# Meta-layer Problem Solving for Computational Construction Grammar

**Paul Van Eecke**
Sony Computer Science Laboratory Paris
6, rue Amyot
75005 Paris, France
vaneecke@csl.sony.fr

**Katrien Beuls**
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
katrien@ai.vub.ac.be

**Abstract**

Bearing the word "fluid" in its name, Fluid Construction Grammar (FCG) is known for its open-ended nature when it comes to formalising linguistic knowledge in the form of constructions. Yet, it is also flexible with respect to the processing of input that cannot be handled in the standard way. This paper presents a meta-layer architecture that is fully integrated into the FCG language processing framework, as well as a number of powerful and general operators that can be used within this architecture for on-the-fly problem solving and learning of constructions.

## Introduction

Although the bulk of language processing happens routinely while we speak and listen, we sometimes need to deal with unexpected input, such as new words popping up, known words being used in different situations, or new grammatical structures being introduced by combining previously existing structures in a novel way. In order to accommodate the innovative and evolving nature of language, any computational platform that aims to process everyday language needs powerful mechanisms for handling these phenomena.

In this paper, we present a general meta-level architecture that facilitates on-the-fly problem solving and learning of constructions. The basic architecture consists of a *routine-layer* for standard processing and a *meta-layer* for problem solving and learning. Furthermore, it provides mechanisms that detect possible impasses during routine processing (*diagnostics*), problem solving stategies that find solutions to these impasses (*repairs*) and learning strategies that consolidate these solutions for later reuse in standard processing (*consolidation strategies*). The meta-level architecture is fully operational and integrated in the core of the Fluid Construction Grammar framework (Steels 2011; 2016).

The paper is structured as follows. First, we will present how constructional language processing can be approached as a search problem and how this is reflected in Fluid Construction Grammar's routine layer. Then, we will describe in more detail how the meta-layer architecture is implemented. Third, a few didactic examples show the meta-layer at work and introduce a number of general and powerful diagnostics, repairs and consolidation strategies. A review of related literature concludes the paper.

A web demonstration containing more details about the actual implementation of both the meta-layer and the examples discussed in this paper is available at `http://www.fcg-net.org/demos/meta-layer`. We will refer to section X in the web demonstration as 'WD-X' in this paper.



## Language Processing as a Search Problem

Fluid Construction Grammar approaches language processing as a *search problem*, a class of problems that has been extensively studied in AI research (see for example Newell and Simon 1972). The search problem for constructional language processing can be analysed into the following three components, as enumerated by Steels and Van Eecke (2016):

1. A problem state (FCG: *transient structure*), which represents the current state of knowledge that a speaker or hearer has about an utterance that he is constructing (in production) or analysing (in comprehension).
2. A goal that characterises a solution state (FCG: *goal test*), e.g. criteria that define a succesful interpretation of an utterance.
3. Operators (FCG: *constructions*) that apply to an existing problem state and create a new problem state that is closer to a solution state.

The search process itself consists of finding a sequence of constructions that expand the transient structure in subsequent steps from the initial state to a solution state. Transient structures are represented as *feature structures*, containing *units*, *features* and *values*, which can themselves be feature-value pairs. The *initial transient structure* consists of an input buffer that either contains a meaning representation that the speaker wants to express (in *production*), or a segmented utterance that the listener is trying to comprehend (in *comprehension*).

Just like transient structures, constructions take the form of feature structures, although the latter are a little more structured. Constructions consist of a *comprehension lock*, a *production lock* and a *contributor* (see Figure 1). When a constuction tries to apply during comprehension, the features of the comprehension lock are matched with the transient structure. If the match succeeds, the features of both

the production lock and the contributor are merged into the transient structure. In production, the same process happens, but here the features of the production lock are matched and those from the comprehension lock merged.

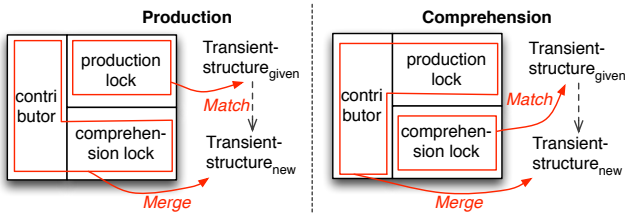


Figure 1: Construction application happens by first matching either the production lock (in production - left) or the comprehension lock (in comprehension - right) onto the given transient structure. When matching succeeds, the features from the other lock and the contributor are merged into the new transient structure.

