Notice

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

Wellens, Pieter (2011). Organizing Constructions in Networks. In Luc Steels (Ed.), *Design Patterns in Fluid Construction Grammar*, 181–201. Amsterdam: John Benjamins.

BibTeX:

```
@incollection{wellens2011networks,
   Author = {Wellens, Pieter},
   Title = {Organizing Constructions in Networks},
   Pages = {181--201},
   Editor = {Steels, Luc},
   Booktitle = {Design Patterns in {Fluid Construction Grammar}},
   Publisher = {John Benjamins},
   Address = {Amsterdam},
   Year = {2011}}
```

Organizing Constructions in Networks

Pieter Wellens

Abstract

Fluid Construction Grammar supports different ways to organize the inventory of constructions into networks. This is not just usable for descriptive purposes only. It plays an important role in streamlining the processes deciding which construction to consider first. Networks become increasingly more important as the complexity, multifunctionality, and size of a grammar grows. This chapter shows how networks of constructions are represented in FCG and how they are used to optimize language processing. Two examples are explored in more detail. The first example concerns family relations of specificity between constructions and the second one concerns conditional dependencies.

1. Introduction

Construction grammarians agree that constructions do not stand on their own but have various relations to each other and that these relations form part of the knowledge language users have of their language (Goldberg, 2003), (Croft & Cruse, 2004, chapter 10). One relation present in most approaches to construction grammar is the *taxonomic relation*, which captures a relation of schematicity between two constructions. Such taxonomic relationships form a network, rather than a simple tree, in the sense that each construction can have multiple specializations and a more specific construction can inherit from more than one schematic parent construction.

Besides taxonomic links Goldberg (1995) also proposes subpart, meronymy, and polysemy relations, the latter relating two constructions that have identical syntactic poles but differ in their semantics. Another type of relation is the metaphorical or prototype-extension relation between a central sense and its extended uses. These relationships have also been posited by Lakoff (1987, appendix 3), among others.

The properties these relations capture are all of a descriptive, content-based nature. For example, schematicity, meronymy and polysemy relationships are based on relational properties that hold between the content of the constructions involved in the relation. Nevertheless, cognitive linguists have argued that these relations emerge and are acquired from language use and they thus view them as *usage-based* (Tomasello, 2003). For example, schematicity is hypothesized to follow from a process of schematization (Langacker, 2000), and the prototype-extension relation follows from the act of extending the use of a construction.

Fluid Construction Grammar supports the organization of constructions in terms of *construction sets*. All constructions that cover a particular aspect of grammar, for example the lexicon, can be grouped together. FCG allows easy access to the constructions within a set and provides primitive operations to find, add and delete constructions. Sets can already be used to influence the search process because all constructions within a set are considered before the next set would be. Bleys et al. (2011) illustrated that templates can exploit schematicity relations because one construction can be built by taking an existing construction and adding more constraints to it. For example, the French nominal phrase construction can be defined in a schematic way, without yet determining the ordering of the adjective and the noun, and then there can be two more specific variants, one for adjectives in pre-nominal and another one for adjectives in post-nominal position.

This chapter extends the power of FCG by making it possible, not only to define *networks* of constructions based on relations between individual constructions, but to use this information as well in processing. In the construction network each node corresponds to a single construction and the edges linking the nodes are directed, i.e. they have a start and an end node, and they can have a label indicating which relation is intended, for example INHERITS-FROM, INSTANTIATES or EXTENDS. It is therefore possible to have multiple relations defined over the same set of constructions. Moreover, the relations between constructions are not only used for descriptive purposes but play an integral part in linguistic processing itself. For example, schematicity relations can be used to influence the order of application of the constructions within a set.

FCG is not only able to represent construction networks but also provides extra primitive operations for dealing with construction networks, most of which manage the edges that link the constructions. The FCG-interpreter maintains consistency in the network, a necessary operation since these structures can become quite intricate and are dynamic because constructions are being added and deleted by learning processes. For example, when a construction is deleted, the network automatically

removes all edges that link to or from that construction, in order to avoid loose edges. There are also operators for accessing all associated nodes from a given node (i.e. its neighbors in the network) and ways to view a construction network, creating the kind of graphical representations that will be seen later in this chapter.

To make the discussion concrete, this chapter considers two examples of construction networks and how they influence processing. The first example, discussed in Section 2, concerns the representation and handling of *families* of constructions. It will be shown that this poses challenges for language processing because the more general construction(s) can normally trigger in the same circumstances as the more specific ones, although they will only incompletely cover the sentence. In such cases we want the most specific constructions to have priority. This chapter shows how the organization of constructions in networks reflecting these family relationships solves this problem.

The second example, discussed in Section 3, looks at *conditional dependencies* between constructions. Such dependencies exist if one construction provides constituents or information that another construction needs. For example, a nominal phrase requires the presence of a nominal and it therefore does not make sense to look for the application of nominal phrase constructions if such a nominal could not be found. It turns out that these conditional dependencies can be automatically inferred from language use and then be employed to prioritize the order in which constructions are considered, effectively achieving the equivalent of priming. Experiments reported here show that this can lead to dramatic efficiency improvements.

2. Families of Constructions

Many constructions are closely related, and, what is thought of as one construction is in fact often a whole group of interrelated constructions. For example, Goldberg and Jackendoff have shown that the English resultative (as in ”she watered the plants flat”) is best viewed as a family of constructions (Goldberg & Jackendoff, 2004). There is not a single unified resultative but rather a central resultative construction which is further elaborated with subconstructions, sharing certain properties but differing in others. The notion of a 'family' echoes the work of Rosch & Mervis (1975) and Wittgenstein (1967) on the family structure of concepts.

FCG supports not only the representation of information about family structure but also the exploitation of such structures in language processing. This section begins by introducing a simplified example grammar fragment that focuses on the family of English transitive constructions (Section 2.1). The grammar fragment

serves as an illustration of how to represent family relations between constructions but should not be taken as the final word on how to model argument structure in English. Using this grammar fragment, this section then illustrates that having related constructions, but failing to explicitly encode their relations in a network structure, is problematic from a processing point of view (Section 2.2). The problem stems precisely from the many commonalities that the constructions share. This is followed by a procedure for defining family relations in FCG , including an explanation as to how such a network can aid linguistic processing (Section 2.3).

