (form    {(order u (jo sl bl t ma))})
      (syn-cat {SVOtoO-sentence}))
 (sl  (form    {(string sl slides)})
      (syn-cat {(stem sl slide)
                (numb gram sl singular)
                (speech-part sl verb)
                (role sl pred)}))
 (t   (form    {(string t to)})
      (syn-cat {(speech-part t prep)}))
 (jo  (form    {(string jo John)})
      (syn-cat {(stem jo john),
                (numb nat jo singular)
                (speech-part jo noun))
                (role jo subject)}))
 (bl  (form    {(string bl blocks)})
      (syn-cat {(stem bl block)
                (numb nat bl plural)
                (speech-part bl noun))
                (role bl dir-obj)}))
 (ma  (form    {(string ma Mary)})
      (syn-cat {(stem ma mary)
                (numb nat ma singular)
                (speech-part ma noun)
                (role ma ind-obj)})) }
```

Fig. 1. Semantic (left) and syntactic (right) source structures built at an intermediate stage during the parsing process of the sentence *John slides blocks to Mary.*

```
(=
 (?su (sem-sub (= ?eu ?au ?ou ?ru)))

 (?eu (referent ?e)
      (sem-cat (= (ev-type ?e tr) (ag ?e ?a)
                  (obj ?e ?o) (rec ?e ?r))))
 (?au (referent ?a))
 (?ou (referent ?o))
 (?ru (referent ?r)))
```

```
(=
 (?su (syn-sub (= ?eu ?au ?tu ?ou ?ru))
      (syn-cat (= SVOtoO-sentence)))
 (?eu (syn-cat (= (role ?eu pred))))

 (?tu (form    (= (string ?tu to))))
 (?au (syn-cat (= (role ?au subject))))
 (?ou (syn-cat (= (role ?ou dir-obj))))
 (?ru (syn-cat (= (role ?ru ind-obj)))))
```

Fig. 2. Construction which associates a *transfer-to-recipient (tr)* semantic pattern structure (left) with a *Subject + Verb + Dir-Object + to + Indir-Object (SVOtoO)* syntactic pattern structure (right). Features whose values are the empty set or variables which appear only once in the construction are omitted.

However, unifiability is not a necessary requirement for merge. If source and pattern are not unifiable, the source structure might still be merged with the pattern, provided it can be minimally extended so that its extension and the pattern are unifiable, and the result of merge does not violate the set of facts consistency condition.

Let us see an example of construction application where merge cannot be reduced to unification. Figure 3 shows a morphological construction which decomposes the word "slides" into a stem and a number of syntactic categories. We apply this construction to the source structure in figure 4. First, unification is applied to the left pattern of the construction and the source structure (see figure 5), to determine whether the construction can be used to extend the structure. Then the unifier σ constructed during this process is applied to the right pattern of the construction and the source structure.

```
(= (?s (form (= (string ?s slides))))) | (= (?s (form     (= (stem ?s slide)))
                                       |       (syn-cat (= (number ?s singular)
                                       |                  (speech-part ?s verb)))))
```

Fig. 3. Construction which decomposes the word "slides" into a stem and a number of syntactic categories.

```
{ (sl (syn-sub {})
      (utter     ?u2)
      (form      {(string sl slides)})
      (syn-cat   ?sc2)) }
```

Fig. 4. Extended form of a syntactic source structure containing a syntactic unit associated with the word "slides" at an initial stage during the parsing process.

```
(= (?s (syn-sub {})                    | { (sl (syn-sub {})
       (utter     ?u1)                 |       (utter     ?u2)
       (form      (= (string ?s slides))) |       (form      {(string sl slides)})
       (syn-cat   ?sc1)) )             |       (syn-cat   ?sc2)) }
```

Fig. 5. Unification is applied to the left pattern of the construction and the source structure, yielding the unifier $\sigma = \{?s = sl, ?u1 = ?u2, ?sc1 = ?sc2\}$.

Next, the source structure is merged with the right pattern of the construction. Given that the pattern and the source structure are not unifiable, the source structure is minimally extended so that its extension and the pattern are unifiable. In particular, the value of feature *form* in the source structure is extended with the subset {(stem sl slide)} of the value of the same feature in the pattern (see figure 6).

Finally, the extended source structure and the right pattern of the construction are unified, and the unifier $\tau = \{?sc2 = \{(number\ sl\ singular)\ (speech\text{-}part\ sl\ verb)\}\}$ constructed during this step is applied to the extended source structure in order to obtain the result of merging the source structure with the right pattern of the construction (see figure 7).