Through subsequent application, a number of constructions expand the initial transient structure into a final transient structure. Typically, lexical or morphological constructions will first try to carve out pieces of the 'meaning' or 'form' features of the input. Then, more abstract grammatical constructions combine these units into larger networks. After each construction application, goal tests are run to check whether a transient structure qualifies as a potential solution. If it does, processing stops and the comprehended meaning representation (in comprehension) or produced utterance (in production) are extracted from this final transient structure. If it does not, processing continues and a next construction from the construction inventory is tried. Common goal tests are: no more applicable constructions (queue exhausted), the resulting meaning network is fully integrated (one chunk) or all strings from the input have been processed.

Just like for any search problem, the order in which constructions are applied has a major influence on the efficiency of the search. This optimization problem falls however outside the scope of this paper and for now, we will assume that constructions apply in a random order and that we navigate through the search space in a depth-first manner.

## Routine Language Processing

Let us now have a look at a concrete example of routine language processing in FCG, using the mechanisms described above. Say, we want to comprehend the utterance "the mouse". When FCG is called, it creates an initial transient structure based on the input utterance. The initial transient structure contains two strings ("the" and "mouse"), as well as a "meets" constraint, indicating that "the" immediately precedes "mouse" (see Figure 2).

Then, the search process starts and one by one, the comprehension locks of the different constructions (cxns for short) of the construction inventory are matched against this initial transient structure. At some point, the "mouse-cxn" (see Figure 3) is tried. Its comprehension lock contains a sin-

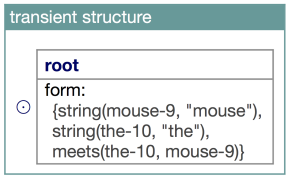


Figure 2: The initial transient structure for the utterance "the mouse".

gle feature, that can indeed be matched[1] with the initial transient structure. The construction can apply and all features from the production lock and the contributor are merged into the transient structure, as shown in Figure 4.

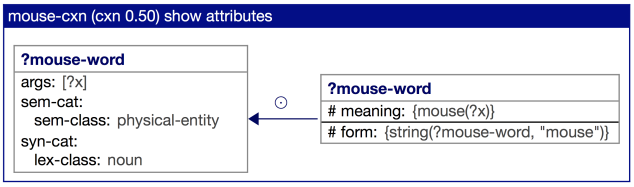


Figure 3: The construction for "mouse", with its contributor (left), production lock (right above) and comprehension lock (right below).

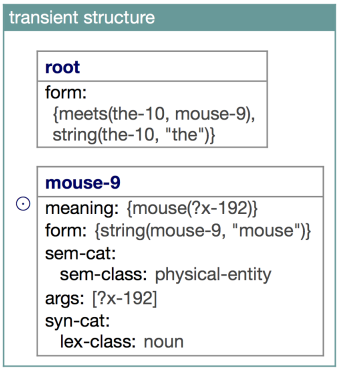


Figure 4: The transient structure created by applying the mouse-cxn in Figure 3 to the initial transient structure in Figure 2.

The construction application process continues and respectively the "the-cxn" and the "noun-phrase-cxn" apply. Then, no more constructions can apply, which in this case defines a succesful goal-test result. The final transient structure is returned and its meaning predicates are extracted and visualised as a semantic network. The complete construction application process and the extracted meaning network are shown in Figures 5 and 6 below.

This example is shown in full detail in Section WD-1 of the web demonstration that supports this paper.

[1] Matching is a form of unification and symbols preceded with a ? are variables.

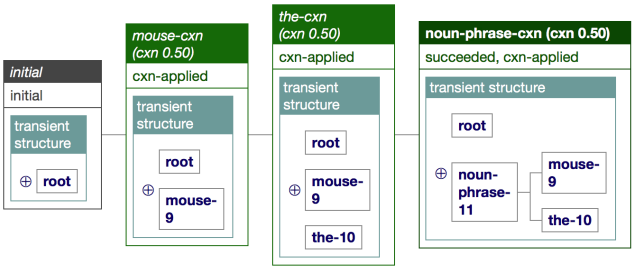


Figure 5: The construction application process for the utterance "the mouse".

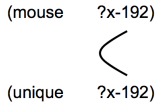


Figure 6: The resulting semantic network extracted from the final transient structure.

## Meta-layer Language Processing

The routine processing layer described in the previous section is well suited for handling input that is entirely covered by the grammar. In real-world use however, static models will never achieve full coverage because of the dynamic and evolving nature of language. Language users are creative and innovative, the accoustical conditions may be suboptimal, or language users might just make downright errors. In these cases, FCG's routine layer will still process the input as far as possible, which may be good enough for some purposes, but additional mechanisms for problem solving and learning of innovations are often desirable.