2.1. An Operational Example: Family of Transitive Constructions

Argument structure constructions have received quite a bit of attention from construction grammarians, because the marking of participants in an event is one of the most important aspects of language (See also the chapter by van Trijp (2011) later in this book). Argument structure constructions show great diversity both across different languages and within one language. One subset of argument structure constructions are the transitive constructions, which express a transitive action and include at least one direct object, as for example "He loved his car". In English, events are semantically subcategorized, in terms of cause-motion events, cause-receiver events, cause-become events, etc., and these give rise to a family of transitive constructions with the general transitive construction at its center.

In order to construct an interesting grammar fragment, we have used inspiration about possible transitive constructions and semantic categorizations from the FrameNet project (Fillmore, 1982; Baker et al., 1998) . FrameNet annotates lexical units (words and their different senses) with frames. These frames are stored as an inheritance network. For example, the verb *throw* evokes the CAUSE_MOTION frame, which itself inherits from the more general frame TRANSITIVE_ACTION. In turn, TRANSITIVE_ACTION is the child of the two very general frames EVENT and OBJECTIVE_INFLUENCE. Lexical constructions can be automatically generated from this information, including its frames for every lexical entry. The frame hierarchy is represented as a list of semantic categorizations. Here is an example for the lexical constructions THROW(here defined in terms of templates). Syntactic categorizations include the part of speech (`POS`) verb (`v`).

```
(def-lex-cxn throw-cxn
  (def-lex-skeleton throw-cxn
    :meaning (== (throw ?causer))
    :string "throws")
  (def-lex-cat throw-cxn
    :sem-cat (== cause_motion transitive_action
                 objective_influence event)
    :syn-cat (==1 (pos v))))
```

Grammatical constructions cannot be extracted automatically from FrameNet and thus still need to be coded by hand. The core construction of the family, namely the general transitive construction, reveals few surprises. It is a more specific case of the phrase structure constructions studied earlier in (Steels, 2011a). The construction requires three constituents, two referents (the actor and the undergoer) and an event categorized as TRANSITIVE_ACTION, which means that the construction can categorize any transitive action but does not do much more than link the participants as generic actor and undergoer to the event in a further unspecified manner.

The other subconstructions that are part of the family mirror the FrameNet frame hierarchy at the constructional level, which means that a variant of the transitive construction is created for each subframe of the core TRANSITIVE_ACTION frame according to the following process:

- The event constituent requires that the verb satisfies a set of semantic categorizations from the frame hierarchy. For example, the CREATING construction requires the frames CREATING, TRANSITIVE_EVENT, EVENT and OBJECTIVE_INFLUENCE.
- The two participant roles are linked with the roles appropriate to the frame. For example CREATING fuses `Creator` and `Created_Entity` for the actor and the undergoer respectively.
- In some cases additional meanings or frames are necessary. For example, the CAUSE_MOTION construction includes a path.

The TRANSITIVE_ACTION frame has over thirty subframes, resulting in equally many constructions. Examples are DESTROYING, CREATING, CAUSE_IMPACT, INTERRUPT_PROCESS and, of course, CAUSE_MOTION.

The grammar contains additional grammatical constructions beyond the family of transitives. In total there are an extra seventeen grammatical constructions, including the intransitive construction and some variants of it. These other constructions are not required for understanding the ideas in this section and are not further elaborated upon but are nevertheless needed to process the example utterances in the following sections. Further details about the conversion of Frame Semantic meanings to full FCG constructions is documented in (Micelli et al., 2009; De Beule & Micelli, To appear).

2.2. Why Related Constructions Are Problematic for Processing

Given such a grammar fragment, the problems encountered in linguistic processing can now be demonstrated. There are two problems.

The first one appears because one construction partly overlaps with another one. Following (Goldberg, 1995), we call this the SUBPART-relation. For example, the transitive construction overlaps with the intransitive construction, because the intransitive is entirely the same as the transitive in terms of form, except that it does not have a direct object. Consequently the intransitive is labeled as a subpart of the transitive. The utterance "the student writes" is intransitive whereas "the student writes a construction" is transitive. However in processing a split occurs after "writes". Either the intransitive construction should apply, or other material may still be coming possibly triggering the transitive construction. It would therefore be preferable if the processor first explores whether a transitive is possible instead of jumping already to the conclusion that the intransitive applies. Knowing that a subpart relation applies helps the processor to decide which of these constructions to consider first.

The second problem appears because one construction may be a more specific case of another construction but shares entirely the same form constraints. This relation is generally said to be one of *polysemy* between constructions. For example, "the student writes a construction" is an instance of the transitive construction, but it is also an instance of the more specific CREATING_CXN, which adds additional meaning and frame semantics to the transient structure. It would therefore be preferable that the processor first explores the relevant more specific constructions before the more general ones, and it can do this by exploiting the polysemy relation.

Both problems are illustrated in Figure 1, which depicts an (inelegant) search tree for parsing the utterance "The student writes a construction", taken from an operational implementation. Only two out of seven final leaf nodes are successful,

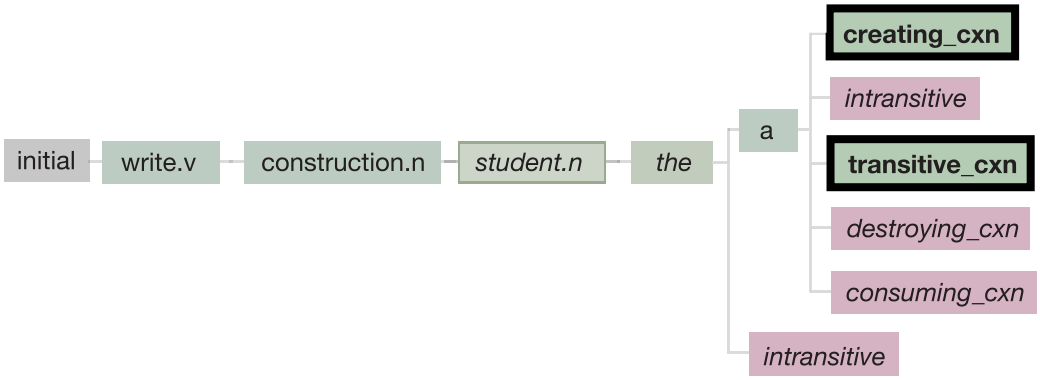


Figure 1. *Search tree for parsing the phrase "The student writes a construction" without any encoding or exploitation of network structure between related constructions. To reduce the size of the Figure, only a fragment of the family of transitive constructions is shown. Only the final constructions with a bold border,* CREATING_CXN *and* TRANSITIVE_CXN*, led to a successful parse.*

