Insight Grammar Learning

Emilia Garcia-Casademont\(^2\) and Luc Steels\(^{1,2,3}\)

\(^1\)ICREA \(^2\) Institut de Biologia Evolutiva (UPF-CSIC), Spain
\(^3\) Vrije Universiteit Brussel, Belgium

emilia.garcia.casademont@gmail.com, steels@arti.vub.ac.be

We report on computational experiments in which a learning agent incrementally acquires grammar from a tutoring agent through situated interactions. The learner is able to (i) detect impasses in routine language processing, such as missing a grammatical construction to integrate a word in the rest of the sentence structure, (ii) move to a meta-level to repair these impasses, primarily based on semantics, and (iii) then expand or restructure his grammar using insights gained from repairs. The paper proposes a cognitive architecture able to support this kind of insight learning and tests it with a computational implementation for a grammar learning task.

**Keywords:** Language acquisition, insight learning, insight problem solving, Fluid Construction Grammar

1. Learning construction grammars

The *constructional perspective* on grammar (Goldberg, 2006) emphasizes first of all that the structure is motivated by usage, instead of innate, arbitrary, formal principles of universal grammar. Grammar learners have to discover what role the grammatical markers and syntactic patterns play in expressing meaning and achieving communicative functions, and how syntax helps to dampen combinatorial explosions and avoid ambiguity.


Date submitted: 02/11/16  Date reviewed: 02/20/16
Date confirmed for publication: 03/11/16

©2016 Institute for Cognitive Science, Seoul National University
Construction grammar therefore argues that the fundamental unit of grammar is the construction, a sign that relates meaning or function to form through the intermediary of syntactic and semantic categories and structures, including phrase structure, argument structure (cases), grammatical functions, etc.

A construction is very rich, packaging constraints at different layers of language (from phonetics and phonology to morphology, syntax, semantics and pragmatics) and from many different perspectives (phrase structure, functional structure, argument structure, temporal structure, information structure, etc.). Constructions are acquired in a gradual data-driven way with learners creating progressively a huge inventory of more sophisticated constructions linked in networks (Tomasello, 1992). The question addressed in this paper is how constructions can be acquired.

Today, most models of language acquisition use a form of Bayesian unsupervised grammar learning operating over large amounts of data (Bod & Scha, 2003). Because these models are data-driven, they subscribe to one of the key tenets of the usage-based approach to grammar learning that is often associated with construction grammar (Bybee, 2006), namely that language is learned by gradual abstraction of concrete language interactions acting as exemplars. Bayesian language learning has therefore already been used for simulating child grammar acquisition (Bannard, Lieven, & Tomasello, 2009). Although we see that there is great value and interesting results following this approach, we explore here a complementary learning method which views grammar learning as a form of insight learning. We believe that insight grammar learning is an essential tool in the tool box of the language learner.

1.1 Problem solving

The standard model of (routine) problem solving (Newell & Simon, 1972) conceptualizes problem solving as a process that starts from an initial problem state and a set of operators that transform this state into a final goal state. The classical example is chess, where the initial state is the board position of the player, the operators are the rules of chess, and the goal state is a winning game for the player. The intermediary state are possible
board positions between initial and final state. In the case of language, the states are called transient structures. It captures all that is known about an utterance (whether in parsing or producing). The operators are constructions, forming a continuum from lexical to grammatical constructions, or possible inferences using the situation model or world knowledge. In the case of language production, the initial transient structure (i.e. initial problem state) is the meaning to be expressed, the possible operators are constructions used to map meaning to form (e.g. wordstrings) or intermediary structures (such as phrase structure), and the final transient structure (or goal state) is a complete description of the form of an utterance. In the case of language comprehension, the initial transient structure is an utterance, the possible operators are again constructions, now used to map form to meaning or intermediary structure, and the final transient structure contains the complete syntactic and semantic structure.

For most non-trivial problems, there is more than one possible operator that can be applied and so the language processor needs to search to find a path from the initial state to the final goal state. The search tree gets combinatorially explosive if the path from initial to final state is long and many nodes can be explored at each node in the tree. Human language users apply a lot of heuristics to pick the best path to follow or to restrict search to a particular depth. They also bring world knowledge and the context to bear as much as possible, as well as backtracking from a dead-end to an earlier stage in the search tree when a wrong choice was made. This situation occurs in chess - and makes the game difficult - but also in language because of ambiguity of words, morphemes, and grammatical patterns.

