Constructions at Work! Visualising Linguistic Pathways for Computational Construction Grammar

Sébastien Hoorens [a] Katrien Beuls [a] Paul Van Eecke [ab]

[a] *Artificial Intelligence Lab, VUB, Pleinlaan 2, 1050 Brussels, Belgium*
[b] *Sony Computer Science Laboratories, 6, Rue Amyot, 75005 Paris, France*

Abstract

Computational construction grammar combines well-known concepts from artificial intelligence, linguistics and computer science into fully operational language processing models. These models allow to map an utterance to its meaning representation (*comprehension*), as well as to map a meaning representation to an utterance (*formulation*). The processing machinery is based on the unification of usage-patterns that combine morpho-syntactic and semantic information (*constructions*) with intermediate structures that contain all information that is known at a certain point in processing (*transient structures*). Language processing is then implemented as a search process, which searches for a sequence of constructions (*a linguistic pathway*) that successfully transforms an initial transient structure containing the input into a transient structure that qualifies as a goal. For larger grammars, these linguistic pathways become increasingly more complex, which makes them difficult to interpret and debug for the human researcher. In order to accommodate this problem, we present a novel approach to visualising the outcome of constructional language processing. The linguistic pathways are visualised as graphs featuring the applied constructions, why they could apply, with which bindings, and what information they have added. The visualisation tool is concretely implemented for Fluid Construction Grammar, but is also of interest to other flavours of computational construction grammar, as well as more generally to other unification-based search problems of high complexity.

1 Introduction

A crucial first step in natural language understanding and production is the implementation of a system that can reliably map between an utterance and its meaning representation. The utterance is used for transferring a meaning representation from one human or artificial agent to another, and the meaning representation is used for interpreting the utterance in a given grounded or textual context [17]. Computational construction grammar is a field of study that aims to tackle this challenge by combining insights from linguistics, artificial intelligence and computer science into fully operational, bidirectional language processing models. Linguistically, the models are inspired by the basic principles of construction grammar [4, 5, 7, 3, 6], in particular the tight integration of syntax and semantics, the lexicon-grammar continuum and the use of multiple perspectives, such as phrase structure, functional structure, case structure and information structure. From artificial intelligence and computer science, the models borrow core concepts, such as problem solving through search [11, 12] and the unification of feature structure representations [13, 14, 10].

Computational construction grammars implement language processing as a search process, with operators, called *constructions*, expanding intermediate structures until a solution is found. For larger grammars, these intermediate structures can easily consist of "dozens of units and hundreds of features" [19]. Each intermediate structure has been shaped by a whole range of

constructions of different types, each construction contributing units or features to the analysis. As constructions fit together like the pieces of a puzzle, it often happens that certain preconditions of a construction have been contributed by one earlier construction, and others by a different one. Such constructional dependencies can be hard to grasp, yet it is important that they are correctly captured by the grammar engineer.

In order to enhance the human readability of computational construction grammar analyses, we introduce a tool that clearly visualises the processing results as a graph, featuring the applied constructions, why they could apply, with which bindings, and what information they have contributed. The tool is designed to help grammar engineers develop and debug grammars, as well as to help regular users inspect and interpret the results of a computational construction grammar analysis. In this way, the tool will help to address one of the biggest challenges faced by computational construction grammars, namely scaling up for attaining larger coverage.

There are multiple computational construction grammar implementations available, of which *Embodied Construction Grammar (ECG)* [2, 1] and *Fluid Construction Grammar (FCG)* [15, 16] are the most advanced projects. Our visualisation tool has concretely been implemented for Fluid Construction Grammar, but the proposed visualisations should be easily transferable to other computational construction grammar implementations and can also be applied to other unification-based search problems of high complexity. This generality is demonstrated in this paper by applying the tool to a planning problem.

The paper is structured as follows. First, we introduce the necessary basics of bidirectional language processing in Fluid Construction Grammar. Then, we present the design and implementation of the visualisation tool and demonstrate the tool in a use case. Finally, we show that the tool can also be used to visualise other problems than language processing, by demonstrating its application to a planning problem. The paper is supported by an interactive web demonstration, which can be accessed via `https://www.fcg-net.org/demos/visualising-pathways`.