being CREATING_CXN and TRANSITIVE_CXN. The other final constructions only partially apply in that their conditional pole, here their form pole, matched, but their semantics later turned out to be incompatible in merging. When traversing the tree left to right, the first four nodes show the application of four lexical constructions, followed by a first split in the tree. This split occurs because two constructions, INTRANSITIVE and A (triggered by the presence of the word "a"), both match with the transient structure. The INTRANSITIVE matches because its form *overlaps* with the form of transitive constructions as it only requires two constituents, one for the event and one for the actor. The final split in the search tree would normally show all the variants of the transitive construction that match, but here it has been reduced to show only five, plus the intransitive again. Problematically, most of the constructions in the family of transitives are polysemous and thus match on the same form. Only later, during merging, is FCG capable of finding that TRANSITIVE_CXN and CREATING_CXN are fully compatible and can be merged. A third and final problem is that the CREATING_CXN is not preferred over TRANSITIVE_CXN, while it does lead to a more precise interpretation.

2.3. Solving Processing Problems by Defining a Network

The previous section demonstrated that related constructions lead to disordered linguistic processing, which is especially unsatisfactory because the opposite is expected, namely that the constructions would be processed more efficiently. This section shows that this outcome is indeed possible but only when explicitly coding the relations in a network and allowing these relations to influence processing. We start by showing how a grammar designer can define relations between constructions in FCG , which is applied here to the example family of transitives.

Fluid Construction Grammar provides a primitive template `link-constructions` that specifies a link between two constructions with a directed, labeled edge as follows:

```
(link-constructions :start creating_cxn
                    :end transitive_cxn
                    :label polysemy)
```

The slots :start and :end are filled with the names of the two constructions that are linked in a network and the :label indicates the kind of relation between them.

Follwing the types of connections proposed by Goldberg (1995), we can use these network operations to manually define the relationships between the constructions of the example grammar fragment. The intransitive construction is first linked to the transitive construction with a SUBPART relation:

```
(link-constructions :start intransitive
                    :end transitive_cxn
                    :label form_subpart)
```

Next, all of the subconstructions of the family of transitives are linked to the general transitive construction with a POLYSEMY link, which in this case also denotes INHERITANCE. An example of this operation is already given above for CREATING_CXN and TRANSITIVE_CXN. Using these operations results in a network of which a fragment is shown in Figure 2. FCG includes software to render these networks automatically.

How can the network shown in Figure 2 be put to use during processing so as to avoid the problems illustrated in Section 2.2? We need to look at both the competition between constructions based on the SUBPART relation, as between the intransitive and the transitives and, the competition among the polysemous transitives.

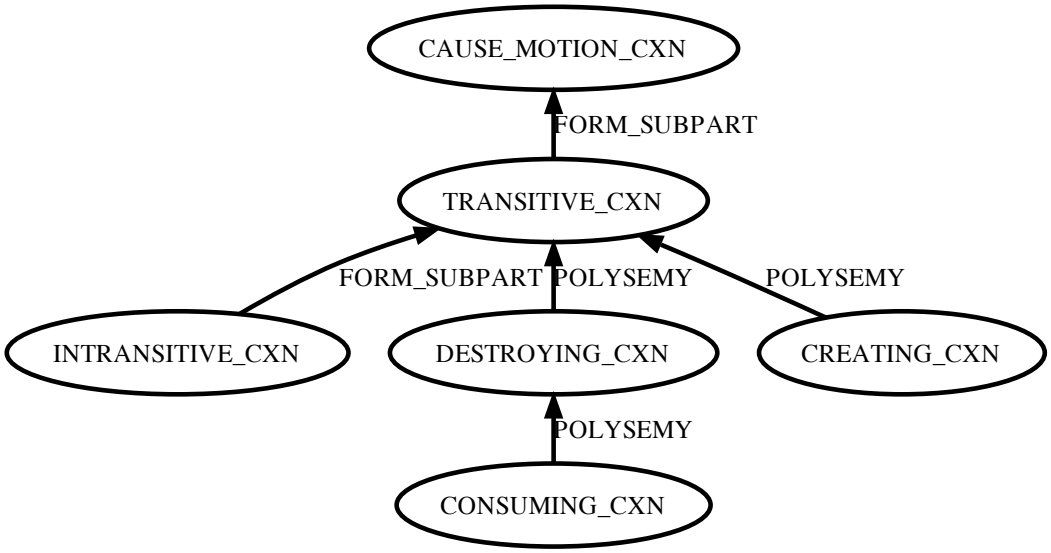


Figure 2. *A fragment of a construction network in* FCG . *Shown here are two different types of edges, the* FORM_SUBPART *edge for intransitive and transitive, the* POLYSEMY *edge for most of the constructions in the transitive family.*

Competition between constructions can be minimized by *prioritizing* the application of constructions based on their relationships:

1. The FORM_SUBPART relation prioritizes the more elaborate construction. Thus the transitive construction would gain priority over the intransitive one because the intransitive construction forms a subpart of the transitive one.

2. The POLYSEMY relation prioritizes the more specific constructions. For example CREATING_CXN has priority over TRANSITIVE_CXN.

3. For non-ambiguous utterances the subconstructions of the same construction are all mutually exclusive in application. Their processing can be halted as soon one applies successfully. For example, when CREATING_CXN has successfully applied on a certain part of the utterance, it is not necessary to still try for example DESTROYING_CXN on that same part.