```
{ (sl (syn-sub {})                  |  (= (sl (syn-sub {})
      (utter    ?u2)                |         (utter    ?u2)
      (form     {(string sl slides) |         (form     (= (stem sl slide)))
                (stem sl slide)})   |
      (syn-cat  ?sc2)) }            |         (syn-cat  (= (number sl singular)
                                    |                   (speech-part sl verb)))))
```

Fig. 6. Unifier σ is applied to the right pattern of the construction, and the source structure is minimally extended so that it unifies with the pattern in the sense of FCG.

```
{ (sl (syn-sub {})
      (utter    ?u2)
      (form     {(string sl slides) (stem sl slide)})
      (syn-cat  {(number sl singular) (speech-part sl verb)})) }
```

Fig. 7. Result of merging the source structure with the instantiated right pattern of the construction.

5 Outline of an FCG-Unification Algorithm

Let $p = (== v_1 \dots v_m)$ be a pattern structure and $s = \{u_1 \dots u_n\}$ a source structure. For each unit v_i in p we define the set $C_i = \{(u_j^k, \sigma_j^k) \mid j = 1 \dots n,\ k = 1 \dots n_j\}$, where u_j^k is a v_i-arrangement of unit u_j in s such that u_j^k and v_i are unifiable in the sense of First Order Logic and $\sigma_j^k = mgu(v_i, u_j^k)$.

For example, let $p = (== v_1 \dots v_6)$ be the syntactic pattern structure in figure 2 and $s = \{u_1 \dots u_6\}$ the syntactic source structure in figure 1. In order to construct C_2, we first try to unify unit v_2 (i.e. *?eu*) in the pattern and unit u_1 (i.e. u) in the source. Units in figures 1 and 2 appear in abbreviated form, where features which do not contain relevant values are omitted. Unification requires using the extended form of these units, which is shown below.

```
(?eu (syn-sub ())                   |  (u (syn-sub {sl jo bl t ma})
     (utter    ?u1)                 |     (utter    ?u2)
     (form     ?f )                 |     (form     {(order u (jo sl bl t ma))})
     (syn-cat  (= (role ?eu pred)))) |     (syn-cat  {SVOtoO-sentence}))
```

In order to unify two units, it is necessary to unify their feature values, and it is clear that the values of feature *syn-cat* are not FCG-unifiable: there is no 1-arrangement of {SVOtoO-sentence} which can be made equal to ((role ?eu pre)). Therefore units v_2 and u_1 are not unifiable.

We consider now the pair of units v_2 (i.e. *?eu* in the pattern) and u_2 (*sl* in the source). The extended form of these units is shown below.

```
(?eu (syn-sub ())                   |  (sl (syn-sub {})
     (utter    ?u1)                 |     (utter    ?u2)
     (form     ?f)                  |     (form     {(string sl slides)})
     (syn-cat  (= (role ?eu pred)))) |     (syn-cat  {(stem sl slide)
                                     |                (numb gram sl singular)
                                     |                (speech-part sl verb)
                                     |                (role sl pred)}))
```

The values of features *syn-sub, utter* and *form* in units *sl* and *?eu* are unifiable. The value of feature *syn-cat* has four 1-arrangements, however only one of them,

((role sl pred)), satisfies the unifiability condition. It is easy to check that unit ?*eu* does not unify with the rest of the units in the source structure. Therefore, set C_2 consists only of the pair (u_2^1, σ_2^1), where σ_2^1 is the unifier {?eu = sl, ?u1 = ?u2, ?f = ((string sl slides))} and u_2^1 the ?*eu*-arrangement of unit u_2 (i.e. *sl*) shown below.

