Chapter 10
Fluid Construction Grammar on Real Robots

Luc Steels[1,2], Joachim De Beule[3], and Pieter Wellens[3]

Abstract This chapter introduces very briefly the framework and tools for lexical and grammatical processing that have been used in the evolutionary language game experiments reported in this book. This framework is called Fluid Construction Grammar (FCG) because it rests on a constructional approach to language and emphasizes flexible grammar application. Construction grammar organizes the knowledge needed for parsing or producing utterances in terms of bi-directional mappings between meaning and form. In line with other contemporary linguistic formalisms, FCG uses feature structures and unification and includes several innovations which make the formalism more adapted to implement flexible and robust language processing systems on real robots. This chapter is an introduction to the formalism and how it is used in processing.

Key words: computational linguistics, construction grammar, Fluid Construction Grammar, parsing, production, grammar design

10.1 Introduction

In previous chapters of this volume, several layers of processing have already been discussed, from embodiment, low level motor control and signal processing, and feature extraction and pattern recognition, to the construction of world models and the planning and interpretation of the meaning of utterances. We have now arrived at the top-most layer which uses all this information to maintain dialogs in the form of evolutionary language games. This chapter introduces the fundamental framework and implementation tools that we have developed for this purpose. This framework is based on the notion of a construction.

[1]Sony Computer Science Laboratory Paris, France, e-mail: steels@arti.vub.ac.be
[2]ICREA Institute for Evolutionary Biology (UPF-CSIC), Barcelona, Spain
[3]Artificial Intelligence Laboratory Vrije Universiteit Brussel, Belgium

The notion of a construction has been at the core of linguistic theorizing for centuries (Östman and Fried, 2004). A *construction* is a regular pattern of usage in a language, such as a word, a combination of words, an idiom, or a syntactic pattern, which has a conventionalized meaning and function (Goldberg and Suttle, 2010). The meaning and functional side of a construction, as well as relevant pragmatic aspects, are captured in a *semantic pole*. All aspects which relate to form, including syntax, morphology, phonology and phonetics are captured in a *syntactic pole*. Constructions clearly form a continuum between quite abstract grammatical constructions, such as the determiner-nominal construction, and so called *item-based constructions*, which are built out of lexical materials and frozen syntactic patterns. They contain open slots in which structures with specific semantic and syntactic properties can fit, as in the "let-alone" construction, underlying a sentence like "Joan is unable to write 5 pages, let alone a whole book" (Fillmore et al, 1988).

From a linguistics point of view, a constructional approach to language implies developing a catalog of all constructions in a language. This goal differs profoundly from a generative grammar approach which attempts to generate all possible syntactic structures of a language. In construction grammar, meaning and form always go hand in hand and the key issue is to understand the bi-directional mapping between the two.

The constructional approach has abundantly proven its worth in descriptive linguistics (Goldberg, 1995) and is also used almost universally in second language teaching. Moreover empirical evidence from child language acquisition shows that language learning can be understood by the progressive usage-based acquisition of constructions (Lieven and Tomasello, 2008). The constructional perspective has also been very productive for historical linguists, and there is now a large body of clear examples showing how new constructions may develop from the creative extension of existing constructions by a few individuals to a productive common pattern that is adopted by the linguistic community as a whole (Fried, 2009).

But construction-based approaches to language are also relevant for language processing, and particularly for language processing on robots, because they integrate aspects of meaning directly in considerations of grammar. This has many advantages. Most of the utterances produced by real speakers are incomplete and ungrammatical in the strict sense. Hearers focus on meaning. They try to understand an utterance, even if syntactic structures are not conform the established norms or if it is still incomplete. A construction-based grammar makes this much easier because aspects of meaning are directly available for semantic analysis as soon as certain words or partial fragments have been recognized.

Second, language is non-modular in the sense that many linguistic decisions on how to say or interpret something involve many different levels: pragmatics, semantics, syntax, morphology, and phonology. For example, the suffix added to a Hungarian verb expresses features such as number and gender which are both derived from the subject *and* the direct object (so called poly-personal agreement, see Beuls, 2011). It follows that constructions should have access to whatever layer of analysis they need in order to define all the constraints relevant for a particular step in linguistic decision-making. Because constructions can cross all these levels they are much more efficient representations of grammar than if these constraints are teased apart in separate autonomous layers.

There are many ways to implement construction grammars, depending on what representational and computational mechanisms are adopted as underlying foundation. Here we focus on a new formalisation called Fluid Construction Grammar (FCG for short) which has been developed in the project described in this book. As the name suggests, FCG was specifically designed to deal with the robustness and flexibility that natural language processing on real robots requires.

FCG uses techniques now common in formal and computational linguistics, such as the representation of linguistic structures with feature structures (Copestake, 2002), and the use of unification for applying constructions to expand linguistic structures in language parsing and production, as pioneered in Functional Unification Grammar (Kay, 1986), and also used in Lexical Functional Grammar (Dalrymple et al, 1995), and Head-driven Phrase structure Grammar (Pollard and Sag, 1994). Like many other computational linguistics efforts, the FCG-system is embedded within a contemporary Common LISP-based programming environment from which it inherits well-tested mechanisms for representing and processing complex symbolic structures. Other proposals for operationalizing construction grammar, such as Embodied Construction Grammar (Bergen and Chang, 2005) and Sign-Based Construction Grammar (Michaelis, 2009) draw on mechanisms arising from the same computational tradition but use them in different ways. Given the current state of the field, it is highly beneficial that many approaches are explored in order to discover the best way to formalize and implement construction grammars.

Fluid Construction Grammar has been fully implemented in a system called the FCG-system, which is made available for free to the research community (http://www.fcg-net.org/). The FCG-system contains a core component (called the FCG-interpreter) that performs basic operations needed for parsing and production, as well as various tools to aid linguistic research, such as a tool for browsing through linguistic structures and a tool for monitoring the success rate of a grammar when processing a set of test cases. The FCG-system has been under development from around 1998 in order to support experiments in modeling language evolution using language games played by autonomous robots (Steels, 1998). Since then, it has undergone major revisions and enhancements and the system is still continuously being adapted and revised to cope with new linguistic phenomena and new processing challenges. Nevertheless, the system is already sufficiently stable that it can be used to tackle sophisticated issues in the representation and processing of language. A full overview and many detailed examples for different languages are given in Steels (2011, 2012).