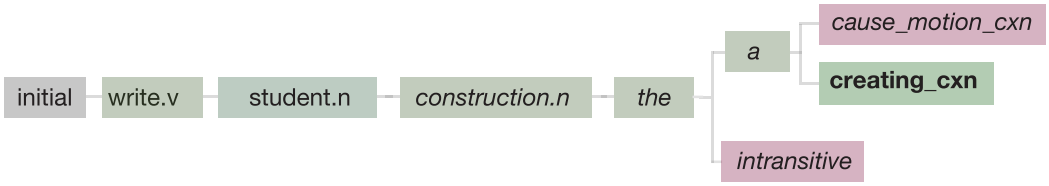


Figure 3. *Search tree for parsing the phrase "The student writes a construction" with the use of the network structure shown in Figure 2.*

Using the network shown in Figure 2 and the principles described above, FCG is able to reduce the search tree from Figure 1 to the one shown in Figure 3. The first principle has inhibited the transitive from applying in the final branch because CREATING_CXN has priority over it. The same holds for the intransitive, which also no longer clutters the final branch. There are also less competing transitive subconstructions, because the successful application of the creating construction signals they no longer need to be considered. The first branch that is still there is unavoidable, given the design of the grammar and the current network. At that point in the application process, the intransitive construction matches with the transient structure, since it has processed the constructions WRITE.V, STUDENT.N and THE and, as such, contains the necessary constituents (event and actor).

3. Streamline Linguistic Processing through Dependency Networks

The previous section showed how to define a network structure for families of constructions and how this network can significantly aid linguistic processing by prioritizing the constructions. This section looks at another example which is based on detecting and exploiting *conditional dependencies*. Such a dependency exists between two constructions X and Y if X generates some of the preconditions for the triggering of Y. For example, the CAUSE_MOTION_CXN from the previous section requires a total of four constituents, two referents, a predicate and the event, categorized as evoking the CAUSE_MOTION frame. These constituents and their categories need to have been supplied by constructions that applied earlier in the chain of constructions. From a processing point of view, this information can therefore be exploited in two ways: it is not necessary, firstly, even to try CAUSE_MOTION_CXN before the constituents have been supplied and, secondly, it is possible to prioritize

or prime a construction as soon as the constructions that establish its constituents have been active.

Interestingly, the conditional dependencies between constructions can be learned automatically by the FCG-interpreter as a side effect of the normal process of parsing and production, as experiments in this section will show. It suffices to keep track of which constructions co-apply and translate that into dependency relations.

The remainder of this section has four parts. The first subsection shows how networks based on conditional dependencies can drastically reduce search (Subsection 3.1). Next we look at how these dependencies can be learned or inferred from processing (Subsection 3.2), and finally how they can subsequently aid minimizing search by priming the most likely constructions (Subsection 3.3). A final subsection reports experimental results in testing these mechanisms (Subsection 3.3.2).

3.1. Bottlenecks in Constructional Processing

As explained in earlier chapters (Steels, 2011b), (Bleys et al., 2011), constructions apply on the transient structure in a two-phase process. The first phase, called the matching phase, does not add any information to the transient structure. Only when the conditional pole matches, which most often is not the case, does the second phase of application begin. In this phase the transient structure is altered by merging the J-units of the conditional pole of the construction with the transient structure (first merge) and then merging the complete contributing pole (second merge). A construction only applies completely if both phases (matching and merging) are successful. Because the application of a construction is only determined by its success in matching and merging with the transient structure, it follows that the order of application of constructions does not straightforwardly map to the surface order (i.e. the words in the sentence).

A striking feature of constructional processing (be it production or parsing) is that, in the end, only a fraction of all constructions is used in producing or parsing a sentence. Indeed, the whole point of the search process is to find this very small subset as an ordered sequence. However, to guarantee completeness and find this small subset, the default search process needs to iterate over all constructions in every expansion of a search node until it finds an applicable construction. Without optimizations, the search process needs to match, on average, half of the constructions in every expansion.

This last point can be illuminated through the example of parsing the phrase "next to the house". The same grammar is used as introduced in Section 2 and explained in detail in (Micelli et al., 2009; De Beule & Micelli, To appear). The grammar counts a total of 116 constructions, of which 94 are lexical including nouns, adjectives, prepositions and verbs, with the remaining 22 being grammatical. The four constructions involved in processing the phrase are introduced in detail in the next section, but, for the present purpose, only the search tree as shown in Figure 4 is of importance. Every node depicts a successfully matched and merged construction, but what is not shown are all the constructions that had been tried to match and that failed. The amount of failed constructions (the number in brackets) is shown for every node in the tree. With a simple calculation, it follows that in total over 350 constructions have matched, with an average of 87 constructions being matched in each node in the tree. The latter number is indeed the number expected for a total of 116 constructions.

Figure 4. *Search tree for parsing the phrase "next to the house" involving four constructions.The bracketed numbers in each node depict the amount of constructions that tried to match before finding the applicable one. Dependencies help to reduce this number.*

The default search process, which uses just plain sets of constructions, has no means of prioritizing certain constructions as being more likely to apply than others. One straightforward improvement is to order the constructions on token frequency but, on its own, this is still limited. This modification still treats the constructions as independent entities with no relation to one another. When looking again at the search tree in Figure 4 we see that THE and SPATIAL_PHRASE apply last, which, of course, is no accident since these constructions require the categories supplied by the earlier constructions. Just as THE requires a noun, so does SPATIAL_PHRASE have a conditional dependency on the preposition *next to* and on a referring expression, which, in this case, is realized by the combination of THE and a noun. Recording these dependencies in a network creates the network depicted in Figure 5.

As will be shown in the next subsection, such a network structure can drastically optimize linguistic processing and actually transforms the search from a standard

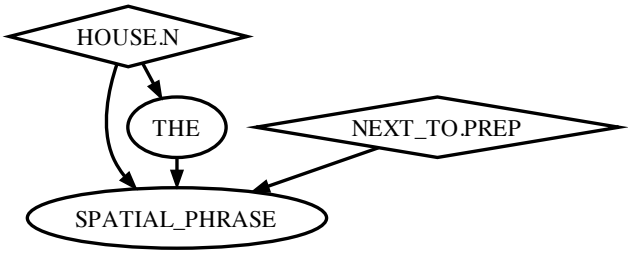


Figure 5. *Network capturing the dependencies between the constructions when parsing the phrase "next to the house". The semantics of the edges could be read as meaning 'supplies a dependency for'. For example* HOUSE *supplies a dependency for the phrasal construction* THE.

heuristic process to a much more plausible process of activation triggered by the constructions and mediated by the categories and frames.