2 Bidirectional Language Processing using FCG

2.1 Language Processing as a Problem Solving Process

Fluid Construction Grammar (FCG, `https://www.fcg-net.org`, [15, 16]) is an open-source computational construction grammar formalism and implementation, designed for mapping between utterances and their meaning representation. It is a bidirectional formalism in the sense that it uses the same grammar and processing mechanisms for both *comprehension*, i.e. mapping from a form to a meaning representation, and *formulation*, i.e. mapping from a meaning representation to a form. FCG implements language processing as a problem solving process [18], consisting of the following main components:

- *Transient Structures.* Transient structures are the state representations in the search problem. A transient structure is a feature structure that contains all information (morpho-syntactic, semantic, pragmatic, ...) that is known at a certain point in processing.
- *Initial Transient Structure.* The initial transient structure is the root of the search problem and is computed directly from the input. In comprehension, the initial transient structure contains a representation of the strings in the input and of the internal ordering between the strings. In formulation, the initial transient structure contains the predicates of the meaning representation that needs to be formulated.
- *Constructions.* Constructions are the operators in the search problem. A construction can apply to a transient structure if it *matches* it, i.e. if there are no conflicts when unifying the *conditional part* of the construction with the transient structure. If the construction matches the transient structure, it can expand the transient structure by *merging* the information from its contributing part into it.
- *Goal Tests.* Goal tests compute whether a given transient structure qualifies as a solution.

The task of the FCG system is to find a sequence of constructions that can expand the initial transient structure into a transient structure that qualifies as a goal. This sequence of

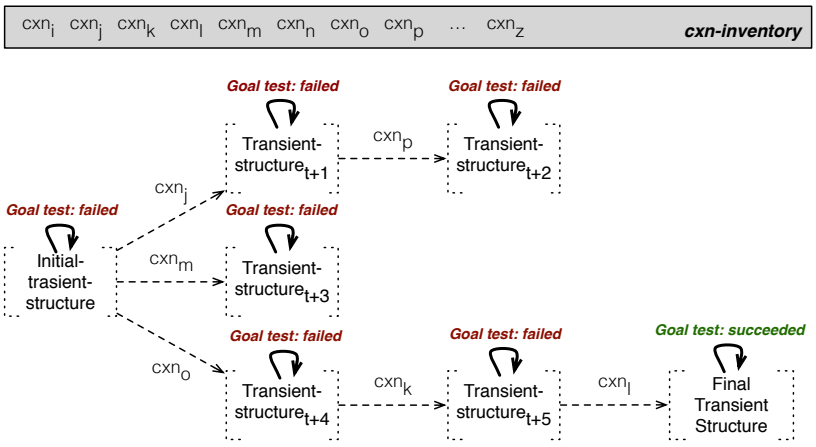

Figure 1: FCG implements language processing as a search process, in which constructions (cxns for short) expand an initial transient structure until a solution is found. A sequence of construction applications that leads to a solution is called a *linguistic pathway*.

construction applications is called a *linguistic pathway*. In order to make the search process computationally feasible, different heuristics and optimization strategies are used, which fall however outside the scope of this paper. A schematic representation of how FCG implements language processing as a problem solving process is shown in Figure 1. For studying FCG in more detail, we recommend playing around with the interactive web service at `https://www.fcg-net.org/fcg-interactive`.

2.2 Bidirectional Constructions

Constructions are, linguistically speaking, usage-patterns that combine morpho-syntactic and pragmato-semantic information. Constructions can be very concrete, for example in the case of morphological and lexical constructions, but can also be more abstract, for example in the case of argument structure constructions. Computationally speaking, constructions are feature structures which always have the same basic design, as exemplified in Figure 2. Constructions consist of a *contributing part*, left of the arrow, and a *conditional part*, right of the arrow. The conditional part consists of 1 or more units, and the contributing part of 0 or more. The units on the conditional part of a construction each consist of two parts, separated by a horizontal line. The upper part is called the *formulation lock* and the lower part the *comprehension lock*. When the construction is used in comprehension, the comprehension lock of the units on the conditional part will be matched with the units of the transient structure. Matching is a subset unification process, which checks whether the conditions stated in the locks are compatible with the transient structure. If matching succeeds, the rest of the construction, i.e. the formulation lock and the contributing part, will be merged into the construction. Merging is an other unification process, which, in addition to matching, adds features from the construction that do not yet occur in the transient structure to the transient structure. When the construction is used in formulation, it will be the formulation locks that are matched and the comprehension locks and contributing part that will be merged. The active locks, i.e. the comprehension locks in comprehension and the formulation locks in formulation, can thus be thought of as the *preconditions* of the construction, and the non-active locks and the contributing part as *postconditions*. The fact that the construction is structured into this lock system, facilitates an efficient implementation of bidirectional language processing.