```
(?eu (syn-sub ())                    | (sl (syn-sub ())
     (utter    ?u1)                  |     (utter    ?u2)
     (form     ?f)                   |     (form     ((string sl slides)))
     (syn-cat  (= (role ?eu pred)))) |     (syn-cat  ((role sl pred))))
```

The same reasoning can be applied to units v_3, v_4, v_5 and v_6 (i.e. ?*tu*, ?*au*, ?*ou* and ?*ru*) in the pattern. Only one arrangement of unit u_3 (i.e. *t*) and unit v_3 (i.e. ?*tu*) are unifiable, and the same happens to the pairs of units (?*au*, *jo*), (?*ou*, *bl*) and (?*ru*, *ma*).

Unit v_1 (i.e. ?*su*) is more interesting though, because several v_1-arrangements of source unit u_1 (i.e. *u*) satisfy the unifiability condition. In fact, the set C_1 contains 120 pairs of the form (v_1-arrangement, unifier), one for each permutation of the set {*sl jo bl t ma*}. We just show two of them.

The unifier σ_1^1 = {?su = u, ?eu = sl, ?au = jo, ?tu = t, ?ou = bl, ?ru = ma, ?u1 = ?u2, ?f = ((order u (jo sl bl t ma)))} corresponds to u_1^1, the ?*su*-arrangement of unit u_1 (i.e. *u*) shown below.

```
(?su (syn-sub (= ?eu ?au ?tu ?ou ?ru)) | (u (syn-sub (sl jo t bl ma))
     (utter    ?u1)                    |    (utter    ?u2)
     (form     ?f)                     |    (form     ((order u (jo sl bl t ma))))
     (syn-cat  (= SVOtoO-sentence)))   |    (syn-cat  (SVOtoO-sentence)))
```

The unifier σ_1^2 = {?su = u, ?eu = sl, ?au = jo, ?tu = t, ?ou = ma, ?ru = bl, ?u1 = ?u2, ?f = ((order u (jo sl bl t ma)))} corresponds to u_1^2, the ?*su*-arrangement of unit u_1 (i.e. *u*) shown below.