3.2. Learning Constructional Dependency Networks

Before demonstrating that conditional dependencies indeed drastically improve processing, let us investigate how these dependencies can be induced from language use by inspecting the interplay between the constructions. A dependency is found when one construction merges information in the transient structure that is later required during matching by another construction, as illustrated in Figure 6.

The first construction shown in Figure 6 is the lexical construction for HOUSE. The construction is quite basic and links the predicate `(house ?buil)` to a lexeme "house". Of interest here is what this construction merges in the transient structure. Focusing on the syntactic pole (the right pole), we can see in the added box that the construction merges `(lu (pos n))`, which is extracted from FrameNet and denotes that it is a lexical unit and that its part of speech is noun. Following the line to the connected box, we end up in a constituent unit of the construction for THE. During parsing this construction matches only when there is a unit containing `(pos n)`, which in the current example is supplied by HOUSE. It can thus be concluded that in parsing the phrase "next to the house", THE depends on HOUSE, because HOUSE supplies (merges) the syntactic category `(pos n)` which is required (matched) by THE.

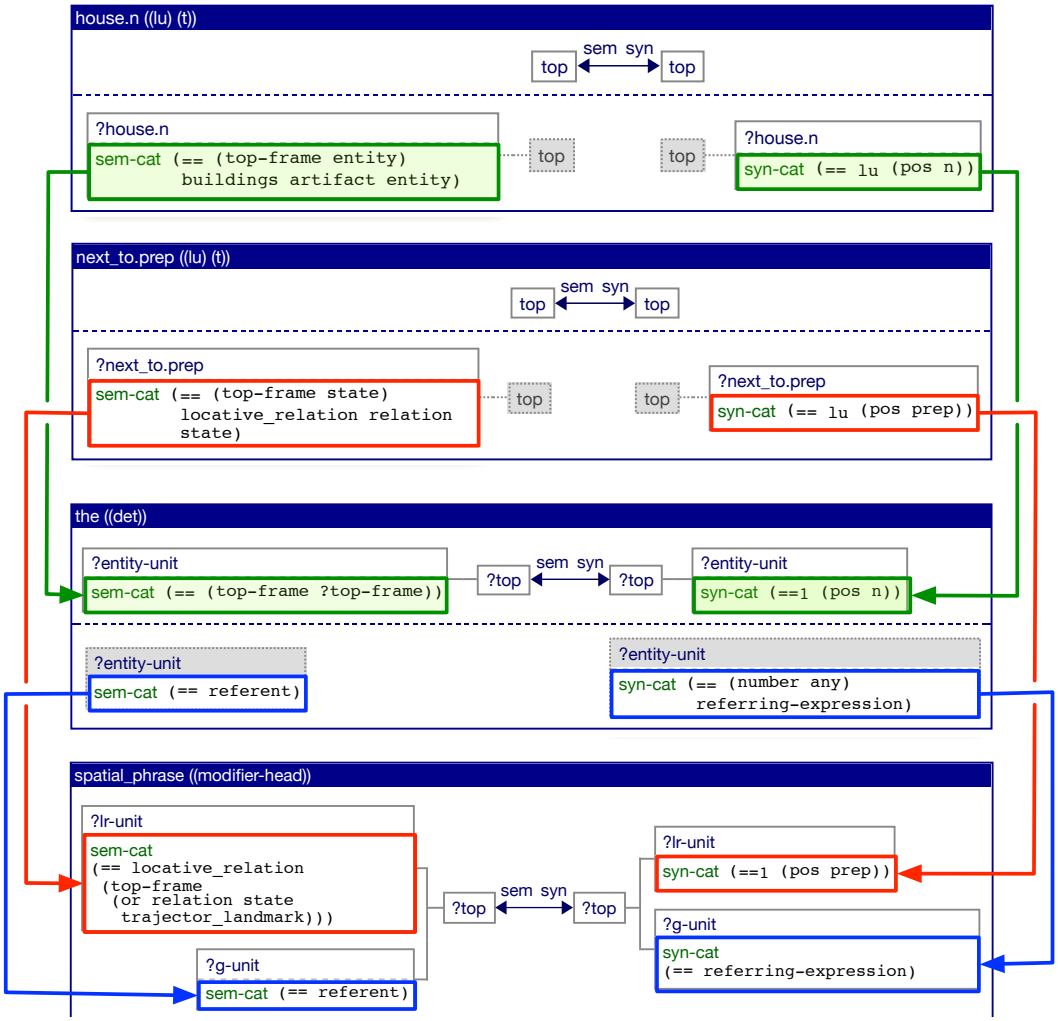
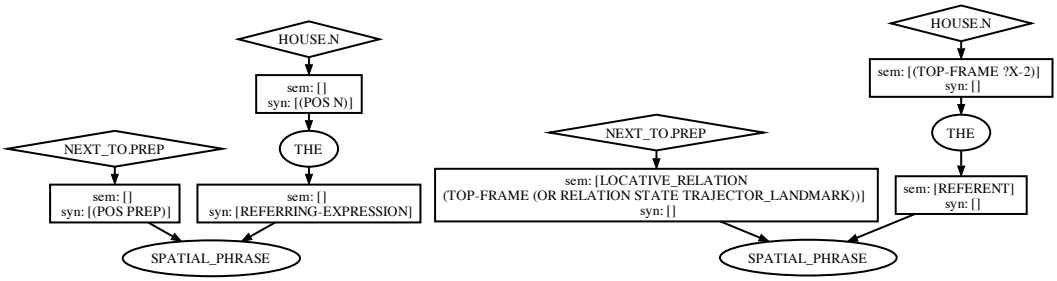


Figure 6. *The four constructions required for producing and parsing the utterance "next to the house". For readibility only the syntactic and semantic categories (`syn-cat-sem-cat`) are shown. In each construction the categories below the dotted line are introduced (merged) by the construction,the categories above the dotted line are required (matched). The dependencies are visualised by connected boxes. For example* NEXT_TO *introduces `syn-cat (lu (pos prep))` which is required by one of the two constituents in* SPATIAL_PHRASE. *As such for this utterance* SPATIAL_PHRASE *thus partially depends on* NEXT_TO.