An example construction is shown in Figure 2. The conditional part of this determinednoun-cxn consists of three units and the contributing part of one unit. Conventionally, (logic) variables are preceded by a question mark. In comprehension as well as in formulation, this construction adds a noun to an existing noun phrase unit, with the additional constraint that the noun needs

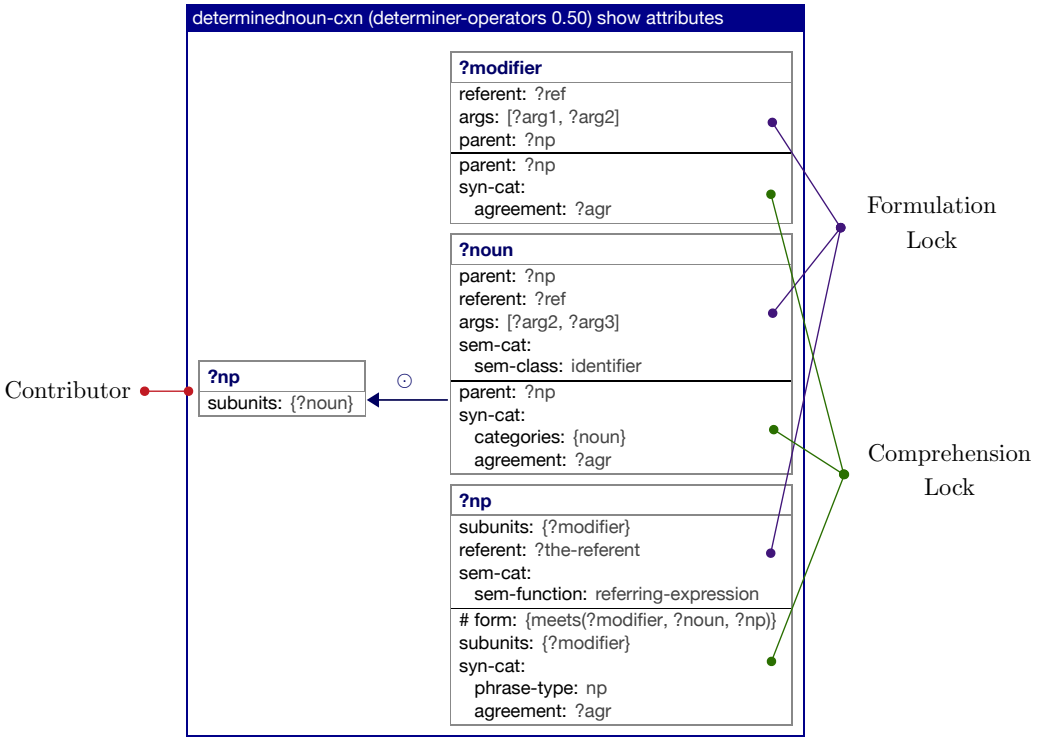

Figure 2: A determinednoun-cxn with labels highlighting its *comprehension lock*, *formulation lock* and *contributor*.

to be right adjacent to a modifier that is already part of the noun phrase.

2.3 Constructional Dependencies

For more complex grammars and inputs, the number of applied constructions, units and features can become quite large. Structures containing a thousand features over sixty units, contributed by a hundred constructions are no exception. These large structures are difficult to debug and interpret and it is highly non-trivial to tell why a construction could apply, i.e. which previous constructions contributed the necessary features that satisfy the locks of the construction. Moreover, from a linguistic point of view, the construction application process is often more informative than the final transient structure, as it reveals which constructions have applied with which inputs and outputs.

At this moment, FCG's standard visualisation system [8] visualises the construction application process as a tree structure, drawing one node for each construction application. Each node shows the transient structure before construction application, the construction itself, and the transient structure after construction application, as well as the bindings from matching. This standard visualisation is shown in Figure 4 for the input utterance *the cat*, as analysed by the Basic English Grammar [20]. The tree structure is linear here, as the comprehension process did not require any search. Each green box represents the application of a construction, and contains the transient structure after the application of that construction. The figure should be read from left to right, and shows how the transient structure is gradually build up. For space reasons, the transient structure before construction application, the construction itself, the matching bindings and the features inside the units are collapsed here, but they can be explored in full detail in the web demonstration supporting this paper.

Although the standard visualisation has proven its worth in grammar engineering for over a decade now, it does not provide insight in the complex puzzle of constructional dependencies. It does not transparently display which earlier constructions have provided the necessary features

that satisfy the lock of a later construction. An earlier effort at integrating the idea of construction networks into FCG focused mainly on their benefits in optimizing the search space, rather than visualising the constructional dependencies as such [22, 21]. The new visualisation that is proposed in our paper specifically tackles the problem of visualising linguistic pathways with a special focus on these dependencies.