FCG is in principle neutral with respect to which representation is used for semantics, but in the case of applications with robots, the meaning always consists of (partial) IRL networks, as explained in earlier contributions to this volume (Spranger et al, 2012). This meaning is either directly associated with words or it is first mapped to semantic categorizations, then to syntactic categorizations, and finally to a surface form through a variety of constructions.

There are two different levels in Fluid Construction Grammar. The first lowest level is the processing level. It concerns the fundamental primitive datastructures and operations that are available for writing construction grammars and the machinery for computing syntactic and semantic structures during parsing and production. The second level is the design level. It concerns methods and techniques that have been developed for coping with the complexity of writing real grammars and computational abstractions in the form of templates that capture these methods. The remainder of this chapter discusses very briefly each of these levels. It is not possible within the available space limitations to do more than give readers a suggestive glimpse of the formalism, but many additional sources are available to learn more (Steels, 2011, 2012).

10.2 The processing level

FCG uses *transient structures* to represent all the information about the sentence being parsed or produced. Transient structures consist of a set of units, roughly corresponding to morphemes, words, or phrases, and information attached to each of these units in the form of features and values. The example in Figure 10.1 shows the outline of the transient structure for the German phrase “der Block links von mir” (the block left of me) as it is displayed in the FCG-interface. Spranger and Loetzsch (2011) describes in detail the grammar used in this example. At first sight this structure looks like the kind of trees found in all parsing and production systems, except that it is visualized from left to right for the semantic and from right to left for the syntactic structure. The names of the units (such as `left-unit-14` or `von-unit-21`) are purely conventional. The names have been chosen to make it easier to follow what is going on. The indices are added because there may be several units with the same role, for example more than one occurrence of the word “von”. These indices are automatically computed when new units are created during processing.

When we click on one of these boxes in the FCG user interface, the features associated with each unit reveal themselves. These features may concern any level of language: pragmatic and semantic information is grouped in the semantic pole of the unit and syntactic, morphological, and phonological features in the syntactic pole. For example, if we click on `left-unit-14` we see the semantic (left) and syntactic pole (right) associated with this unit as shown in Figure 10.2. Which features are adopted for a particular grammar is entirely open to the grammar designer. We see here on the semantic pole information about the meaning of the word “links” and its semantic categories (namely that it is an angular lateral spatial category). The syntactic pole contains a form feature with information on what is the stem for this unit (namely “link”), the string it covers (namely “links”) and a syn-cat (syntactic category) feature containing information relevant for morphology (case/gender) and the lexical category or part of speech (lex-cat). The FCG processing level is en-

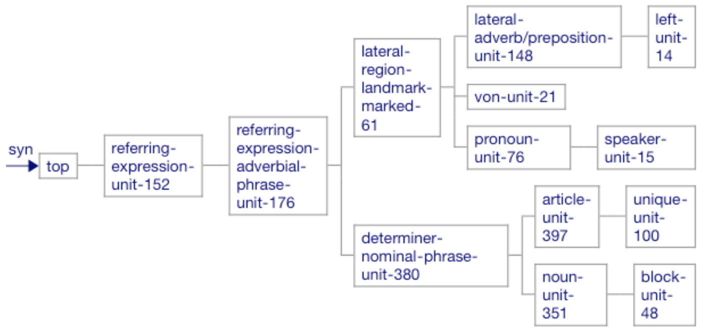
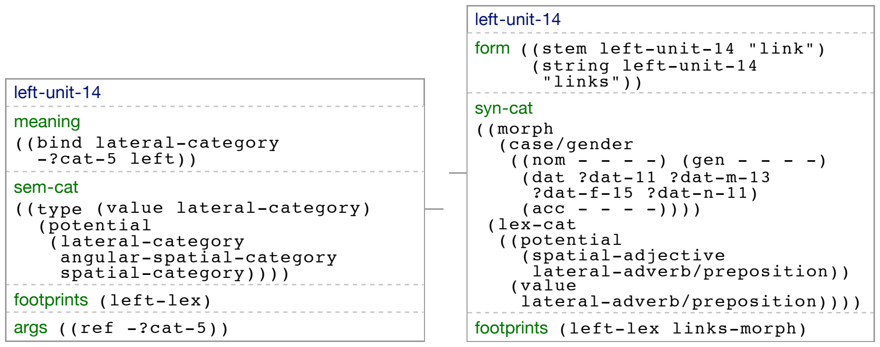

syn
top
referring-expression-unit-152
referring-expression-adverbial-phrase-unit-176
lateral-region-landmark-marked-61
lateral-adverb/preposition-unit-148
left-unit-14
von-unit-21
pronoun-unit-76
speaker-unit-15
determiner-nominal-phrase-unit-380
article-unit-397
unique-unit-100
noun-unit-351
block-unit-48

Fig. 10.1 Syntactic pole of a transient structure created during the production or parsing of the German phrase “der Block links von mir”.

tirely agnostic about which features and values are used in the grammar. A grammar designer can introduce new ones at any time by just using them.



```
left-unit-14
meaning
((bind lateral-category
  -?cat-5 left))
sem-cat
((type (value lateral-category)
  (potential
   (lateral-category
    angular-spatial-category
    spatial-category))))
footprints (left-lex)
args ((ref -?cat-5))
```

```
left-unit-14
form ((stem left-unit-14 "link")
      (string left-unit-14
       "links"))
syn-cat
((morph
  (case/gender
   ((nom - - - -) (gen - - - -)
    (dat ?dat-11 ?dat-m-13
     ?dat-f-15 ?dat-n-11)
    (acc - - - -))))
 (lex-cat
  ((potential
    (spatial-adjective
     lateral-adverb/preposition))
   (value
    lateral-adverb/preposition))))
footprints (left-lex links-morph)
```

Fig. 10.2 Details of left-unit-14 in the transient structure covering the word “links”. On the left the semantic pole of a unit and on the right the syntactic pole of this unit.

Transient structures are also used to represent the hierarchy of units and subunits, as shown in Figure 10.3, which displays the semantic pole of `speaker-unit-15` (covering the word "mir") and `pronoun-unit-76`, which is its parent-unit (hierarchy is shown from right to left because this is the semantic pole). The parent-unit was created by the application of a construction (see later). Notice the feature sem-subunits in `pronoun-unit-76`, which is filled with a list of subunits, in this case only `speaker-unit-15`. The explicit representation of subunits makes it possible to represent trees which are not strictly hierarchical, and the separation between semantic and syntactic subunits makes it possible to have differences in semantic or syntactic structure.