For this reason, we have integrated a meta-layer into the FCG framework. The meta-layer architecture is based on *diagnostics*, *repairs*, and *consolidation strategies*. It provides the necessary mechanisms to:

- detect problems during comprehension or production;
- temporarily stop the expansion of the transient structure that is being built up;
- find a solution to the problem;
- resume processing at the present or an earlier stage;
- consolidate the solution for later reuse during routine processing.

### Diagnostics

Diagnostics are tests that inspect transient structures for abnormalities and potential problems. They are defined on grammar-level and are run after each construction application. If an abnormality is found, a diagnostic will create a problem of a certain type (e.g. unknown word, agreement mismatch) and trigger a jump to the meta-layer. Diagnostics typically take the current transient structure as input, but have access to all previous steps in the search process, as well as to the construction inventory and to all constructions that have been applied so far.

### Repairs

Repairs are strategies to solve problems. They are specialised towards the specific (classes of) problems that diagnostics can trigger. Just like diagnostics, repairs have complete access to the current and previous transient structures, the previously applied constructions, and the complete construction inventory. Based on the diagnosed problem, they try to find a solution in the form of a *fix*. The type of the fix and the way that the fix repairs the problem is open-ended. Here, we will focus on a specific kind of fix, namely fixes that are themselves constructions. Repairing the problem consists then of making the appropriate fix-cxn and to apply this fix-cxn to the current transient structure. After this, routine processing takes over again.

### Consolidation

Diagnostics and repairs can handle language use that is not covered by routine processing. Yet, it is also desirable to learn from the problem solving process, such that similar input can in the future be handled by routine processing and will not lead to the creation of a problem anymore. In the case that fixes are themselves constructions, the transfer of the solution from the meta-layer to the routine layer is quite straightforward. When a branch of the search tree leads to a solution, the constructions that were created on the fly by a repair can simply be added to the construction-inventory.

In many cases, it is necessary to find a good balance between the specificity and generality of a fix-cxn. If it is too specific, it might only apply to exactly the same input, whereas if it is too general, it might apply to cases it is not appropriate for. In the next section, we will go into more detail about techniques for finding this balance, especially the *anti-unification* and *pro-unification* algorithms.

Figure 7 schematically shows how the meta-layer is implemented in FCG. After node t+2, a problem is diagnosed and the routine processing pipeline is briefly interrupted by the repair process taking place. Then, routine processing smoothly resumes and the fix is consolidated after a solution is found.

## Diagnostics and Repairs at Work

The power of a meta-layer architecture lies of course in the specific diagnostics, repairs and consolidation strategies that are used. The meta-layer that we have integrated into FCG is completely open-ended in this respect and provides the full power of a programming language to implement any functionality that is needed. In this section, we present a number of diagnostics, repairs and consolidation strategies that are now part of the FCG core. The first one shows how new lexical constructions can be created on the fly, the second one how new phrasal constructions can be induced from co-referential relations, and the third one presents powerful algorithms for making constructions more general or more specific.

### Creating New Lexical Constructions

Sooner or later, any grammar that is used in a real-world setting will need to deal with missing lexical constructions. In

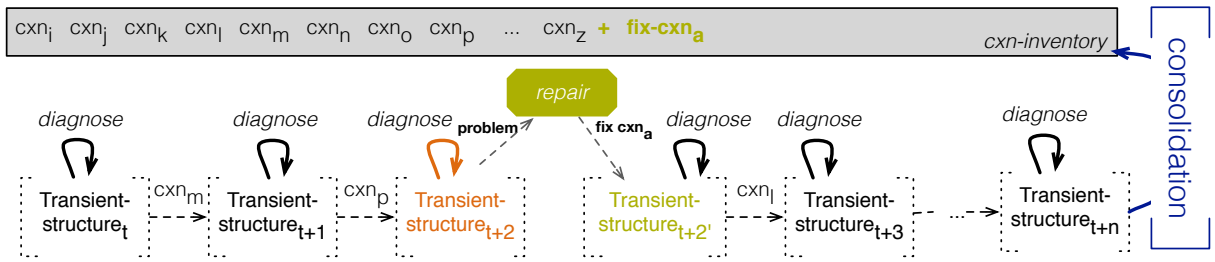


Figure 7: Routine language processing is briefly interrupted when a problem is diagnosed at time step 2 in the processing pipeline. After the repair has suggested a potential fix for the problem, routine processing is resumed.

comprehension, this problem occurs when the input utterance contains a string that is not covered by any construction. In production, it occurs when no construction covers a concept that needs to be expressed.