3 Visualising Linguistic Pathways

The main asset of the new visualisation is that it gives insight into how a solution, in this case a certain feature structure that resulted from the construction application process, came into being. To explain the specifics of the new visualisation, we work again with the example utterance *the cat*. We comprehend the utterance using the Basic English Grammar [20]. The initial transient structure consists only of the input strings and word order information. Then the search process begins, and six constructions apply, gradually extending the transient structure. These constructions are the the-cxn, the cat-noun-morph-cxn, the cat-noun-lex-cxn, the determiner-operation-cxn, the determined-noun-cxn and the singular-cxn. The final transient structure is shown in Figure 3 and the standard visualisation of the application process of the six constructions in Figure 4.

The final transient structure contains three units: an NP unit and two lexical units as its children. The highlighted text in the figure shows certain variable equalities (e.g. referent) and important features such as agreement, whose value is the same in all three units. These features have been contributed by the six different constructions, and certain constructions could only apply after a combination of other construction had contributed the necessary features. How these constructions depend on each other to build the transient structure that we see here, is exactly what our tool visualises.

This section first explains the design of the visualisation tool (3.1) and then discusses how it has been integrated into Fluid construction Grammar (3.2).

3.1 Design

The graph in Figure 5 shows the constructional dependencies between the six constructions that were active in the analysis of the example utterance *the cat*. It consists of three main components:

1. *Constructions* are represented as rectangular, blue node clusters. In each node cluster, the name of the construction is shown at the top in white text on a blue background (e.g. determinednoun-cxn). The clusters are laid out from left to right, in their order of application. Constructions more to the left are responsible for earlier expansions of the transient structure.

2. *Units of constructions* make up the nodes of the graph and are shown as rectangular boxes within the constructions. These boxes contain the names of the units (e.g. ?definite-article or ?modifier). The unit names are variables that are bound to units in the transient structure during construction application. When building up the transient structure shown in Figure 3 for instance, ?definite-article, ?determiner and ?modifier will all be bound to the-179. ?cat-unit and ?noun will be bound to cat-45 and ?np will be bound to np-828. Units of constructions can have a white or a green background. White units were already present in the transient structure before the construction had applied, and were thus added by an earlier construction application. Green units are added by the construction during its application.

3. *Constructional dependencies* are represented by labelled edges between units. The direction of the edges follows a chronological order: the construction of the unit at the source applied before the construction of the unit at the target. Hence, the unit at the target (usually) depends on information from the unit at the source. The labels on the edges contain conditional information that was present in the target construction's formulation

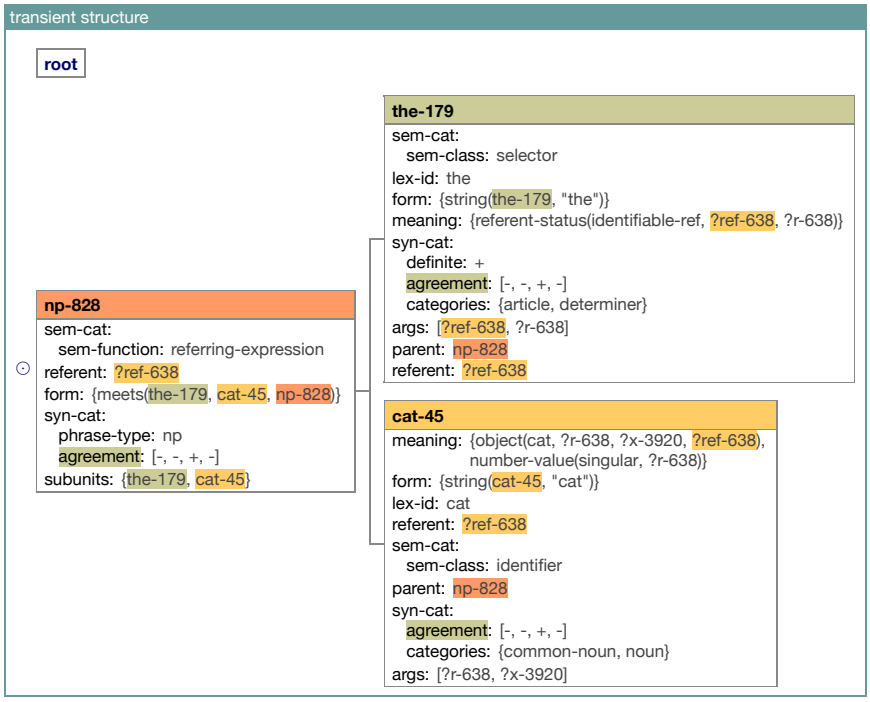

Figure 3: Resulting transient structure after comprehending *the cat*. Six different constructions have shaped this transient structure: the the-cxn, the cat-noun-morph-cxn, the cat-noun-lex-cxn, the determiner-operation-cxn, and the determined-noun-cxn.