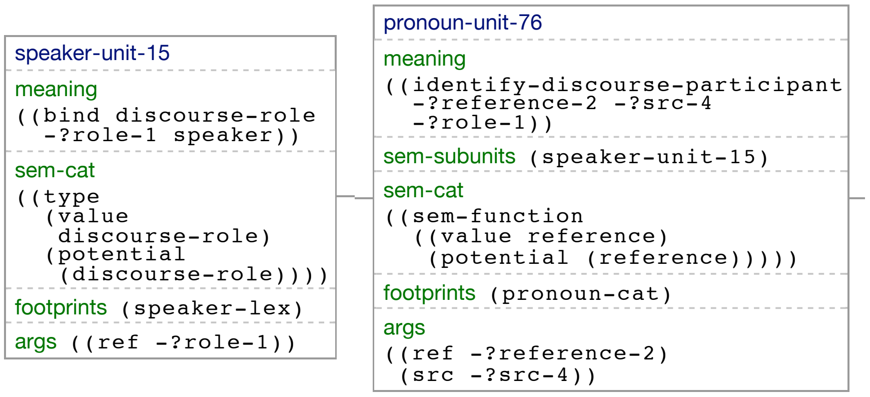

speaker-unit-15
meaning
((bind discourse-role -?role-1 speaker))
sem-cat
((type (value discourse-role) (potential (discourse-role))))
footprints (speaker-lex)
args ((ref -?role-1))
pronoun-unit-76
meaning
((identify-discourse-participant -?reference-2 -?src-4 -?role-1))
sem-subunits (speaker-unit-15)
sem-cat
((sem-function ((value reference) (potential (reference)))))
footprints (pronoun-cat)
args
((ref -?reference-2) (src -?src-4))

Fig. 10.3 Small section of a hierarchical semantic structure. The pronoun-unit is the parent node of the speaker-unit.

Constructions have the same structure as transient structures. They also consist of units, features and values for these features and the information is again organized into two poles. Constructions in FCG are considered to be bi-directional associations, in the sense that they establish bi-directional mappings between meaning and form through the intermediary of syntactic and semantic categorizations. They are usable both as part of a production process that translates meaning into form or a parsing process that translates form into meaning. This dual usage happens without changing or recompiling the representation of a construction and without giving up the efficiency needed both for language production and parsing.

Constructions (and transient structures) have not only a graphical representation (as shown in Figure 10.4) but also a list representation which is particularly useful if constructions become complicated with many units and syntactic and semantic specifications attached to them. The list representation of the construction shown in Figure 10.4 is as follows:

Example 1.

```
(def-cxn mouse-cxn ()
  ((?top-unit
    (tag ?meaning
         (meaning
          (== (bind object-class ?class mouse)))))
   ((J ?mouse-unit ?top-unit)
     ?meaning
     (args (?class))
     (sem-cat
      (==1 (is-animate +) (class object)
           (is-countable +)))))
  <==>
  ((?top-unit
     (tag ?form}
          (form (== (string ?mouse-unit "mouse")))))
   ((J ?mouse-unit ?top-unit)
     ?form
     (syn-cat
       (==1 (lex-cat noun)
            (number singular))))))
```

This example is a lexical construction named `mouse-cxn` that defines the meaning contributed by the word “mouse”. The details are probably overwhelming at this point but not important for getting a first impression. This is a representation at the lowest level, like machine code, and a higher level representation using templates will be introduced in the next section. On the semantic side the construction introduces an IRL operation that binds the semantic entity `mouse-class` to a variable `?class`. There are also semantic categorizations, namely that this class refers to an animate countable object. On the syntactic side, the construction introduces the word and some syntactic categorizations, namely that “mouse” is a singular noun.

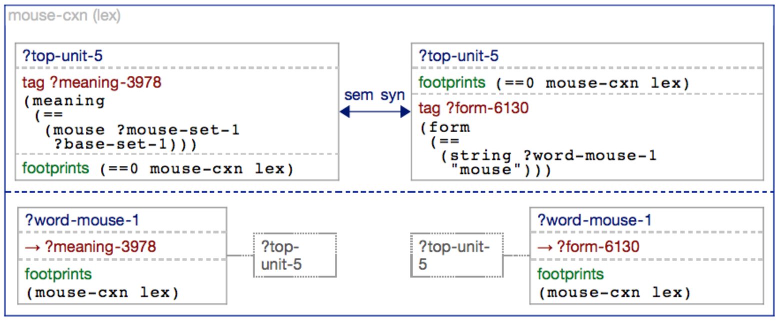


Fig. 10.4 Example of a lexical construction for the word “mouse”. The top part describes the semantic (left) and syntactic (right) constraints on the application of the construction. The bottom part describes what is contributed by the constraint to the transient structure.

Notice that there are two parts to constructions, as shown in Figure 10.5. The top part is the conditional part and defines what has to be there in the transient structure before the construction can apply. There is both a semantic and a syntactic conditional part. The semantic conditions are checked first in production and the syntactic conditions in parsing. The bottom part is the contributing part of a construction. It defines what will be added to the transient structure. Again there is both a semantic and a syntactic contributing part. The semantic part is added in parsing and the syntactic part in production. So FCG constructions are not like generative grammar rules that rewrite a non-terminal symbol, they always associate semantic with syntactic structure. Moreover each construction not only defines a syntactic structure but also how that syntactic structure is to be interpreted semantically, which makes it possible to have a tighter integration of syntax and semantics, as compared to proposals where syntax and semantics are kept separate (as in Montague grammar for example).

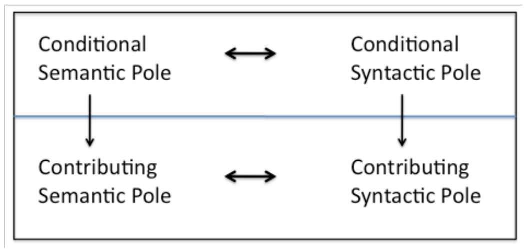


Fig. 10.5 General structure of a construction. There is a conditional part and a contributing part. In parsing, the conditional part of the syntactic pole is matched first and the rest is merged into the transient structure if successful. In production, the conditional part of the semantic pole is matched first and the rest is merged into the transient structure if successful.