**Diagnostic** Diagnosing a missing lexical construction is quite straightforward. In comprehension, a problem of the type `unknown-words` is created when no more constructions can apply and the root unit (initial input buffer) still contains one or more strings. In production, a problem of the type `unknown-meaning-predicates` is created when no more constructions can apply, and the root unit still contains one or more meaning predicates.

**Repair** A new lexical construction is created based on a grammar-specific construction template. The template contains the features that are common to the different lexical constructions in the grammar. In comprehension, the string can be filled in and a new meaning predicate is invented. In production, the meaning predicate is filled in and a new string is invented[2]. The values of the other features are either inferred by external resources (such as morphology-based taggers in the case of the syntactic category) or left underspecified as free variables.

**Consolidation** The freshly created lexical constructions can be added as such to the construction inventory. Typically, they will be added with a very low initial score that will increase if the new construction is used multiple times in succesful interactions and gets therefore more entrenched.

**Example** Let us have a look at an example for production, which can be explored in full detail as WD-2 in the web demonstration. Imagine that an agent needs to express the conceptualised meaning shown in Figure 8. The agent's grammar contains the required lexical constructions for (unique ?x), (linguist ?x) and (deep-affection ?x ?y), but contains no construction covering the predicate (book ?x). In other words, the agent knows the concept of a book, needs to communicate with another agent about a book, but has no word for it.

[2]In production, new strings can either be invented randomly, or based on existing lexical constructions, potentially stretching their semantics. In comprehension, meaning predicates can also be invented randomly, or inferred from the world model and mapped to an ontology.

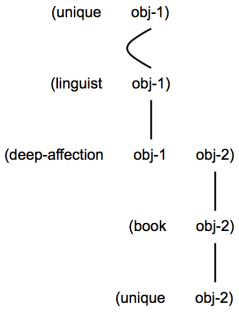


Figure 8: The conceptualised meaning that needs to be expressed. The concept (book ?x) is unfortunately not covered by the agent's grammar.

When routine processing starts, the "likes-cxn", "verb-phrase-cxn", "linguist-cxn" "the-cxn" (2 times) and "noun-phrase-cxn" apply. The unknown-meaning-predicates diagnostic is run after each construction application. After the application of the "noun-phrase-cxn", no more constructions are applicable, but the root still contains a meaning predicate, so the diagnostic notifies a problem, as shown by the orange node in Figure 9. Then, FCG jumps to its meta-layer and a creates on the fly a new lexical construction, as shown by the yellow node in Figure 9.

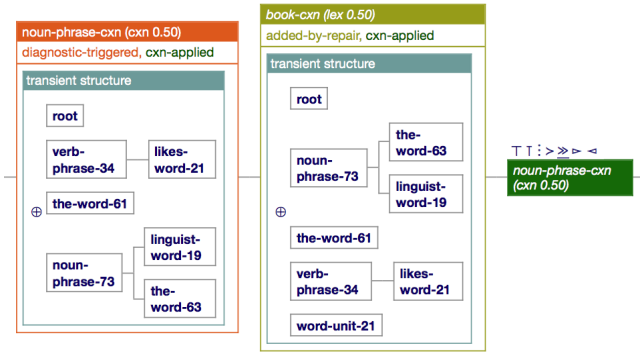


Figure 9: Excerpt from the construction application process triggering a diagnostic (orange node) and repair (yellow node) for adding a new lexical construction.

The freshly created lexical construction is shown in more detail in Figure 10. We can see that the string that was invented for this construction is "ropapa" and that the lex-class and sem-class are left underspecified (and will therefore match any symbol).

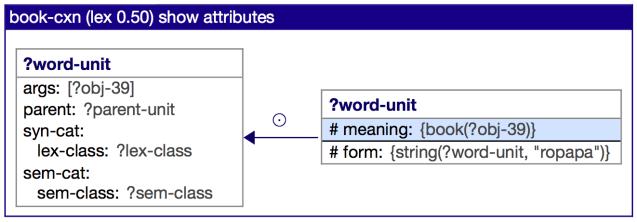


Figure 10: Lexical construction created for the concept (book ?x). The string that was assigned to this construction happens to be "ropapa" and the lex-class and sem-class are left underspecified.

Once the book-cxn has been applied, routine processing can resume, as indicated by the green node for "noun-phrase-cxn" in Figure 9. Finally, After the "noun-phrase-cxn" and the "transitive-clause-cxn" have applied, we end up in a solution state. From this final transient structure, the utterance "the linguist likes the ropapa" can be extracted. It will now be left to the cognitive capacities of the other agent to infer the meaning of ropapa from the situation and to eventually create a new lexical construction himself. This way, the word "ropapa" with the meaning of (book ?x) might spread in the population of agents.