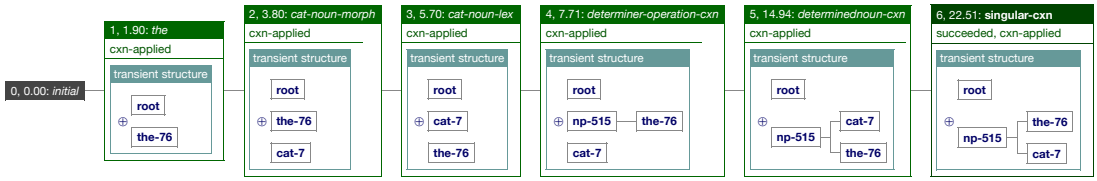

Figure 4: The construction application process for the utterance *a cat* using the Basic English Grammar [20], as visualised by FCG's standard visualisation library. The green boxes show how the application of the different constructions sequentially expand the transient structure.

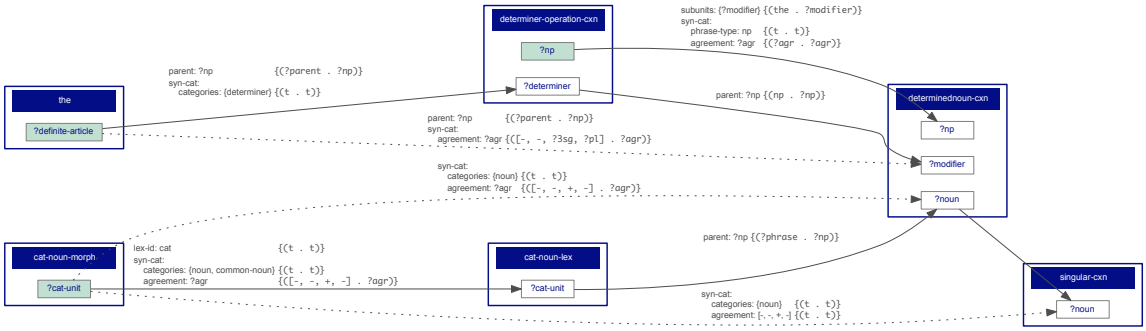

Figure 5: The construction application process for the utterance *a cat* using the Basic English Grammar [20], as visualised by the tool introduced in this paper. The dependencies between the constructions and their units are clearly indicated with arrows.

or comprehension lock, depending on the direction of processing. Every label consists of two columns:

(a) The first column contains all features that were contributed by the source construction and were used by the target construction's lock for matching.

(b) The second column is the binding list that resulted from the matching operation, with on the left the source feature value and on the right the target feature value to which it could be bound. If the binding looks like (t . t), it means that two atomic values were unified, e.g. np ⊔ np.

Figure 5 includes two types of edges. Solid arrows indicate dependencies between directly adjacent nodes and thus reflect the chronology of construction applications, while dotted arrows display longer-range dependencies. Let's focus on the determined-noun-cxn in the example, and more in particular on its ?noun unit. A solid arrow connects the ?cat-unit from the cat-noun-lex construction to this unit and the label on the arrow indicates that the latter gets its parent feature from there. The fact that the arrow is solid marks that the no other construction has affected this unit between the cat-noun-lex construction and the determined-noun-cxn. A dotted arrow connects the ?cat-unit from the ?cat-noun-morph construction to the ?noun unit of the determined-noun-cxn and the label indicates that it gets its categories and agreement features from there. The dottedness of the arrow shows that an other construction has affected the unit between the cat-noun-morph construction and the determined-noun-cxn, namely the cat-noun-lex construction.

3.2 Integration into FCG

The visualisation tool has been integrated into the FCG web interface and monitoring system [8, 9], and will be included in FCG's next release. The graph with constructional dependencies as shown in Figure 5 is automatically added to the visualisation of processing results when the following configuration option is set:

```
(set-configuration (visualization-configuration cxn-inventory)
                   :constructional-dependencies t)
```

The graph can be customised to the user's preferences by setting a few additional configuration options in the visualisation configuration of the construction inventory.

- `(:labeled-paths nil/no-bindings/t)`

  This option defines what information is shown on the arrows that indicate the dependencies. nil leaves the arrows empty, t shows the matched features and their bindings (as shown in Figure 5) and no-bindings shows the matched features without bindings.

- `(:colored-paths nil/t)`

  When this option is set to t, each directed path in the graph is drawn in a different colour. This means that the units in the construction that have been bound to the same unit in the transient structure and the arrows connecting these units will have the same colour.