The exact same reasoning holds for the other dependencies. Again, in Figure 6, we see that the bottom construction SPATIAL_PHRASE has two dependencies in parsing (the boxes in the right pole). One is a preposition (`pos prep`), here supplied by NEXT_TO, the other a referring expression provided by THE.

The conditional dependencies differ for parsing and producing, and, consequently, two separate dependency networks are needed, even though the constructions they link are the exact same. Having two different networks produces an interesting twist in the reversibility of FCG processing. The constructions themselves are fully reversible, but this does not imply that the order in which they are applied, or their conditional dependencies, have to be the same.



(a) Parsing Network (b) Production Network

Figure 7. *The constructional dependency networks that can be inferred after parsing (left) and producing (right) the phrase "next to the house". Diamond shaped nodes (*HOUSE.N *and* NEXT_TO.PREP*) represent constructions that have been found to be independent. Egg shaped nodes (*THE *and* SPATIAL_PHRASE*) are constructions that are dependent on the incoming categories (rectangular nodes).*

FCG thus infers two networks for the example phrase "next to the house". Both networks are shown in Figure 7, with the network captured from parsing at the left. The different syntactic or semantic categories that establish the conditional dependency links are also shown.

FCG updates the correct dependency network after every usage event. When new constructions are used, they are added to the network. Either they are dependent on specific categories, in which case they are linked appropriately, or they are applied independently of any other constructions, in which case the network also records them as such. Existing constructions might also be used in novel ways, which then results in extending their links with other constructions in the network. Even when the network does not require any extension, it is still altered by every

use, since the links that have been encountered become more entrenched. Figure 8 shows a fragment of the network after processing a series of sentences.

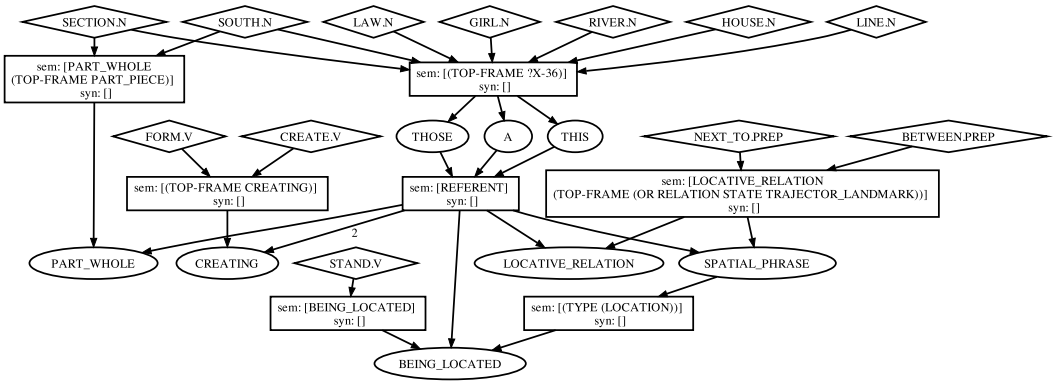


Figure 8. *Fragment of the constructional dependency network for production learned from a series of usage events. Just as before, square nodes contain (semantic) categories. The other nodes represent dependent and independent constructions.*

By delving into the intricate interplay between matching and merging, we have shown through this example how FCG can track dependencies between constructions. In the following section, we explain how dependency networks are exploited by FCG to optimize processing.

3.3. How Dependency Networks Streamline Linguistic Processing

3.3.1. *Network-Driven Linguistic Processing Explained*

Dependency networks can be used in processing by priming or prioritizing constructions whose dependencies (incoming categories) have been realized in the transient structure. For example, even the small network shown in Figure 7 (a) indicates that it is wothwhile in parsing to try the SPATIAL_PHRASE construction when a preposition and a referring expression have been merged in the transient structure. Constructions can thus be seen as 'communicating' with each other during processing through the categories that they require (from previous constructions) and that they supply (to later constructions). Categories thus become the main regulators of linguistic processing.

Constructions for which all dependencies in the network have been met get primed, which means that they get prioritized over all other non primed construc-

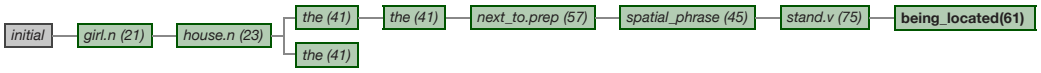
tions. When a construction applies, its outgoing edges fire and activate the connected categories. For the network in Figure 7 (a) for example, after the HOUSE construction applies, the connected (POS N) category gets activated. The active categories then determine the priming of the following constructions by prioritizing all constructions for which all incoming categories are active. Returning to the example, because the (POS N) category is active, the THE construction can now be primed, since (POS N) is its only incoming category.

As the network grows, multiple constructions can be primed at the same time. Even these primed constructions are internally ordered, since, in the network, every edge is scored, denoting the amount of times that that edge has led to a successful prime. The non-primed constructions also have an internal ordering: firstly, all of the known independent constructions, and then all the remaining constructions, both of which are ordered on their token frequency. These ordering principles result in a highly accurate priming of constructions. Over time, the network only becomes more and more accurate.