## Inducing New Phrasal Constructions

An important function of grammar is to narrow down referential ambiguity by encoding co-reference relations explicitly. The formal means by which these relations are encoded can vary, but many languages employ word order and/or markers for this purpose. In this example, we will show how meta-layer diagnostics and repairs can be used to learn phrasal constructions that integrate new units into existing phrases.

**Diagnostic** Missing phrasal constructions are diagnosed when (i) no more constructions are applicable, (ii) all input strings have been covered by lexical constructions, (iii) the semantic network extracted from the final transient structure is not fully connected (i.e. it contains of more than one chunk), and (iv) no existing constructions can easily be generalised to apply to the transient structure (see next section).

**Repair** A new phrasal construction is created. This construction makes the referents of two units equal and captures the observed word order.

**Consolidation** The phrasal-cxn is too general to be added as such to the cxn-inventory, as it would apply in cases for which it is not appriopriate. However, the pro-unfication technique that will be explained in the next section can be used to constrain the new phrasal construction towards the observed case before adding it to the cxn-inventory.

**Example** In the following example (see WD-3), the utterance that needs to be comprehended is "the green mouse". The grammar contains lexical constructions for "the" "green" and "mouse", as well as a "noun-phrase-cxn" that combines an article and a noun into a noun phrase. The grammar does not contain any constructions that can integrate an adjective into a noun phrase.

Routine processing applies the three lexical constructions and the noun-phrase-cxn. At this moment in processing, the meaning network in the present transient structure is unconnected (as shown in Figure 11) and the diagnostic for missing phrasal constructions creates a new problem, triggering a jump to the meta-layer.


Figure 11: Unconnected semantic network.

On meta-layer level, the repair makes a new phrasal construction as shown in Figure 12. The construction specifies that the lexical unit, in this case the adjective "green", should have the same referent as the phrase it will be integrated in, in this case the noun phrase. This is ensured by making the value of the 'args' feature in both units equal (args: [?ref]). Concerning the form, the construction specifies that the lexical unit is left-adjacent to the phrasal head within the scope of the phrase.

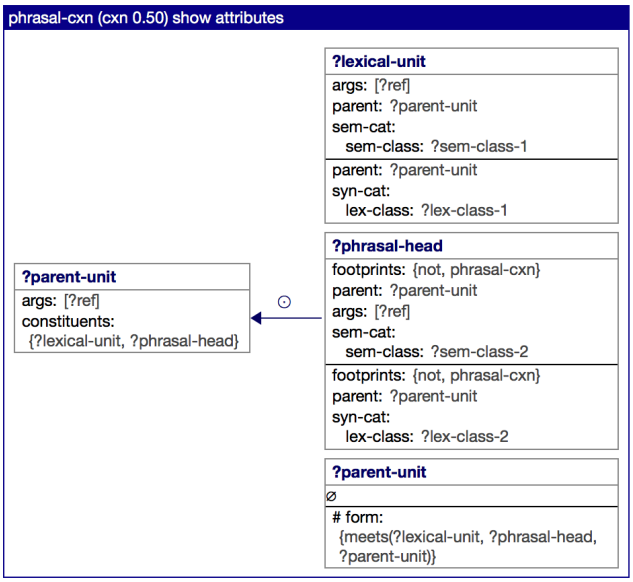


Figure 12: Phrasal construction that was created by the repair to integrate "green" into the noun phrase.

By applying the new phrasal construction to the transient structure, the unit for "green" is integrated into the noun phrase. The problem state now qualifies as a solution, and the resulting meaning representation is a single, fully connected network. The problem states in which the diagnostic

triggered and in which the repair-cxn was created and applied are shown in Figure 13 below.

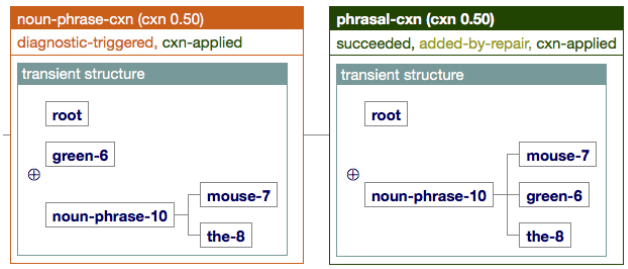


Figure 13: The nodes (problem states) in which the missing phrasal cxn diagnostic triggered (orange) en in which the phrasal-cxn applied (dark green).