- `(:trace-units '(list-of-unit-names))`

  Having the matching features and their bindings on all arrows can make larger graphs quite crowded. Therefore, it is sometimes better to only include the features and bindings of the specific paths that the grammar engineer is interested in. These paths can be specified with using the :trace-units option.

For space reasons, we could not include figures showing these different options into the printed version of this paper, but they are all shown next to each other in the web demonstration supporting this paper.

4 Demonstration

We will now demonstrate the visualisation tool with a somewhat larger example, in which the linguistic pathway grows into a more intricate web of unit dependencies. We will analyse the comprehension process of a single sentence, but the web demonstration that accompanies this paper includes many more (and longer) examples, also for production. Because the graphs become very detailed when a larger number of constructions are used in the analysis, we chose to leave out the labels (i.e. the matching features and bindings) on the edges that link the unit nodes. The complete constructional dependency graphs are included in the on-line web demo.

The example sentence, *the cat will jump*, contains a noun phrase, a modal auxiliary and an intransitive main verb. When analysed with the current version of the Basic English grammar ([20] and interactively consultable at www.fcg-net.org/fcg-interactive), 17 constructions are needed to analyse the sentence in comprehension. The visualisation of the constructional dependencies is shown in Figure 6.

The constructional dependencies graph contains 17 blue node clusters, one for each construction that has been applied. On the left side of the graph, we can see that there are four entry points, i.e. node clusters with no incoming edges. The comprehension locks of these four constructions, jump-morph, will-morph, the and determiner-operation-cxn, only require strings that are found in the input utterance and do not need any features added by previous construction applications. They create one lexical unit each, which is shown on a green background. After the the construction has applied and created a ?definite-article unit, the determiner-operation-cxn can match its ?determiner unit on this unit and create a new ?np unit. The cat-noun-lex and jump constructions can respectively match on the units created by the jump-morph and cat-noun-morph constructions, adding information to these units, but not creating any new ones.

Then, we can see that the lexical units for the modal auxiliary and the lexical main verb are both affected by the marked-modality-cxn, which creates a new ?vp unit. This ?vp unit is then modified by the vp-cxn, non-perfect-cxn and the non-progressive-cxn. The determinednoun-cxn adds information into lexical units for the noun and the definite article, as well as to the ?np unit. The lexical unit for the noun is further affected by the singular-cxn.

After that, the subject and verb come together. The subject-verb-cxn matches on the ?np and ?vp units, and creates a new ?clause unit. These three units are then further extended by the declarative-main-clause-verb-cxn and the intrasitive-cxn.

From this short description, it becomes already clear that constructional dependencies are a complex matter, and yet, we have only described the information conveyed by the boxes and the solid arrows in the graph. The visualisation tool allows the grammar engineer and user to quickly grasp how the grammar has analysed an input utterance and spot possible bugs or inconsistencies, which was much more difficult using FCG's standard visualisation. The graphs are designed to be studied on computer screens, where the users can zoom in and out on parts of the network. Therefore, they tend to get too large to be printed on paper quite soon.

5 Beyond Language Processing

Although the visualisation tool introduced in this paper was specifically designed for visualising linguistic pathways yielded by computational construction grammar analyses, it can easily be applied to other domains as well. In order to demonstrate the generality of the tool, we will now show an example of its application to a planning problem. As the basic building blocks of FCG, in particular the exploration of a search space, are so general, the planning problem could be implemented in the same formalism.

The goal in our planning problem is having pancakes. The initial state is a kitchen's basic equipment (a stove, pans, bowls, a whisk, cutlery, ...) and basic ingredients (eggs, butter, milk, flour, tabasco, ham, ...). The operators are actions such as taking eggs, collecting the ingredients that are needed for pancake, and putting butter in a pan and putting the pan on the stove. The planning problem consists in finding a series of subsequent actions that form a path from the initial state to a state in which there are delicious pancakes.