Constructions are applied in a process of matching and merging, described more formally in Steels and De Beule (2006):

- *Matching* means that the conditional part C of one pole of the construction (the semantic pole in production or the syntactic pole in parsing) is compared with the corresponding pole T in the transient structure to see whether correspondents can be found for every unit, feature, and value. C and T may both contain variables. Variables are denoted by names preceded by a question mark. As part of the matching process, these variables get bound with specific values (or other variables) in the target using a unification operation familiar from logic programming or other feature-based formalisms. For example, if the construction contains the value (stem ?left-unit "link") and the transient structure contains the value (stem left-unit-14 "link") for the same feature, then these two values match if `?left-unit` is assumed to be bound to `left-unit-14`.
- *Merging* means that the conditional part of the other pole of the construction (the syntactic pole in production or the semantic pole in parsing) is combined with the corresponding pole in the transient structure, in the sense that everything missing in the target pole of a transient structure is added unless they are in conflict. In addition, the contributing parts of both poles from the construction are added to the corresponding poles of the transient structure.

An example of the application of a construction is shown in Figure 10.6. The construction itself, taken from Spranger and Loetzsch (2011), is defined in the appendix, but the details are not important. The construction takes a nominal phrase with the potential and turns it into a referring expression. On the syntactic side it settles case and gender and on the semantic side it adds extra meaning. The detailed semantic and syntactic poles after construction application are shown in Figures 10.7 and 10.8.

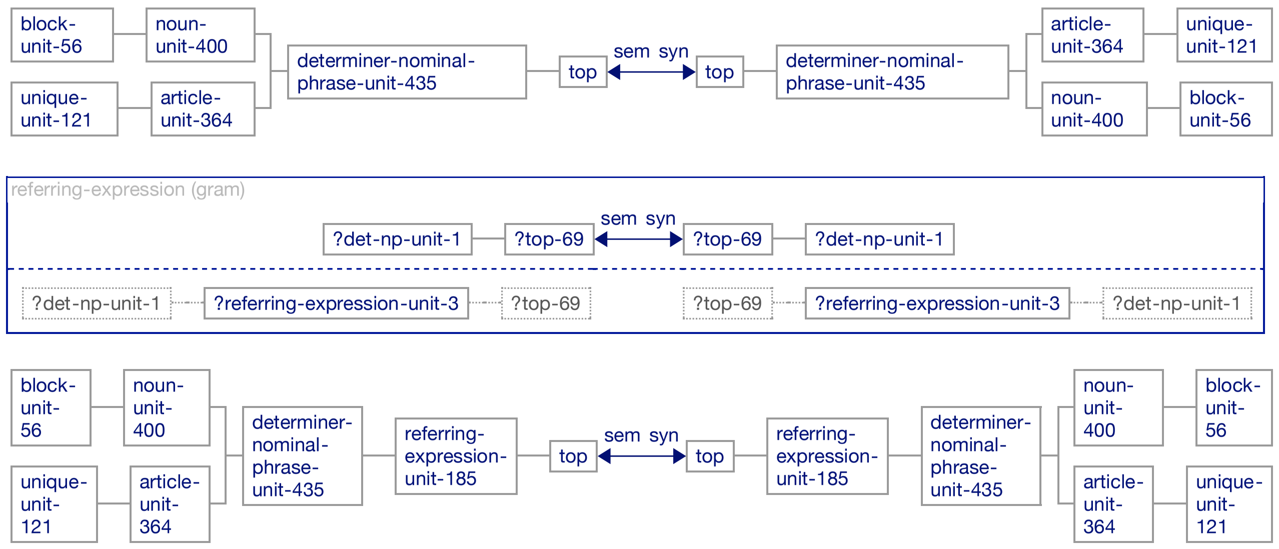


Fig. 10.6 Example of construction application. The top shows the transient structure before application and the bottom shows the expanded transient structure after the operation of matching and merging. The transient structure has an additional unit for the referring expression. The middle shows the schematic of the construction defined in detail in the appendix.

An important property of FCG is that exactly the same construction applies both in parsing and in production. This means that if during parsing a unit is found with the stem “mouse”, this triggers the same construction and builds the same structures as it would in production. This mirror property has a large number of advantages. For example, during parsing it is possible to start applying constructions in a top-down manner to predict properties of words or phrases that have not been pronounced yet or to fill in information about fragments of the utterance that have not been understood properly or are unknown. During production it is possible to engage in self-monitoring, because partial structures constructed by application of constructions from meaning to form can be re-entered in a parsing process, in order to see for example whether any combinatorial search is occurring or whether the utterance to be pronounced indeed expresses the intended meaning as accurately as possible. Although top-down prediction at the syntactic level is also possible with generative grammars, FCG has direct access to the semantic level and can therefore make much more precise predictions based on partial understanding.

Language users often have stored several alternative constructions in their memory because there is unavoidably a lot of variation in the language they encounter and because of constructional synonymy, syncretism, homonymy and ambiguity. Constructions therefore have an associated score which reflects their past success in utilisation. Usually more than one construction can apply to a given transient structure and consequently the exploration of a search space is unavoidable. Part of such a search space for the example of “un gros ballon rouge” is shown in Figure 10.9 (taken from Bleys et al, 2011). The different nodes in the search space are represented in a tree. At the moment of parsing “ballon”, it is still unclear whether the phrase will end with “ballon” (as in “un gros ballon”), or whether it will continue, as it does in this case. FCG supports standard heuristic best-first search.

A score is computed for every possible expansion and then the transient structure with the highest score is pursued further. This score might for example take into account what constructions had most success in the past and are therefore probably more entrenched in the population. Another component that determines the score, particularly of final nodes in the search space, are goal tests. They examine a transient structure to see whether it satisfies desired criteria. The FCG-interpreter performs backtracking to earlier nodes in the search space if a particular transient structure reaches a dead-end or if an unsatisfactory end state was reached.