## Generalising and Specialising Constructions using Anti- and Pro-unfication

Innovation in language often relies on novel constructions that share most of their properties with already existing constructions. These innovations pose a challange to computational systems, as the small number of features that differentiate the novel constructions from existing ones block their application completely. This is for example the case for coercions (different syntactic or semantic category), the emergence of new word orders (different word order features), or the raise and decline of agreement systems (different equality constraints). In order to process these phenomena, it is useful to be able to temporarily relax the conflicting features of a construction, while still matching and merging the bulk of the features when the construction applies. At the same time, the features and values that are more specific in the novel construction than in the general one should be learned. This avoids storing constuctions that are too general and therefore apply too widely.

(Steels and Van Eecke 2016) present two general and powerful operators that allow to flexibly match constructions and learn from their application. The first operator, called *anti-unification*, finds the least general generalisation of a construction that matches a given transient structure. In other terms, the anti-unification of a construction and a transient structure always returns a new construction that matches that transient structure. The features of the original construction that blocked the matching process are relaxed through generalization. The algorithm also returns a cost, indicating the distance between the original construction and the anti-unified construction. When the original construction already matches the transient structure, the same construction is returned with cost 0. When it does not match, a matching, generalised construction is returned with a cost $>$ 0, depending on the number, depth, and kind of features that needed to be relaxed.

While anti-unification generalises a construction to match a transient structure, the second operator, called *pro-unification* specialises a construction towards a transient structure. There are many options in the algorithm, but one of the basic functions is to bind different variables in the construction that are bound to the same values in the transient stucture to each other. Pro-unification is used immediately after anti-unification before a new construction is stored.

Given a new observation (transient structure) and a construction, anti-unification returns a generalised construction that matches the observation, while pro-unification specialises the generalised construction towards the observation. The pro-unified construction strikes a good generality-specificity balance, such that it can be added to the construction inventory and become part of the grammar. The generality-specificity and matching relations between constructions and transient structures involved in anti- and pro-unification are schematically sketched in Figure 14.

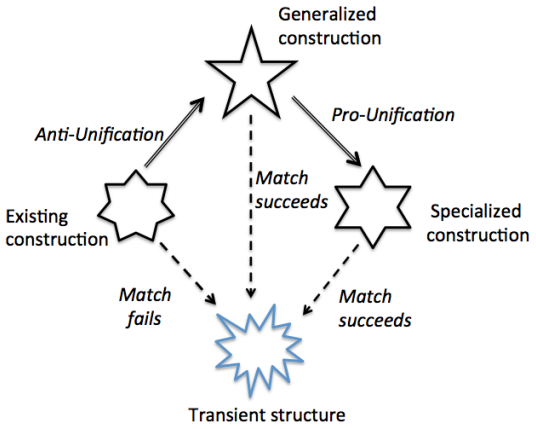


Figure 14: A schematic representation of anti-unification and pro-unification of a construction with a transient structure. Figure adapted from (Steels and Van Eecke 2016).

**Diagnostic** A problem of the type `matching-conflict` is created when no more constructions can apply and the meaning network extracted from the final transient structure is not fully connected, i.e. consists of more than one chunk.

**Repair** The different grammatical constructions of the grammar are anti-unified with the final transient structure and the cost is recorded. Then, the anti-unified construction with the lowest cost is applied to the transient structure and processing in the routine layer can continue. If no anti-unified construction can be found with a cost under a grammar-specific threshold, the repair signals that it cannot be used for this problem.

**Consolidation** The anti-unified construction is pro-unified with the resulting transient structure. The pro-unified construction is then added to the construction inventory.

**Example** We will now show an example of how anti-unification and pro-unification are integrated in the meta-layer framework (see WD-4). The example shows how the word order constraints in a noun phrase can be relaxed to

process a new observation and how the observed word order can be captured in a new construction.

For the sake of clarity, let's assume that the grammar of a French-learning agent consists only of lexical constructions and a noun phrase construction that groups a determiner, an adjective and a noun (in that order) into a noun phrase. This means that the agent can comprehend and produce utterances such as "le formidable dîner" ('the splendid dinner'). Now, the agent observes an utterance "le dîner formidable" ('the dinner splendid') in which the adjective is placed after the noun. This word order is also correct in French, so we would like the agent to learn a new construction covering noun phrases with adjectives in postposition. Let us first have a look at the transient structure after the application of the lexical constructions, as shown in Figure 15. Units of which the features are not relevant to this example are collapsed in the image and coreference relations are indicated in color.