1.2 Insight Problem Solving

There exists a class of problems, called Insight Problems (Ohlsson, 1984), which cannot be handled in the usual way because the operators or the representation of the problem need to be revised, in order to make the problem solvable. Chess does not belong to this class because the rules of the game are fixed. A well known example where this issue arises is the famous nine-dot problem, where the problem solver has to revise a
constraint (staying between the lines) that one would normally impose on the solution. The operationalization of insight problem solving requires the ability to reason at a higher, i.e. meta, level.

Meta-level problem solving starts when routine processing (with the existing operators and problem representations) reaches an impasse, or when a diagnostic fires because the (possibly partial) solution in a node of the search tree is not acceptable. Meta-level operators then come in action to unblock the impasse. For example, they may reinterpret the problem statement by relaxing some of its constraints (as needed for solving the 9-dot problem), they may change internal representations inferred from the problem (as in the Anthony and Cleopatra problem (Patrick & Ahmed, 2014), or possibly introduce new operators. One very common meta-operator, studied in particular by Koehler in his classic insight learning with chimpanzees, is to coerce objects to have functions that they normally do not have (Koehler, 1956). For example, to view a shoe as a hammer so that it can play the role of the instrument in a hitting action.

There is still a final step. After solving a problem using meta-operators, we can imagine a process whereby a learner tries to expand his available repertoire of routine operators (i.e. constructions) to capture the insight gained from dealing with the impasse to ensure that in the future the impasse does not occur anymore. When that is the case, the literature talks about insight learning (Koehler, 1956).

The human capacity for insight problem solving is not in doubt as it has been demonstrated through a large number of protocols used by

Figure 1. Left: The nine dot problem requires that 4 straight lines are drawn to connect the nine dots. Right: It can be solved by relaxing the constraint that the lines have to be inside the $3 \times 3$ square created by the dots, which many problem solvers initially believe to be part of the problem statement.
cognitive psychologists. There have been recent neuro-imaging experiments identifying areas involved in insight problem solving (Anderson, Anderson, Ferris, Fincham, & Jung, 2009) and there is a steady stream of prior work on computational simulations of insight problem solving, using at first rule-based production systems (such as OPS5) and then more sophisticated architectures such as SOAR (Laird, 2008), which has similar mechanisms for detecting impasses and learning from their resolution.

1.3 Language and insight

We map this framework in the following way to language. The problem is either to reconstruct the meaning of an utterance, as listener, or to produce an utterance expressing a target meaning, as speaker. Producing or comprehending an utterance obviously requires a large number of decisions: What words will used to express the meaning? Which grammatical constructions are best used? How should a particular sequence of words be parsed? What is the interpretation of semantic structure? etc. Often these decisions cannot be taken on the available evidence. For example, an ambiguous word or a word with multiple functions may appear in the sentence, forcing the listener to explore different possible paths, or there is more than one possible way in which the same meaning can be expressed, but some of them might not fit with additional meanings that have to be expressed as well. So we get also a search space. We call the nodes in the search space transient structures, they contain information extracted from the input (the utterance in comprehension or the meaning in production) and additional information that could be derived by the application of constructions, world knowledge, or situation knowledge. For example, a construction might combine an article and a noun already present into a transient structure into a noun phrase. Some exploration and backtracking may be necessary to know which construction needs to be applied at each step.

An impasse in this context means, for example, that the hearer encounters an unknown word, a particular word does not fit within the syntactic pattern implied by its immediate context, an ordering constraint is violated, no structure can be found that integrates all words in the utterance, there is
syntactic and semantic ambiguity, implying that some grammatical signals preventing ambiguity have not been picked up properly, etc. These impasses are frequent when learning a new language. The speaker may also reach an impasse because, for example, he may lack a word to express a particular meaning, a word he would like to use does not fit with a construction already chosen, the available constructions may leave significant ambiguity, etc.

Meta problem solving for language involves a set of meta-operators in the form of diagnostics and repairs and a cognitive architecture as in Figure 2. For example, the listener could try to handle a sentence which he cannot parse by ignoring possible errors in it (such as the wrong word order). Or he may infer the unknown meaning of a word by inferring it from the syntactic context, the situation model or the ontology. The speaker could coerce a word to have an additional lexical category so that it fits in a given construction, as in “He googled him” where the noun “Google” has been coerced into a verb.