```
(?su (syn-sub (= ?eu ?au ?tu ?ou ?ru)) | (u (syn-sub (sl jo t ma bl))
     (utter    ?u1)                    |    (utter    ?u2)
     (form     ?f)                     |    (form     ((order u (jo sl bl t ma))))
     (syn-cat  (= SVOtoO-sentence)))   |    (syn-cat  (SVOtoO-sentence)))
```

Given a pattern structure $p = (== v_1 \dots v_m)$, a source structure $s = \{\ u_1 \dots u_n\ \}$ and a tuple of sets $(C_1, \dots, C_m)$ of the sort defined above, the set of FCG-unifiers of p and s can be defined as follows.

$$\{\sigma \mid \exists i_1 \dots \exists i_m (a_1^{i_1} \in C_1 \wedge \dots \wedge a_m^{i_m} \in C_m \wedge \sigma = mgu(p, (a_1^{i_1}, \dots, a_m^{i_m}))\}$$

That is, the set of FCG-unifiers of p and s is the set of most general unifiers of p and s', where s' is any p-arrangement of s of the form $(a_1^{i_1}, \dots, a_m^{i_m})$ such that s' and p are unifiable.

Clearly, constructing all the p-arrangements of a source structure s, and checking whether p and any of them are unifiable is not a practical approach to unification. Consider the unification example discussed above. Sets C_2 to C_6 contain a single pair of the form (arrangement, unifier), but set C_1 has 120

pairs. Therefore, in the worst case we would have to construct 120 structure arrangements and check whether each of them and the pattern are unifiable.

Instead, we use a *heuristic depth first search strategy* to explore the space of structure arrangements. The depth first search part of our approach consists in applying the unifier associated with a unit arrangement in a set C_i to the whole pattern and source structures. This requires undoing substitutions in backtracking steps, but it can be easily implemented in Prolog. The heuristic part is used to determine the order in which the sets C_i will be used to explore the set of structure arrangements. For example, if set C_i has fewer elements than set C_j, then we instantiate the i-esim component of an arrangement earlier than the j-esim one. Similarly, semantic relevance is used as a criterion for determining order of instantiation. For example a unit representing the predicate of a sentence should take precedence over other units representing its subject or its complements. Figure 8 shows part of a preliminary Prolog implementation of the FCG-unification algorithm outlined in this section.

Let us illustrate these ideas describing the application of the algorithm to our previous example of unification of the syntactic pattern of the construction in figure 2 and the syntactic source structure in figure 1. In accordance with the heuristics just mentioned the unit-arrangement and unifier in set C_2 would be used in first place, because unit u_2 describes a predicate and the set C_2 only has one element. Sets C_3 to C_6 also have one element. They might be explored in order of semantic relevance: C_4 subject, C_5 direct object, C_6 indirect object and C_3 preposition. Finally, the last set to be used would be C_1, because it has 120 elements. In fact, as we will see later, set C_1 will not even be constructed explicitly. All its arrangements but one will be pruned out by unification during the depth first search process. It is possible to estimate the number of elements of a set C_i without actually computing it[5]. Therefore, we need not assume that the sets C_i must be computed before starting the depth first search process.

First, unification is applied to units v_2 and u_2, the v_2-arrangement u_2^1 and unifier σ_2^1 in C_2 are used as follows: unit u_2 (i.e. *sl*) in the source is substituted for arrangement u_2^1, and unifier σ_2^1 = { ?eu = sl, ?u3 = ?u4, ?f2 = ((string sl slides)) } is applied to the construction and the source structure (see figure 9).

Next, sets $C_i = \{(u_i^1, \sigma_i^1)\}, i = 4, 5, 6, 3$ are used in that order: unit u_i in the source is substituted for arrangement u_i^1, and unifier σ_i^1 is applied to the pattern and the source structures. The result is shown in figure 10.

As we said before, once variables *?eu*, *?au*, *?ou*, *?tu* and *?ru* have been instantiated during the heuristic depth first search process (see figure 10), the set C_1 of arrangements of units in the source structure which can be made equal to unit v_1 consists of a single element rather than 120. That is, $C_1 = \{(u_1^1, \sigma_1^1)\}$,

[5] This is clear in the case of C_1, because the value of feature *syn-sub* in unit *?su* is a set consisting of five variables, therefore all the permutations of such a set unify with the value of the same feature in unit *u*. For the rest of the units one can simply check whether their single element in feature *syn-cat* belongs to the value of the same feature in each of the units in the source structure.

```
% fv_unif(P,S,A)
% P a pattern feature-value and S a source feature-value. If P and S
% are FCG-unifiable this predicate succeeds and A is instantiated to
% Ps, S' is a P-arrangement of S unifiable with P and s = mgu(P,S').

% unit_unif(P,S,A)
% P a pattern unit and S a source unit. If P and S are FCG-unifiable
% this predicate succeeds and A is instantiated to Ps, where S' is a
% P-arrangement of S unifiable with P and s = mgu(P,S').

% str_unif(P,S,A)
% P a pattern structure and S a source structre. If P and S are
% FCG-unifiable this predicate succeeds and A is instantiated to
% Ps, S' is a P-arrangement of S unifiable with P and s=mgu(P,S').

str_unif(P,S,A) :- sort_heur(P,S,SP), str_arrang(SP,S,A).

st_arr([],_,[]).
st_arr([H|T],S,[AH|R]) :- member(U,S), unit_unif(H,U,AH),
      delete(S,U,Rest), st_arr(T,Rest,R).