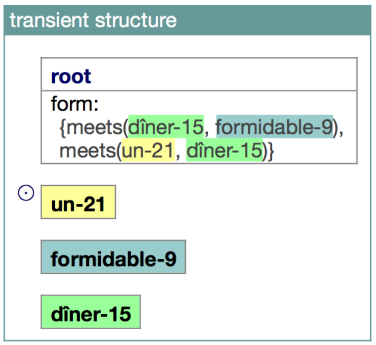


Figure 15: The transient structure after applying the lexical constructions to "un dîner formidable".

We can see that the observed word order of the utterance is present in the transient structure in the form of two 'meets' constraints, indicating that 'un' immediately precedes 'dîner' and 'dîner' immediately precedes 'formidable'. When we have a look at the agent's noun-phrase-cxn, which is shown in Figure 16, we can see that the comprehension lock stipulates that the article should meet the adjective and that the adjective should meet the noun. The construction does not match the transient structure and can therefore not apply.

At this point, a problem of the type `matching-conflict` is created by the diagnostic. FCG jumps to its meta-layer and the different constructions of the inventory are anti-unified with the transient structure. The np-cxn has the lowest anti-unfication cost. The resulting anti-unified construction is shown in Figure 17. We can see that the conflicts in the 'meets' constraints are solved through generalisation. The meets constraints now contain unbound variables (?adj-468 and ?noun-412 are not linked to units anymore). The anti-unified np-cxn can now apply to the transient structure and lead to a succesful result.

The construction made by anti-unifying the np-cxn with the transient structure solves the comprehension problem for the observation at hand. It is however too general to add to

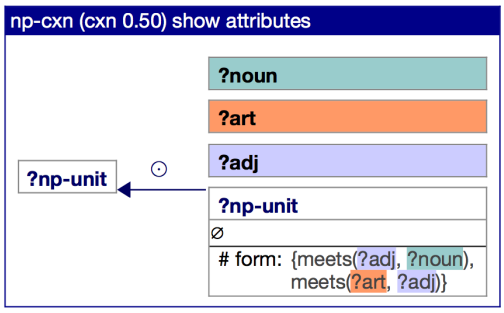


Figure 16: The NP-cxn from the cxn-inventory of the French-learning agent.

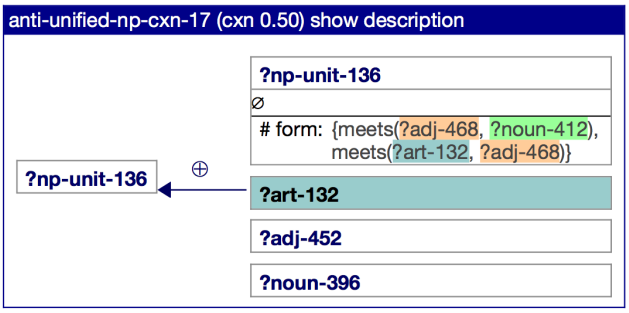


Figure 17: The result of anti-unifying the NP-cxn from Figure 16 with the transient structure from Figure 15. The resulting pro-unified construction.

the construction inventory because its word order features are not 'meaningful' anymore and would except any word order. We want to specialise this construction towards the observation, by capturing the observed word order.

In order to achieve this, we will now pro-unify the construction with the transient structure. The pro-unification algorithm searches for variables in the construction that are bound to the same value in the transient structure. When unifying the anti-unified construction in Figure 17 and the transient structure in Figure 15, we see that the variables ?adj-468 and ?noun-396 from the construction will both be bound to dîner-15 in the transient structure. Likewise, ?noun-412 and ?adj-452 will be both bound to formidable-9. For both cases, the pro-unification algorithm will replace one of the variables from the construction by the other one. This captures the observed word order in the construction. The names of the variables ?adj-468 and ?noun-412 can be confusing, because they are indeed never bound to an adjective and noun. They were however just free variables. Their name was chosen by the anti-unfication algorithm based on the variable names in the original construction, but is as meaningless as ?x ?y, or ?some-variable-name.

The construction that results from the pro-unification process is shown in Figure 18. We can see that it now states that the noun should immediately precede the adjective and that the article should immediately precede the noun. This captures indeed the word order of the observed example. Dur-

ing consolidation, the construction will be added to the construction inventory and will in the future cover any new noun phrases with this word order in routine processing.