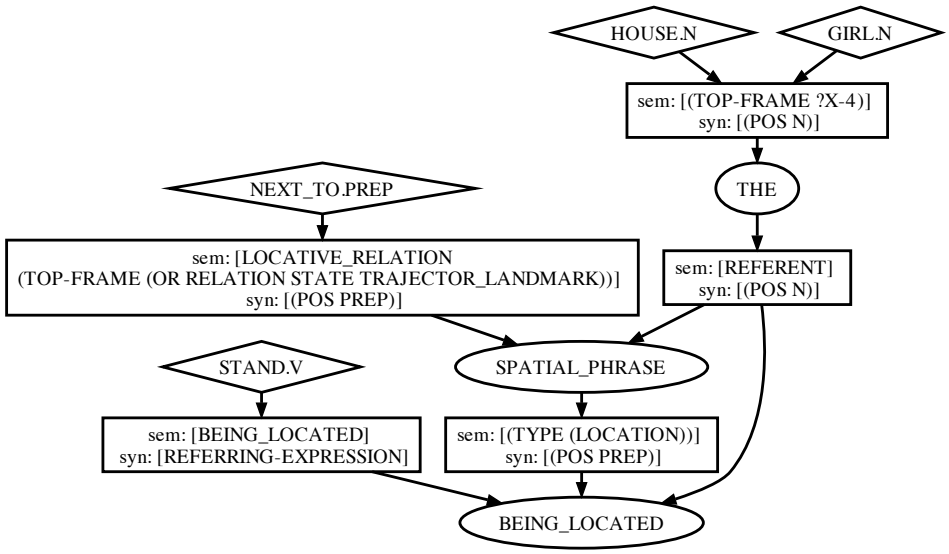
As such, the linguistic processing is based more on activation patterns in the network, where constructions fire their outgoing edges, subsequently activating their outgoing categories, which in turn prime new constructions. The applications of the constructions themselves are what drive the application of the subsequent constructions through the categories that they supply. The default FCG search (Bleys et al., 2011) requires that in every expansion, each construction is checked as to whether or not it can match with the transient structure. With the aid of the conditional dependency network, the search can be changed quite radically from a best-first search to a search process regulated by the network itself.

3.3.2. *The Impact of Dependency Networks on Processing*

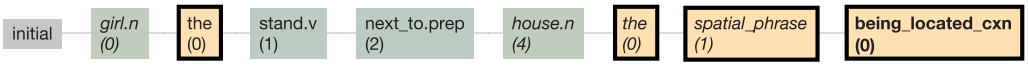
Although the application process discussed in Section 3.1 looked streamlined, it became evident that, behind the scenes, FCG needed to match many constructions to find the correct one (Figure 4). When this example is expanded to include the slightly more complicated phrase "the girl stands next to the house", seven constructions are required. Akin to Figure 4, Figure 9(a) shows the search tree for producing the new example utterance. This application process is done *without* the aid of a dependency network. What is interesting is, again, not the tree itself but the numbers inside the individual nodes. These indicate the amount of constructions tried for application (and that failed) before the shown construction was applied. For example, before the first applied construction (here, the lexical construction GIRL.N), 21 other constructions had been unsuccessfully tried for matching. This number is



(a) Search Tree without Priming



(b) Constructional Dependency Network



(c) Search Tree using Dependency Priming

Figure 9. *Shown here at the top is the search tree for producing the utterance "the girl stands next to the house" and in the middle the network that can be inferred from it. Most instructive here are the numbers in the nodes of the search trees, right after the construction name, since they denote the number of constructions that failed to match before the final, successful match. At the bottom we again see the search tree but this time, when using the inferred dependency network. Search tree (a) has an average of* 46 *failed construction applications per node, while (c) only has an average of* 1.

even quite small since the average for this tree is 46. This number should, of course, be as small as possible.

The network shown in the middle of Figure 9 is automatically learned by FCG after processing the utterance. The egg-shaped, and diamond-shaped nodes represent the seven constructions and the square nodes, the mediating categories. This network is quite illuminating even without knowing or understanding the inner workings of the constructions, as it shows, for example, that the BEING_LOCATED construction (at the bottom of the network), which is a sort of extended intransitive construction, has three distinct dependencies. By tracing upwards in the network, the network shows which constructions supply these dependencies.

Using this network during processing results in the reduced search tree shown in Figure 9(c). Some of the nodes have a bold border, indeed, only the grammatical ones, which indicates that the construction was primed by the network, while also explaining why the number of failed constructions is so low for these constructions. The other constructions, namely, the lexical constructions, also have reduced the number of failed attempts because of the ordering on token frequency. For this example utterance, the network achieves a reduction from an average of 46 matched constructions per node to slightly more than one.

The question that remains is whether or not this network approach can also scale when processing hundreds of sentences covering all 116 constructions of the benchmark set derived from FrameNet. For this experiment, we produced and parsed five hundred valid sentences, updating the network after every sentence. Figure 10 shows, for every grammatical construction, the evolution of the number of constructions that failed to match before the successful grammatical construction was applied. Obviously, the standard search does not improve over time and maintains an average that is around half of the total number of constructions. In contrast, after three hundred completed production or parsing processes, the dependency network was fully learned and the number of failed constructions was reduced to only two. This result means that on average, for grammatical constructions, the second construction that is tried is part of the successful branch.

4. Conclusion

We have looked at two examples of how network structure can organize the inventory of constructions. The first example centered around the notion of a family of constructions. It is beneficial to explicitly encode the different types of relationships between related constructions, to avoid that more general constructions interfere with the consideration of more specific constructions. The second example

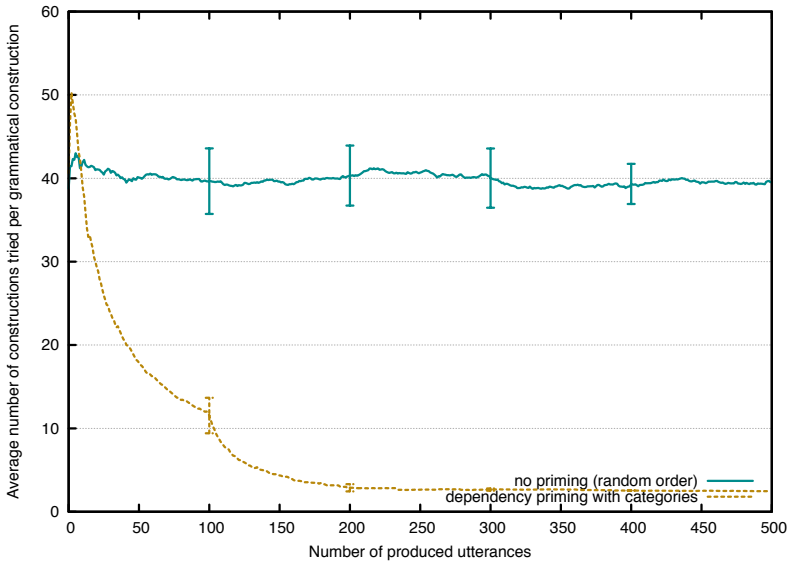


Figure 10. *Evolution of the average number of grammatical constructions matched per construction applied while processing a new sentence. The top curve represents the baseline case corresponding to the default search, amounting, on average, to about half of all constructions being matched each time. The bottom curve shows the improvement of using dependency based priming, amounting to a substantial drop from around forty to around two constructions. Note that this number is also more stable.*

concentrated on explicitly representing conditional dependencies between constructions in a network. It was shown that the exploitation of this structure has a dramatic effect on the efficiency of the search process in FCG.