% sort_heur(P,S,SP)
% P is a pattern structure and S a source structure.
% This procedure instantiates SP to a list containing the units in
% P sorted in accordance with the heuristics used by the algortihm.
% Units in P which should be instatiated in first place are charac-
% terised as 'good' with respect to the source structure S, and are
% placed in the front of SP. A predicate better is used to indicate
% that a unit u1 should be instantiated earlier than another u2, i.e.
% that u1 should precede u2 in SP. The following rules describe some
% heuristics used by the FCG-unification algorithm.

good(U,S) :- unique_unifier(U,S).

better(U1,U2,S) :- more_unifiers(U2,U1,S), !.
better(U1,U2,S) :- \+ better(U2,U1,S), predicate_unit(U1),
   \+ predicate_unit(U2), !.
% similar rules for subject, direct object, indirect object...
```

Fig. 8. Partial description of the Prolog code of the FCG-unification algorithm (\+ denotes negation by failure).

(= (?su	(syn-sub	(= sl ?au ?tu ?ou ?ru))	{(u	(syn-sub	{sl jo bl t ma})
	(utter	?u1)		(utter	?u2)
	(form	?f1)		(form	{(order u (jo sl bl t ma))})
	(syn-cat	(= (SVOtoO-sentence)))		(syn-cat	{SVOtoO-sentence}))
(sl	(syn-sub	())	(sl	(syn-sub	{})
	(utter	?u4)		(utter	?u4)
	(form	((string sl slides)))		(form	((string sl slides)))
	(syn-cat	(= (role sl pred))))		(syn-cat	((role sl pred))))
(?tu	(syn-sub	())	(t	(syn-sub	{})
	(utter	?u5)		(utter	?u6)
	(form	(= (string ?tu to)))		(form	{(string t to)})
	(syn-cat	?s1)		(syn-cat	{(speech-part t prep)}))
(?au	(syn-sub	())	(jo	(syn-sub	{})
	(utter	?u7)		(utter	?u8)
	(form	?f3)		(form	{(string jo John)})
	(syn-cat	(= (role ?au subject))))		(syn-cat	{(stem jo john)
					(numb nat jo singular)
					(speech-part jo noun)
					(role jo subject)}))
(?ou	(syn-sub	())	(bl	(syn-sub	{})
	(utter	?u9)		(utter	?u10)
	(form	?f4)		(form	{(string bl blocks)})
	(syn-cat	(= (role ?ou dir-obj))))		(syn-cat	{(stem bl block)
					(numb nat bl plural)
					(speech-part bl noun)
					(role bl dir-obj)}))
(?ru	(syn-sub	())	(ma	(syn-sub	{})
	(utter	?u11)		(utter	?u12)
	(form	?f5)		(form	{(string ma Mary)})
	(syn-cat	(= (role ?ru ind-obj))))		(syn-cat	{(stem ma mary)
					(numb nat ma plural)
					(speech-part ma noun)
					(role ma ind-obj)})) }

Fig. 9. $C_2 = \{(u_2^1, \sigma_2^1)\}$ is used in first place: unit u_2 in the source is substituted for arrangement u_2^1, and unifier σ_2^1 is applied to pattern and source.

(= (?su	(syn-sub	(= sl jo t bl ma))	{(u	(syn-sub	{sl jo bl t ma})
	(utter	?u1)		(utter	?u2)
	(form	?f1)		(form	{(order u (jo sl bl t ma))})
	(syn-cat	(= (SVOtoO-sentence)))		(syn-cat	{SVOtoO-sentence}))
(sl	(syn-sub	())	(sl	(syn-sub	())
	(utter	?u4)		(utter	?u4)
	(form	((string sl slides)))		(form	((string sl slides)))
	(syn-cat	((role sl pred))))		(syn-cat	((role sl pred))))
(t	(syn-sub	())	(t	(syn-sub	())
	(utter	?u6)		(utter	?u6)
	(form	((string t to)))		(form	((string t to)))
	(syn-cat	((speech-part t prep))))		(syn-cat	((speech-part t prep))))
(jo	(syn-sub	())	(jo	(syn-sub	())
	(utter	?u8)		(utter	?u8)
	(form	((string jo John)))		(form	((string jo John)))
	(syn-cat	((role jo subject))))		(syn-cat	((role jo subject))))
(bl	(syn-sub	())	(bl	(syn-sub	())
	(utter	?u10)		(utter	?u10)
	(form	((string bl blocks)))		(form	((string bl blocks)))
	(syn-cat	((role bl dir-obj))))		(syn-cat	((role bl dir-obj))))
(ma	(syn-sub	())	(ma	(syn-sub	())
	(utter	?u12)		(utter	?u12)
	(form	((string ma Mary)))		(form	((string ma Mary)))
	(syn-cat	((role ma ind-obj)))))		(syn-cat	((role ma ind-obj))))}

Fig. 10. Result of using the arrangements and unifiers in sets C_4, C_5, C_6 and C_3 in that order to explore the space of structure arrangements.

where unifier σ_1^1 is {?su = u, ?u1 = ?u2, ?f1 = ((order u (jo sl bl to ma)))} and arrangement u_1^1 is as follows.