Human language users must have hundreds of thousands of constructions in memory and they are applying them at incredible speeds. Increasing efficiency and damping combinatorial explosions in search is therefore one of the key challenges in building operational parsing and production systems. This challenge can partly be tackled by writing grammars in such a way that they minimize search (see section on design below). In addition, FCG has various mechanisms to help grammar designers control at a more fine-grained level the selection and application of constructions:

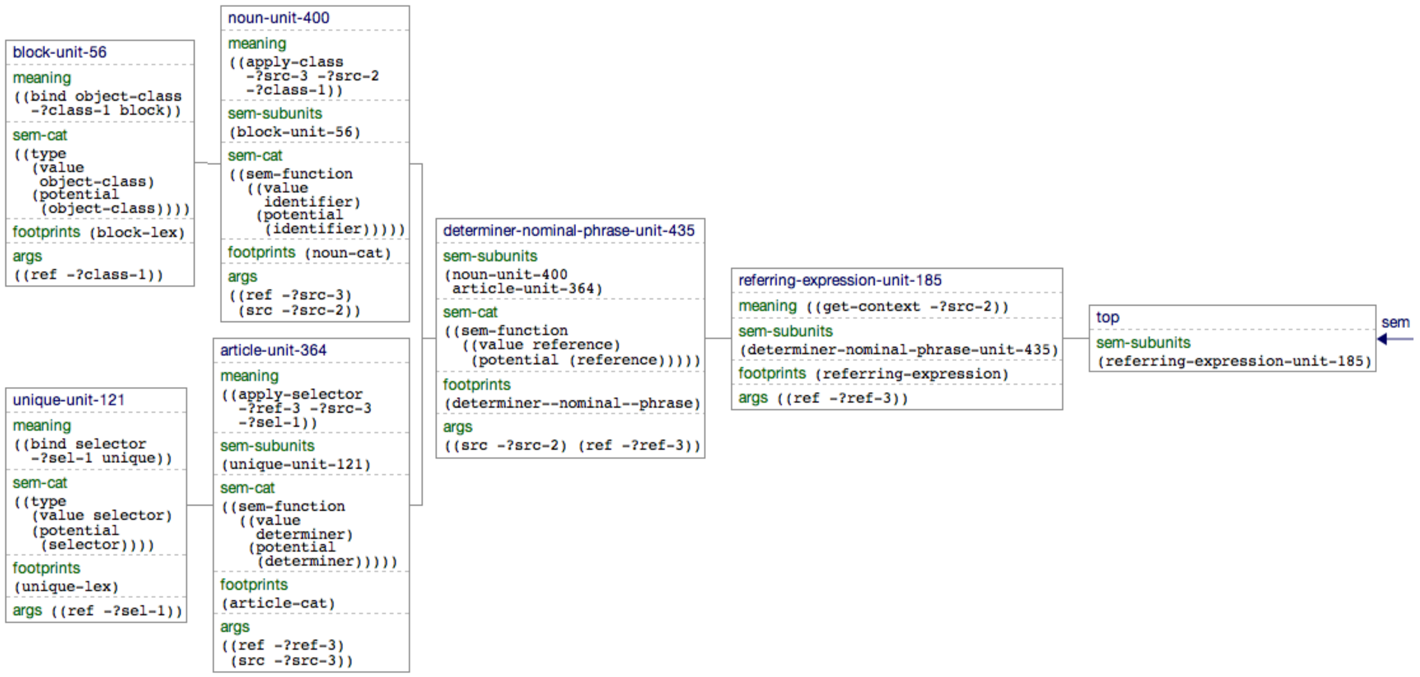
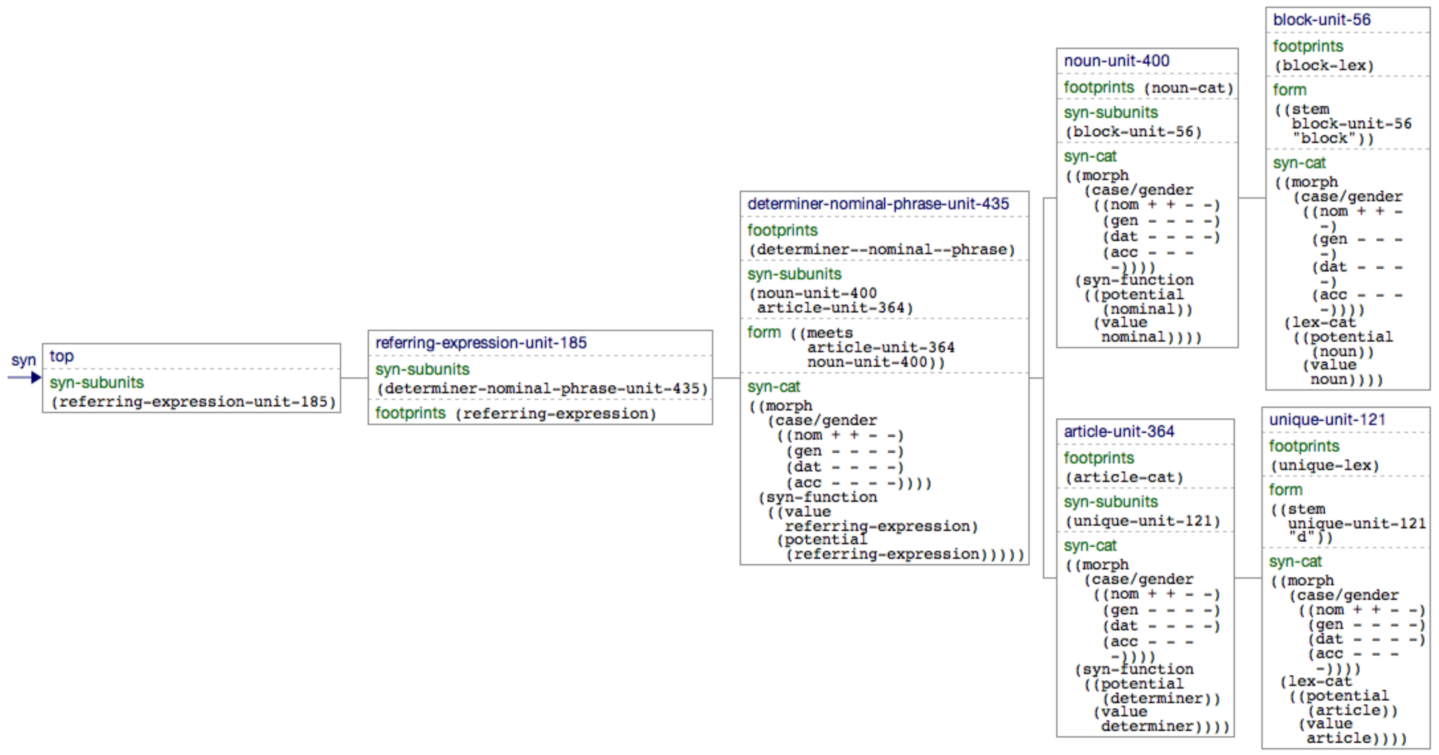