There is a growing body of evidence supporting the hypothesis that acquiring a language involves the learning of usage-based dependency patterns among constructions Tomasello, 1992; Saffran et al., 2008. There is also evidence that acquired patterns of constructional usages influence language processing, for example through the priming of frequently co-occurring constructions Tomasello et al., 1997; Saffran, 2001. This chapter has shown that both observations have been successfully operationalized in Fluid Construction Grammar. This process involves the learning of constructional dependency networks from randomly generated but valid

sentences according to an FCG grammar documented in Micelli et al., 2009. As demonstrated, such networks can be learned and can reduce the amount of processing required for parsing or producing a sentence. Additionally, I have proposed to explicitly include semantic and syntactic categories in the network, which provides the glue between constructions, and I have shown how this leads to a powerful capacity to generalize from observations and to an associated further reduction of processing load. In Wellens & Beule (2010) we have investigated the impact of not explicitly representing categories in the network but instead linking the constructions direclty. There it was shown that representing the categories was beneficial both in terms of accurracy of priming and stability of the network. Besides aiding in processing Steels et al. (2007) have shown that a network structure can help in the formation of a systematic grammar.

The dependency networks naturally give rise to an interplay of lexical and grammatical processing. As soon as dependencies are met grammatical constructions are primed and can apply giving rise to more incremental processing where lexical and grammatical constructions work together in a non-sequential manner.

The processing efficiency and accuracy in humans is nothing short of amazing, and the mechanisms regulating our capacity for language have clearly been under strong evolutionary pressure. If not, language processing would have become too slow or the inventory of constructions could not expand to more than a limited lexicon. The dependency network and the results presented in Section 3 show that language does indeed contain structure that allows one to build and shape a constructional dependency network, which can be used to optimize both efficiency and accuracy. Moreover, there is a nontrivial connection between the dependencies we tracked in our networks and the hierarchical structure of language, thus hinting that the recruitment of these general cognitive capabilities might be a necessary requirement to learn and process hierarchical large scale language systems (Steels, 2007).

Acknowledgements

This research was conducted at the AI laboratory of the Vrije Universiteit Brussel. The author was financed by the EU projects ALEAR and EUCOG II, and as a research assistent at the VUB Computer Science Department. I would like to thank the reviewers for their insightful comments.

References

Baker, Collin F., Charles J. Fillmore, John B. Lowe (1998). The Berkeley FrameNet Project. In *Proceedings of the 17th international conference on Computational linguistics*. Morristown, NJ, USA: Association for Computational Linguistics.

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

Croft, William, D. Alan Cruse (2004). *Cognitive Linguistics*. Cambridge Textbooks in Linguistics. Cambridge: Cambridge University Press.

De Beule, Joachim, Vanessa Micelli (To appear). Who framed fluid construction grammar? In Hans C. Boas (Ed.), *Computational Approaches to Construction Grammar and Frame Semantics*. Amsterdam: John Benjamins.

Fillmore, Charles J. (1982). Frame semantics. In *Linguistics in the Morning Calm*, 111–137. Seoul.

Goldberg, Adele, Ray Jackendoff (2004). The English resultative as a family of constructions. *Language*, 80(3), 532 – 568.

Goldberg, Adele E. (1995). *A Construction Grammar Approach to Argument Structure*. Chicago: Chicago UP.

Goldberg, Adele E. (2003). Constructions: A new theoretical approach to language. *Trends in Cognitive Science*, 7(5), 219–224.

Lakoff, George (1987). *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. Chicago: The University of Chicago Press.

Langacker, Ronald W. (2000). A dynamic usage-based model. In Michael Barlow, Suzanne Kemmer (Eds.), *Usage-Based Models of Language*, 1–63. Chicago: Chicago University Press.

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

Rosch, Eleanor, B. Catlin Mervis (1975). Family resemblances: Studies in the internal structure of categories. *Cognitive Psychology*, 7, 573 – 605.

Saffran, Jenny, Marc Hauser, Rebecca Seibel, Joshua Kapfhamer, Fritz Tsao, Fiery Cushman (2008). Grammatical pattern learning by human infants and cotton-top tamarin monkeys. *Cognition*, 107, 479–500.

Saffran, Jenny R. (2001). The use of predictive dependencies in language learning. *Journal of Memory and Language*, 44, 493–515.

Steels, Luc (2007). The recruitment theory of language origins. In C. Lyon, C. Nehaniv, A. Cangelosi (Eds.), *The Emergence of Communication and Language*, 129–151. Berlin: Springer Verlag.

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

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

Steels, Luc, Remi van Trijp, Pieter Wellens (2007). Multi-level selection in the emergence of language systematicity. In Fernando Almeida e Costa, Luis M. Rocha, Ernesto Costa, Inman Harvey (Eds.), *Advances in Artificial Life (ECAL 2007)*, LNAI 4648, 421–434. Berlin: Springer.

Tomasello, Michael (1992). *First Verbs: A Case Study of Early Grammatical Development*. Cambridge: Cambridge University Press.

Tomasello, Michael (2003). *Constructing a Language. A Usage Based Theory of Language Acquisition*. Harvard University Press.

Tomasello, Michael, Nameera Akhtar, Kelly Dodson, Lauren Rekau (1997). Differential productivity in young children's use of nouns and verbs. *Journal of Child Language*, 24, 373–87.

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

Wellens, Pieter, Joachim De Beule (2010). Priming through constructional dependencies: a case study in fluid construction grammar. In *The Evolution of Language (EVOLANG8)*, 344–351. World Scientific.

Wittgenstein, Ludwig (1967). *Philophische Untersuchungen*. Frankfurt: Suhrkamp.