```
(u (syn-sub (sl jo t bl ma))
   (utter    ?u2)
   (form     ((order u (jo sl bl t ma))))
   (syn-cat  (SVOtoO-sentence)))
```

The result of using the arrangement and unifier in C_1, that is of substituting unit u_1 in the source structure for arrangement u_1^1, and applying unifier σ_1^1 to the source structure, is a complete arrangement s' of source structure s which satisfies the unifiability condition.

The heuristic depth first search process described above has allowed us to check that the syntactic pattern of the construction and the syntactic source structure are unifiable. The unifier built during this process can be stored for later use. But in order to implement construction application, we apply the substitutions of previous steps not only to the syntactic source structure and the syntactic pattern of the construction, but also to the semantic source structure and the semantic pattern of the construction (see figure 11).

```
(= (u (sem-sub  (= sl jo bl ma))
      (referent ?r1)
      (meaning  ?m1)
      (sem-cat  ?sc1))
 (sl  (sem-sub  ())
      (referent ?e)
      (meaning  ?m3)
      (sem-cat  (= (ev-type ?e tr) (ag ?e ?a)
                   (obj ?e ?o) (rec ?e ?r))))
 (jo  (sem-sub  ())
      (referent ?a)
      (meaning  ?m4)
      (sem-cat  ?sc7))
 (bl  (sem-sub  ())
      (referent ?o)
      (meaning  ?m5)
      (sem-cat  ?sc9))
 (ma  (sem-sub  ())
      (referent ?r)
      (meaning  ?m6)
      (sem-cat  ?sc11)))
```