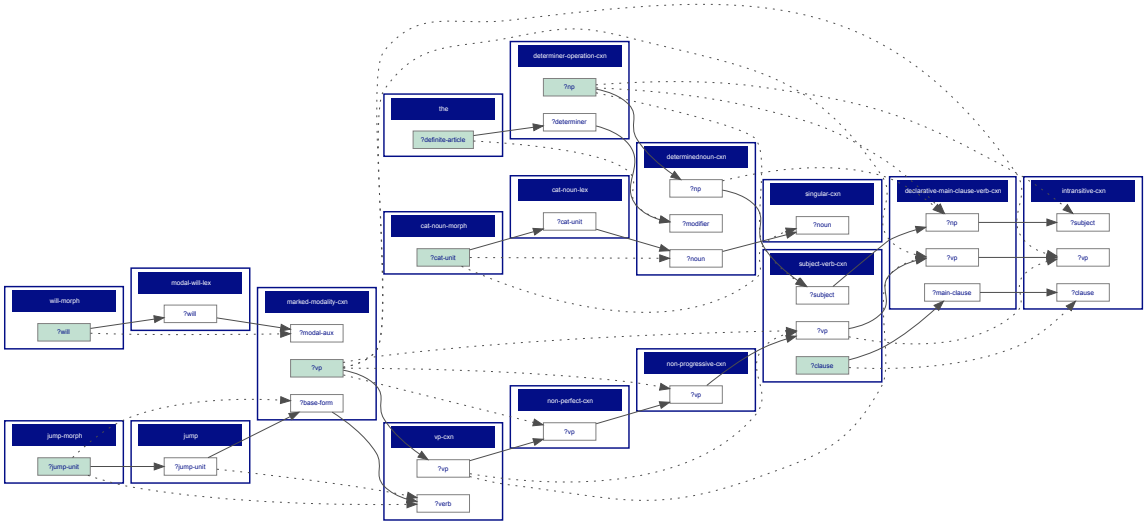

Figure 6: The constructional dependencies graph for the utterance *the cat will jump* using the Basic English Grammar [20].

The graph containing the constructional dependencies for the pancake problem is shown in Figure 7. We can see that in total, 11 actions have been performed to make the pancakes. 7 actions were take-... actions that could be independently performed based on the initial state. Then, a collect-pancakes-ingredients action matched on the ?flour, ?eggs and ?milk units, creating a new ?pancake-ingredients unit. The make-pancake-dough action then matched on the ?pancake-ingredients, ?whisk and ?bowl units and created a new ?pancake-dough unit. The prepare-pan action matches on the ?pan, ?butter and ?stove units and creates a new ?prepared-pan unit. Finally, the make-pancakes action matches on the ?prepared-pan unit and the pancake-dough unit and creates a new pancakes unit. The arrows are labelled with the features on which the actions match, and the bindings of these features. When looking at the make-pancakes action for example, we can see that it matches on a ?pancake-dough unit that does not have baked among its properties (marked in red in the figure). The make-pancakes construction will add baked to the properties of this unit, and indeed, pancake dough can only be baked a single time.

This visualisation is of course only one possibility for visualising a planning process. It does not replace other, complementary visualisations, such as the explored search space. The worth of this visualisation is its focus on the dependencies between the actions, and on which previous actions have facilitated the performance of later actions. It is particularly useful when actions are complex and depend on many different factors affected by other actions.

6 Future Work

At this moment, we use the tool to visualise the dependencies between the construction that were active in the comprehension or formulation process of a single utterance. For the future, we plan to comprehend a corpus of sentences and gradually build up a large graph containing the dependencies between all grammatical constructions in the grammar. The edges in the graph can then be coded with frequency information about (parts of) linguistic pathways and capture systematic dependencies between groups of constructions. This would allow us to automatically build and visualise a usage-based constructional dependency graph that evolves over time, for a complete grammar.

7 Conclusion

In this paper, we have introduced a novel approach to visualising the outcome of constructional language processing. The linguistic pathways are visualised as graphs, featuring the applied constructions, why they could apply, with which bindings, and what information they have contributed. By revealing the dependencies between the applied constructions, these graphs are very helpful when developing and debugging grammars, while also enhancing the human readability of construction grammar analyses. The tool has been concretely implemented in Fluid Construction Grammar, but the proposed visualisations should be easily transferable to other construction grammar implementations. In order to highlight that the applicability of the tool is not limited to language processing problems, we have also demonstrated how the tool can be used to visualise planning problems.

Acknowledgements

We are particularly indebted to Luc Steels, Remi van Trijp and Yannick Jadoul for their support and valuable feedback during the development of the tool.

References

[1] B. Bergen and N. Chang. Embodied Construction Grammar in simulation-based language understanding. In *Construction Grammar(s): Cognitive and Cross-Language Dimensions.* John Benjamins, 2005.