block-unit-56
meaning
((bind object-class -?class-1 block))
sem-cat
((type (value object-class) (potential (object-class))))
footprints (block-lex)
args
((ref -?class-1))
noun-unit-400
meaning
((apply-class -?src-3 -?src-2 -?class-1))
sem-subunits
(block-unit-56)
sem-cat
((sem-function ((value identifier) (potential (identifier)))))
footprints (noun-cat)
args
((ref -?src-3) (src -?src-2))
unique-unit-121
meaning
((bind selector -?sel-1 unique))
sem-cat
((type (value selector) (potential (selector))))
footprints
(unique-lex)
args ((ref -?sel-1))
article-unit-364
meaning
((apply-selector -?ref-3 -?src-3 -?sel-1))
sem-subunits
(unique-unit-121)
sem-cat
((sem-function ((value determiner) (potential (determiner)))))
footprints
(article-cat)
args
((ref -?ref-3) (src -?src-3))
determiner-nominal-phrase-unit-435
sem-subunits
(noun-unit-400 article-unit-364)
sem-cat
((sem-function ((value reference) (potential (reference)))))
footprints
(determiner--nominal--phrase)
args
((src -?src-2) (ref -?ref-3))
referring-expression-unit-185
meaning ((get-context -?src-2))
sem-subunits
(determiner-nominal-phrase-unit-435)
footprints (referring-expression)
args ((ref -?ref-3))
top
sem-subunits
(referring-expression-unit-185)
sem

Fig. 10.7 Semantic pole of the transient structure after applying the construction shown in Figure 10.6.



```
syn
top
syn-subunits
(referring-expression-unit-185)
```

```
referring-expression-unit-185
syn-subunits
(determiner-nominal-phrase-unit-435)
footprints (referring-expression)
```

```
determiner-nominal-phrase-unit-435
footprints
(determiner--nominal--phrase)
syn-subunits
(noun-unit-400
 article-unit-364)
form ((meets
       article-unit-364
       noun-unit-400))
syn-cat
((morph
  (case/gender
   ((nom + + - -)
    (gen - - - -)
    (dat - - - -)
    (acc - - - -))))
 (syn-function
  ((value
    referring-expression)
   (potential
    (referring-expression)))))
```

```
noun-unit-400
footprints (noun-cat)
syn-subunits
(block-unit-56)
syn-cat
((morph
  (case/gender
   ((nom + + - -)
    (gen - - - -)
    (dat - - - -)
    (acc - - -
     -))))
 (syn-function
  ((potential
    (nominal))
   (value
    nominal))))
```

```
block-unit-56
footprints
(block-lex)
form
((stem
  block-unit-56
  "block"))
syn-cat
((morph
  (case/gender
   ((nom + + -
     -)
    (gen - - -
     -)
    (dat - - -
     -)
    (acc - - -
     -))))
 (lex-cat
  ((potential
    (noun))
   (value
    noun))))
```

```
article-unit-364
footprints
(article-cat)
syn-subunits
(unique-unit-121)
syn-cat
((morph
  (case/gender
   ((nom + + - -)
    (gen - - - -)
    (dat - - - -)
    (acc - - -
     -))))
 (syn-function
  ((potential
    (determiner))
   (value
    determiner))))
```

```
unique-unit-121
footprints
(unique-lex)
form
((stem
  unique-unit-121
  "d"))
syn-cat
((morph
  (case/gender
   ((nom + + - -)
    (gen - - - -)
    (dat - - - -)
    (acc - - -
     -))))
 (lex-cat
  ((potential
    (article))
   (value
    article))))
```

Fig. 10.8 Syntactic pole of the transient structure after applying the construction shown in Figure 10.6.

1. Constructions can add footprints to transient structures, in a sense tagging the points where they have made a change, so that later on when the construction is tried again on the same transient structure the construction does not re-apply endlessly. Footprints can also be used to regulate the activation of families of constructions or to handle defaults (Beuls, 2011). By convention the name of the footprint is equal to the name of the construction. The use of footprints can be seen for example in Figure 10.4. We see that `?top-unit-5` tests in the conditional part, both on the semantic and the syntactic side, whether the *footprints* feature does *not* already include `mouse-cxn` and `lex`. The contributing part will add these footprints to avoid circular application of this construction.

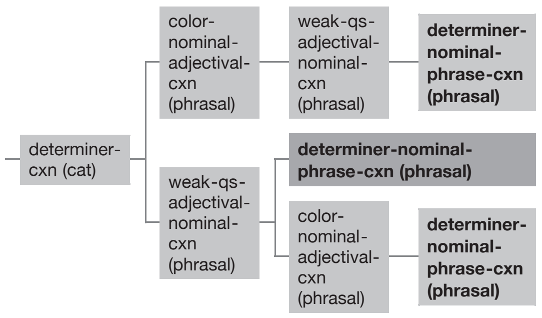

determiner-cxn (cat)
color-nominal-adjectival-cxn (phrasal)
weak-qs-adjectival-nominal-cxn (phrasal)
determiner-nominal-phrase-cxn (phrasal)
determiner-nominal-phrase-cxn (phrasal)
weak-qs-adjectival-nominal-cxn (phrasal)
color-nominal-adjectival-cxn (phrasal)
determiner-nominal-phrase-cxn (phrasal)

Fig. 10.9 Search space computing all possible parses of the utterance “un gros ballon rouge”. The process branches after the application of the functional constructions. The two successful branches (the top and bottom one) lead to similar meanings, the failed branch (with darker background) leads to an incomplete parse.