```
{(u   (sem-sub  {sl jo bl ma})
      (referent ?2)
      (meaning  ?m2)
      (sem-cat  ?sc2))
 (sl  (sem-sub  {})
      (referent ?s)
      (meaning  {(act ?s slide) (arg1 ?s ?a)
                 (arg2 ?s ?o) (arg3 ?s ?r)})
      (sem-cat  {(ev-type ?s tr) (ag ?s ?a)
                 (obj ?s ?o) (rec ?s ?r))})
 (jo  (sem-sub  {})
      (referent ?j)
      (meaning  {(entity-type ?j person)
                 (count ?j one)})
      (sem-cat  ?sc8))
 (bl  (sem-sub  {})
      (referent ?b)
      (meaning  {(entity-type ?b block)
                 (count ?b several)})
      (sem-cat  ?sc10))
 (ma  (sem-sub  {})
      (referent ?m)
      (meaning  {(entity-type ?m person)
                 (count ?m one)})
      (sem-cat  ?sc12))}
```

Fig. 11. Result of applying the unifiers constructed during the heuristic depth first search process to the semantic pattern of the construction (left) and the semantic source structure (right).

As we explained before, construction application consists of two steps. First, the syntactic pattern of the construction and the syntactic source structure are unified. Then the instantiated semantic source structure is merged with the semantic pattern of the construction. In this example, the semantic source structure and the pattern are unifiable, therefore merge is equivalent to applying one of their unifiers to the semantic source structure.

The result of unifying the instantiated semantic source structure and the instantiated semantic pattern is substitution τ = { ?r1 = ?r2, ?m1 = ?m2, ?sc1 = ?sc2, ?e = ?s, ?m3 = ((act ?s slide) (arg1 ?s ?a) (arg2 ?s ?o) (arg3 ?s ?r)), ?a = ?j, ?m4 = ((entity-type ?j person) (count ?j one)), ?sc7 = ?sc8, ?o = ?b, ?m5 = ((entity-type ?b block) (count ?b several)), ?sc9 = ?sc10, ?r = ?m, ?m6 = ((entity-type ?m person) (count ?m one)), ?sc11 = ?sc12 }, which makes variables $?j$, $?b$ and $?m$, associated with the referents of units *jo*, *bl* and *ma* in the semantic source structure, equal to variables $?a$, $?o$ and $?r$, associated with the roles in the *transfer to recipient* event (*tr*) in that structure.

6 Conclusions

This paper has presented a logic programming approach to parsing and production in Fluid Construction Grammar (FCG). It builds on previous work on the formalisation of the *unification* and *merge* operations used in FCG in terms of First Order Logic (FOL) unification and search in the space of a particular set of FOL terms called *structure arrangements*.

Its main contribution is to outline a method for implementing unification and merge based on Logic Programming and Artificial Intelligence techniques such as unification and heuristic search. The formulation of the unification and merge problems in FCG as heuristic search problems in the space of structure arrangements not only allows understanding the problems of parsing and production with constructions in FCG as deduction problems, but also opens the door to the application of efficient automated deduction techniques to these problems.

Acknowledgements

Partially supported by BASMATI MICINN project (TIN2011-27479-C04-03) and by SGR2009-1428 (LARCA). I would like to thank Luc Steels and Joachim De Beule for valuable comments on an earlier version of this paper.

Bibliography

[1] Bleys, J., Stadler, K., De Beule, J.: Search in linguistic processing. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[2] Ciortiuz, L., Saveluc, V.: Fluid Construction Grammar and Feature Constraints Logics. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[3] Goldberg, A.: A Construction Grammar Approach to Argument Structure. Univ Chicago Press (1995)
[4] Harnad, S.: The symbol grounding problem. Physica D 42, 335–346 (1990)
[5] Langacker, R.: Foundations of Cognitive Grammar. Stanford Univ Press (1991)

[6] McCarthy, J.: Recursive functions of symbolic expressions and their computation by machine. Communications of the ACM 3(4), 184–195 (1960)
[7] McCarthy, J.: Formalizing Common Sense. Papers by John McCarthy. Ablex. Ed. V. Lifschitz (1990)
[8] Sierra-Santibáñez, J.: First order logic concepts in Fluid Construction Grammar. In: Biologically Inspired Cognitive Architectures 2011. pp. 344–350. IOS Press (2011)
[9] Steels, L.: The synthetic modeling of language origins. Evolution of Communication 1(1), 1–35 (1997)
[10] Steels, L.: Constructivist development of grounded construction grammars. In: Proceedings of the Annual Meeting of the Association for Computational Linguistics Conference. pp. 9–19 (2004)
[11] Steels, L.: Evolution of Communication and Language in Embodied Agents, chap. Modeling the Formation of Language: Embodied Experiments, pp. 235–262. Springer (2010)
[12] Steels, L., Beule, J.D.: Unify and merge in Fluid Construction Grammar. In: 3rd International Workshop on the Emergence and Evolution of Linguistic Communication. pp. 197–223. LNAI 4211, Springer (2006)
[13] Steels, L. (ed.): Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[14] Steels, L.: Design methods for Fluid Construction Grammar. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[15] Steels, L. (ed.): Experiments in Cultural Language Evolution. John Benjamins, Amsterdam (2012)
[16] Tomasello, M., Brooks, P.: The Development of Language, chap. Early syntactic development: A Construction Grammar approach, pp. 161–190. Psychology Press (1999)
[17] van Trijp, R.: A reflective architecture for language processing and learning. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[18] Wittgenstein, L.: Philosophical Investigations. Macmillan, New York (1953)