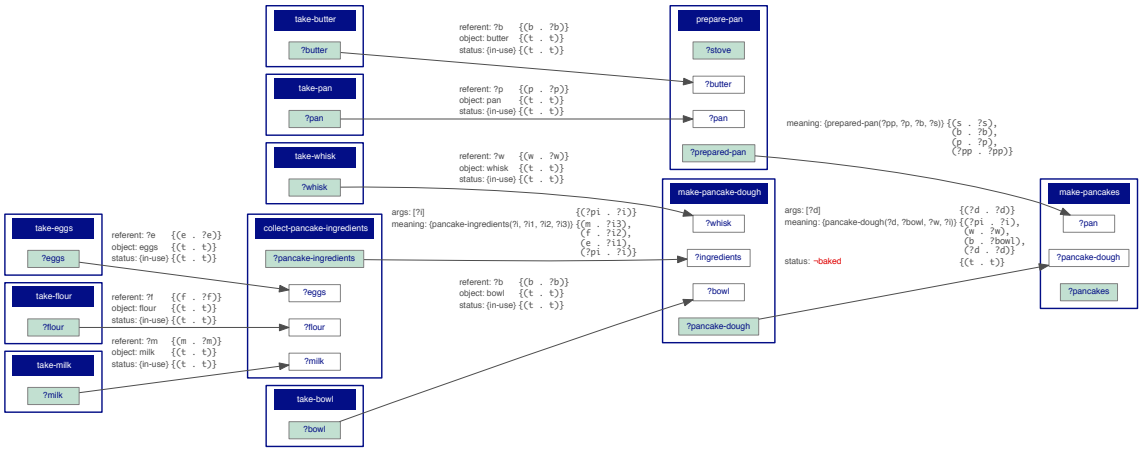

Figure 7: Using the new visualisation tool for a (hierarchical) planning problem.

[2] N. Chang, J. Feldman, R. Porzel, and K. Sanders. Scaling cognitive linguistics: Formalisms for language understanding. In *Proc. 1st International Workshop on Scalable Natural Language Understanding*, 2002.

[3] W. Croft. *Radical Construction Grammar: Syntactic theory in typological perspective*. Oxford University Press, Oxford, 2001.

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

[5] A.E. Goldberg. *Constructions: A construction grammar approach to argument structure*. University of Chicago Press, Chicago, 1995.

[6] A.E. Goldberg. *Constructions at work: The nature of generalization in language*. Oxford University Press, Oxford, 2006.

[7] P. Kay and C.J. Fillmore. Grammatical constructions and linguistic generalizations: the What's X doing Y? construction. *Language*, 75(1):1–33, 1999.

[8] M. Loetzsch. Tools for grammar engineering. In L. Steels, editor, *Computational Issues in Fluid Construction Grammar*. Springer Verlag, Berlin, 2012.

[9] M. Loetzsch, J. Bleys, and P. Wellens. Understanding the dynamics of complex lisp programs. In *Proceedings of the 2nd European Lisp Symposium*, pages 59–69, Milano, Italy, May 2009.

[10] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(2):258–282, 1982.

[11] A. Newell, J.C. Shaw, and H.A. Simon. Empirical explorations of the logic theory machine: A case study in heuristic. In *Papers Presented at the February 26-28, 1957, Western Joint Computer Conference: Techniques for Reliability*, IRE-AIEE-ACM '57 (Western), pages 218–230, New York, NY, USA, 1957. ACM.

[12] N. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill Pub. Co., 1971.

[13] J.A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[14] J.A. Robinson. Computational logic: The unification computation. *Machine intelligence*, 6:63–72, 1971.

[15] L. Steels, editor. *Design Patterns in Fluid Construction Grammar*. John Benjamins, Amsterdam, 2011.

[16] L. Steels. Basics of Fluid Construction Grammar. *Constructions and Frames*, 9(2), 2017.

[17] L. Steels, J. De Beule, and P. Wellens. Fluid Construction Grammar on real robots. *Language Grounding in Robots*, pages 195–213, 2012.

[18] L. Steels and P. Van Eecke. Insight grammar learning using pro- and anti-unification. Forthcoming.

[19] L. Steels and E. Szathmáry. Fluid Construction Grammar as a biological system. *Linguistics Vanguard*, 12(1), 2016.

[20] R. van Trijp. A computational construction grammar for English. In *The AAAI 2017 Spring Symposium on Computational Construction Grammar and Natural Language Understanding Technical Report*, number SS-17-02, pages 266–273, Stanford, 2017. Association for the Advancement of Artificial Intelligence.

[21] P. Wellens. Organizing constructions in networks. In L. Steels, editor, *Design Patterns in Fluid Construction Grammar*, pages 181–201. John Benjamins Publishing Company, 2011.

[22] P. Wellens and J. De Beule. Priming through constructional dependencies: a case study in fluid constructions grammar. In *Proceedings of the 8th International Conference on the Evolution of Language (EVOLANG 8)*, pages 344–351, Singapore, 2010. World Scientific.