2. Constructions are organized in sets and networks. Sets are useful to ensure that a certain set of constructions, for example all morphological constructions, have operated before another set, for example the phrase structure constructions. Networks give a more fine-grained way to prioritize the execution of constructions. For example, one network could be based on the generality/specificity relations between constructions, used for example to represent families of constructions. Another network is based on conditional dependencies (see Figure 10.10 from Wellens, 2011): One construction C-1 (for example a determiner-nominal construction) conditionally depends on another construction C-2 (for example a nominal construction) if the triggering of C-2 creates some of the conditions under which C-1 could potentially trigger. Conditional dependencies can be used for priming. If a nominal construction was able to operate, it will make it more likely that the determiner-nominal construction is applicable, and conversely if no nominal construction was able to apply it does not make sense to try a determiner-nominal construction either.
3. Networks of constructions that have proven to be useful can be chunked. Chunking means that the information required or supplied by individual constructions is combined into a single construction which can thus be matched and merged

more efficiently. Intermediary steps that are no longer necessary can be removed (Stadler, 2012).

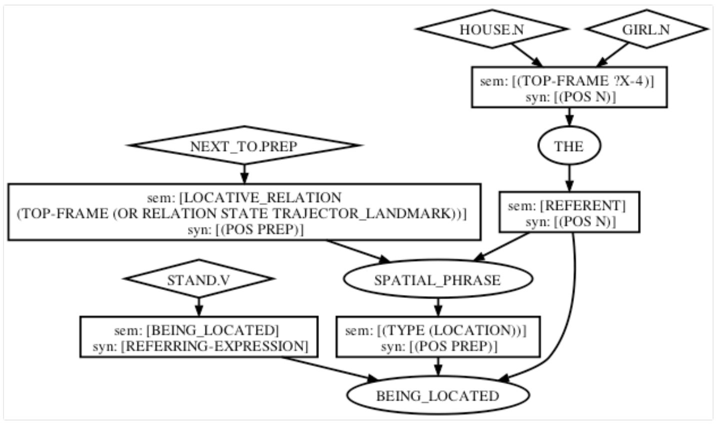


Fig. 10.10 Example of dependency relations between constructions. Such networks can be used to support priming which aids considerably to reduce search and speech up construction access.

10.3 The design level

Writing operational constructions is a very difficult exercise for two reasons. First of all, many factors normally intervene in a single construction, indeed it is one of the main tenets of construction grammar that linguistic knowledge should be packaged in such a way that as many constraints as possible get incorporated in each construction. So a construction could include a phonological constraint (for example vowel harmony to select a morphological affix) or a pragmatic constraint (for example which constituent is being emphasized in the sentence). This makes processing much more efficient compared to horizontally structured grammars where the syntactic level is viewed as autonomous and cannot incorporate issues of meaning or phonology. Second, there are many interactions between constructions that may not be immediately obvious to the grammar designer: Constructions can be in competition with each other because they cover the same meaning or the same form, and the implications of the best choice could only become visible much later.

In the design or investigation of highly complex systems it is often useful to introduce higher level abstractions that then translate into detailed structures and processes. For example, computer programs are usually written in a high level programming language (like LISP or Python) and code written at this level is then translated automatically by compilers or interpreters into a huge number of detailed operations that can be executed at the machine level. The grammar of a human language is certainly a highly complex system and it therefore makes sense to use the same approach.

Through various case studies in Fluid Construction Grammar, a set of design patterns is gradually being discovered, some of which have in fact already a long tradition in linguistics. The notion of a design pattern comes from architecture and is also widely used in computer science. An architectural design pattern is for instance the use of a dome structure (such as the Santa Maria del Fiore Duomo in Florence built by Bruneschelli). There are general principles of dome design but specific details depend on the required size and height of the space that needs to be covered as well as on esthetic considerations. In the context of grammar, a design pattern circumscribes the core solution to a particular aspect of grammar, not just in a descriptive way but also in terms of processing and learning operations. The specific details how the design pattern is instantiated in a particular language still need to be worked out and the details will be significantly different from one language to another one. Some languages may even use certain design patterns which are entirely absent from others.

Here are two examples of design patterns:

1. Many languages feature complex morphological and agreement systems which group a set of features (such as number, case and gender). But there is almost never a simple mapping. Often the same word or morpheme may express different competing feature bundles (for example, the German article "die" expresses the nominative and accusative feminine singular, as well as all plural nominative and accusative cases.) A design pattern to efficiently handle the processing of these ambiguities, is a feature matrix, reminiscent of the distinctive feature matrices in phonology (van Trijp, 2011). It contains rows and columns for the different dimensions and either + or - if there is a known value or a variable if the value is not yet known. Putting the same variable in different slots of the matrix can be used to capture constraints between values. Given such feature matrices, agreement and percolation phenomena can be handled by the standard matching and merging operations of unification-based grammars. Ambiguity does not need to translate into exploding branches in the search tree but translates into open variables which get bound whenever the information becomes available.
2. All human languages tend to reuse the same word forms for different purposes. For example, the English word "slow" may be used as an adjective (The slow train), a verb (They slow down), a predicate (The train was slow) or a noun (The slow go first). It would be highly costly to create new nodes in the search space each time the word "slow" is encountered. An alternative is to use a design pattern based on a distinction between actual and potential. A particular word has the potential to belong to several word classes but then in the course of processing it becomes clear which of these is the actual value. By explicitly representing the potential, it is possible for constructions to eliminate some possibilities or add others (for example through coercion) before a final decision is made, as illustrated clearly in Spranger and Loetzsch (2011).

The FCG-system comes with a collection of templates for supporting such design patterns and with facilities for introducing new templates if the grammar designer wishes to do so. These templates provide ways to leave out many details, particularly details related to the operationalization of constructions. They help to focus on the linguistic aspects of constructions and bridge the gap between the detailed operational level and the level at which linguists usually work. For example, there is a template for defining the skeleton of lexical constructions which has slots to specify the meaning and the word stem. This template then creates the necessary units, features, and values, including the structure building operations that are required.

Concretely, the earlier example of the `mouse-cxn` (Figure 10.4 and Example 1) would actually be defined in a more abstract way using the `def-lex-skeleton` and `def-lex-cat` templates. The skeleton defines the basic structure of the lexican construction: its meaning, arguments and string. The cat-template introduces semantic and syntactic categorizations associated with the word. All the intricacies of J-units, footprints, etc. are all hidden here from the grammar designer.

Example 2.

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

There are also templates for defining the components of phrasal constructions, including templates for specifying what agreement relations hold between the constituents, how information from constituents percolates to the parent unit, how the meanings of the different constituents gets linked, what meanings are added by the construction to the meanings supplied by the different units, and what additional form constraints are imposed by the construction. Other templates are available for defining more complex feature matrices and the grammatical paradigms on which they are based, and for using these feature matrices to establish agreement and percolation, for defining the topology of fields and the constraints under which a constituent can be 'grabbed' by a field, and so on. The inventory of possible templates now used in FCG is certainly not claimed to be a universal or complete set, on the contrary, we must expect that this inventory keeps being expanded and elaborated as new design patterns are uncovered.