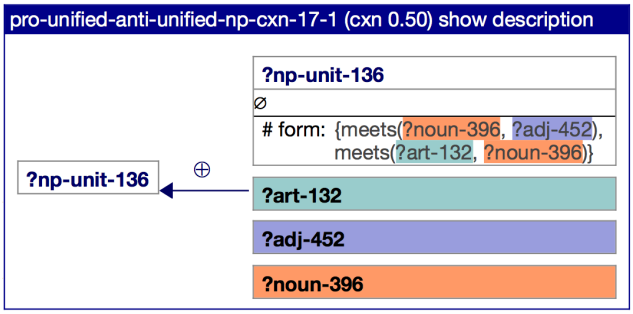


Figure 18: The result of pro-unifying the anti-unified NP-cxn from Figure 17 with the transient structure from Figure 15. The resulting pro-unified construction allows adjectives to occur in postnominal position.

## Related Work

The meta-layer has already a long history in evolutionary linguistics experiments, in which robust communication plays an important role as agents are creating new linguistic conventions from scratch. A previous article on the topic described three different levels of applications that are relevant within the language game approach: the FCG-level (covering linguistic processing itself), the process-level (concerning cognitive processes in the semiotic cycle) and the agent-level (dealing with agent behaviours and turn-taking) (Beuls, Van Trijp, and Wellens 2012). Yet, to use the meta-layer architecture in an FCG grammar, the grammar engineer had to extend the FCG system with their own classes and search heuristics. Our current contribution made an effort to integrate the meta-layer inside regular FCG engineering so that diagnostics and repairs can easily become part of any FCG grammar. Other examples of FCG diagnostics and repairs that were implemented in earlier versions of the formalism are described by (Steels and van Trijp 2011) and (van Trijp 2012).

Cognitive architectures such as ACT-R (Anderson 2007) and Soar (Laird, Newell, and Rosenbloom 1987; Laird 2012) also support the idea of a meta-layer, in the sense of incorporating two processing cycles: A routine layer that is controlled by knowledge retrieved from procedural memory and a meta-layer that modulates this basic processing layer with data from declarative memory. Systems such as NL-Soar applied the Soar theory to sentence processing (Lehman, Lewis, and Newell 1991). A similar system for ACT-R has been presented by Lewis and Vasishth (2005). Paying a lot of attention to the cognitive relevance of their models, ACT-R architectures try to adhere to "well established cognitive constraints" in human language understanding (Ball et al. 2010), something which plays a more peripheral role in Fluid Construction Grammar.

## Conclusion

Agent-based cognitive systems commonly employ a meta-layer architecture for enhancing their robustness. The routine layer then serves routine processing and the meta-layer is used for on-the-fly problem solving. In this paper, we have described the integration of a meta-layer architecture into Fluid Construction Grammar. It is based on diagnostics that detect problems during routine processing, repairs that find solutions to these problems, and consolidation strategies that learn these solutions for later reuse. Besides the architecture, we have presented general and powerful operators that facilitate repair and consolidation in constructional language processing. The meta-layer and these operators make language processing more robust against erroneous input and can accomodate the constant evolution of language by introducing necessary innovations or by learning from the innovative language use of others.

## References


Anderson, J. R. 2007. *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Ball, J.; Freiman, M.; Rodgers, S.; and Myers, C. 2010. Toward a functional model of human language processing. In *Proceedings of the 32nd Annual Meeting of the Cognitive Science Society*.

Beuls, K.; Van Trijp, R.; and Wellens, P. 2012. Diagnostics and repairs in Fluid Construction Grammar. In Steels, L., ed., *Language Grounding in Robots*. Berlin: Springer. 215–234.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial intelligence* 33(1):1–64.

Laird, J. E. 2012. *The Soar cognitive architecture*. MIT Press.

Lehman, J. F.; Lewis, R. L.; and Newell, A. 1991. Integrating knowledge sources in language comprehension. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, 461–466.

Lewis, R. L., and Vasishth, S. 2005. An activation-based model of sentence processing as skilled memory retrieval. *Cognitive science* 29(3):375–419.

Newell, A., and Simon, H. A. 1972. *Human problem solving*, volume 104. Prentice-Hall Englewood Cliffs, NJ.

Steels, L., and Van Eecke, P. 2016. Insight grammar learning using pro- and anti-unification. *forthcoming*.

Steels, L., and van Trijp, R. 2011. How to make construction grammars fluid and robust. In Steels, L., ed., *Design patterns in fluid construction grammar*. John Benjamins. 301–330.

Steels, L. 2011. *Design patterns in fluid construction grammar*, volume 11. John Benjamins Publishing.

Steels, L. 2016. Basics of fluid construction grammar. *Under Review*.

van Trijp, R. 2012. A reflective architecture for robust language processing and learning. In Steels, L., and Hild, M., eds., *Computational issues in Fluid Construction Grammar*. Springer. 51–74.