10.4 Conclusions

Fluid Construction Grammar shows that construction grammar need not stay at a verbal level only. It is entirely possible to formalize notions of construction grammar and use this formalization for achieving the parsing and production of sentences. A constructional approach has a number of advantages for human-robot interaction compared to those used traditionally in computational linguistics, in particular from the viewpoint of efficiency (because constructions can stretch all levels of linguistic analysis), from the viewpoint of robustness (because constructions can be flexibly applied), and from the viewpoint of modeling language as an open, adaptive system, which it certainly is.

Acknowledgements

The FCG system was developed at the Artificial Intelligence Laboratory of the Vrije Universiteit Brussel and the Sony Computer Science Laboratory in Paris. The earliest ideas and implementations already date from the late nineties when the first implementations were made by Luc Steels and Nicolas Neubauer. Since then the system has undergone many revisions and expansions with major contributions to the present version by Joachim De Beule, Martin Loetzsch, Remi van Trijp and Pieter Wellens. We gratefully acknowledge support from the EU FP6 and FP7 framework programs, particularly the ECAgents and Alear projects.

References

Bergen B, Chang N (2005) Embodied Construction Grammar. In: Östman JO, Fried M (eds) Construction Grammars: Cognitive Grounding and Theoretical Extensions, John Benjamins, Amsterdam, pp 147–190

Beuls K (2011) Construction sets and unmarked forms: A case study for Hungarian verbal agreement. In: Steels L (ed) Design Patterns in Fluid Construction Grammar, John Benjamins, Amsterdam

Bleys J, Stadler K, De Beule J (2011) Linguistic processing as search. In: Steels L (ed) Design Patterns in Fluid Construction Grammar, John Benjamins, Amsterdam

Copestake A (2002) Implementing Typed Feature Structure Grammars. CSLI Publications, Stanford

Dalrymple M, Kaplan R, Maxwell J, Zaenen A (eds) (1995) Formal issues in Lexical-Functional Grammar. CSLI Lecture Notes 47, CSLI, Stanford CA

Fillmore C, Kay P, O'Connor M (1988) Regularity and idiomaticity in grammatical constructions: The case of let alone. Language 64(3):501–538

Fried M (2009) Construction grammar as a tool for diachronic analysis. Constructions and Frames 1(2):261–290

Goldberg A (1995) A Construction Grammar Approach to Argument Structure. Chicago UP, Chicago

Goldberg A, Suttle L (2010) Construction grammar. Wiley Interdisciplinary Reviews: Cognitive Science 1(4):468–477

Kay M (1986) Parsing in functional unification grammar. In: Grosz B, Sparck-Jones K, Webber B (eds) Readings in Natural Language Processing, Morgan Kaufmann

Lieven E, Tomasello M (2008) Children's first language acquistion from a usage-based perspective. In: Robinson P, Ellis N (eds) Handbook of Cognitive Linguistics and Second Language Acquisition, Routledge

Michaelis L (2009) Sign-based construction grammar. In: Heine B, Narrog H (eds) The Oxford Handbook of Linguistic Analysis, Oxford University Press, Oxford, pp 155–176

Östman JO, Fried M (2004) Historical and intellectual background of construction grammar. In: Fried M, Östman JO (eds) Construction Grammar in a Cross-Language Perspective, John Benjamins Publishing Company, pp 1–10

Pollard C, Sag I (1994) Head-Driven Phrase Structure Grammar. University of Chicago Press, Chicago

Spranger M, Loetzsch M (2011) Syntactic indeterminacy and semantic ambiguity: A case study for German spatial phrases. In: Steels L (ed) Design Patterns in Fluid Construction Grammar, John Benjamins

Spranger M, Pauw S, Loetzsch M, Steels L (2012) Open-ended procedural semantics. In: Steels L, Hild M (eds) Language Grounding in Robots, Springer Verlag, New York

Stadler K (2012) Chunking constructions. In: Steels L (ed) Computational Issues in Fluid Construction Grammar, Springer-Verlag, Berlin

Steels L (1998) The origins of syntax in visually grounded robotic agents. Artificial Intelligence 103:133–156

Steels L (ed) (2011) Design Patterns in Fluid Construction Grammar. John Benjamins Pub., Amsterdam

Steels L (ed) (2012) Computational Issues in Fluid Construction Grammar. Springer-Verlag, Berlin

Steels L, De Beule J (2006) Unify and merge in Fluid Construction Grammar. In: Vogt P, Sugita Y, Tuci E, Nehaniv C (eds) Symbol Grounding and Beyond., Springer, Berlin, LNAI 4211, pp 197–223

van Trijp R (2011) Feature matrices and agreement: A case study for German case. In: Steels L (ed) Design Patterns in Fluid Construction Grammar, John Benjamins, Amsterdam

Wellens P (2011) Organizing constructions in networks. In: Steels L (ed) Design Patterns in Fluid Construction Grammar, John Benjamins, Amsterdam

Appendix

Example 3.

```
(def-cxn referring-expression (:label gram)
   ((?top
     (tag ?meaning (meaning (== (get-context ?src-det-np))))
     (sem-subunits (==p ?det-np-unit))
     (footprints nil))
    (?det-np-unit
     (sem-cat (==1 (sem-function ((value reference)
                                  (potential (== reference))))))
     (args (== (ref ?ref-det-np)
               (src ?src-det-np))))
    ((J ?referring-expression-unit ?top (?det-np-unit))
     ?meaning
     (footprints (referring-expression))
     (args (== (ref ?ref-det-np)))))
   <-->
   ((?top
     (syn-subunits (==p ?det-np-unit))
     (footprints nil))
    (?det-np-unit
     (syn-cat (==1 (syn-function (== (potential (== referring-expression))
                                     (value referring-expression)))
                   (morph (case/gender ((nom + ?nom-m ?nom-f ?nom-n)
                                        (gen - - - -)
                                        (dat - - - -)
                                        (acc - - - -)))))))
    ((J ?referring-expression-unit ?top (?det-np-unit))
     (footprints (referring-expression)))))
```