In the case of language, insight learning takes place after a repair. It is based on a set of consolidation operators that make small changes to the grammar, for example, add a new lexical category to the lexical construction of a word or make a construction more general by relaxing

![Flow diagram. There is a layer of routine decision-making, diagnostics continuously monitoring activity at this layer in order to discover an impasse, and meta-level operators try to repair the impasse, for example by coercing a word to fit with an available grammatical construction.](image-url)
some constraints on one of its slots.

The main contribution of this paper is to show, for the first time, how insight language learning can be operationalized and tested in computational simulations. An appendix, which is seen as an integral part of the paper, provides more technical detail which is further enhanced with an online web demo available through www.biologiaevolutiva.org/lsteels/demos/J-Cog-Scie-2016/. The demonstration shows tutor-learner interactions, where the tutor acts as speaker and the learner as listener.

Our goal is of course not to identify the full set of diagnostic, repair and learning operators or to handle examples from the full scope and complexity of a human language, that would not be possible to cover in a single paper and the results would be very hard to analyse anyway, but rather, to test the feasibility of the approach with careful very challenging examples, which requires concrete examples of meta-operators (diagnostics and repairs) and learning-operators. We therefore use a minimal representation of meaning and focus on a minimal form of syntactic structure, namely phrase structure. As the reader will see from studying the appendix, the computational challenges are already very high.

Another aspect of the paper is the usage of a fully operational and freely downloadable computational implementation of construction grammar known as Fluid Construction Grammar (Steels, 2011). Obviously, explaining all the details of this formalism is totally beyond a single paper as well, and the reader is encouraged to consult the website: https://www.fcg-net.org/ which contains the software as well as background papers, tutorials, demonstrations and publications. However, the present paper can be understood to a large extent without being familiar with this formalism.

The next section first describes routine language processing using constructions. Then we turn to meta-level problem solving, discussing syntactic and semantic expansions, and finally to learning operators. The paper concludes with some experimental results. All mechanisms discussed in this paper have been formalized and implemented and are described in more detail in the appendix and the web demo.
2. Routine language processing

2.1 Meaning representation

For the purposes of the experiments reported here, we have defined a ‘situation model generator’ that generates possible situation models in the same format as obtainable by visual processing, as described in (Spranger, Loetzsch, & Steels, 2012). The representation of the situation model uses well-known standard techniques from (typed second order) logic. The situation model is internally represented in terms of n-ary predicates, which have objects perceived in the world as arguments. The objects are labeled obj-1, obj-2, etc. The objects can be bound to variables which are written down with a question mark in front, as in ?obj-1, ?obj-2, etc. Each predicate consists of an attribute, that acts also as a type, and a value.

Unary predicates are written down as:

\[
\text{type(predicate, object-to-which-predicate-applies)}
\]

as in

\[
\text{color(red, obj-1) or material(plastic, ?obj-6)}
\]

In the first example, the color red is predicated about obj-1, i.e. obj-1 is red in the current situation model. color is the type of red and used later for semantic categories in the grammar. In the second example, plastic is predicated about a variable object ?obj-6. The type of plastic is material. It will be true if there is an object bound to the variable ?obj-6 in the world which has the material property plastic.

N-ary predicates are decomposed into a number of separate predicates. One for the predicate itself which has also a type and an object for the relation, and then predicates for each of the arguments. For example, suppose there is a predicate moving-away of type movement (which is for all types of movement) with the value away, then there are two predicates for its arguments, as illustrated in the following example.

\[
\text{movement(moving-away, ?r-2); the main predicate}
\]
\[
\text{moving-away-arg-1(?r-2 ?o-3); the object that is moving}
\]
\[
\text{moving-away-arg-2(?r-2 ?o-1); the object being moved away from.}
\]
?r-2 gets bound to the event of moving, ?o-3 to the object that is moving and ?o-1 to the object ?o-3 moves away from.

Different predications can be combined into conjunctions and they are linked together by reusing the same variable-names or object-names. For example, the utterance ”a small paper moves away from a wooden table” would be represented as

\[
\begin{align*}
\text{movement}(\text{moving-away},?r-2) & \text{; the main predicate} \\
\text{moving-away-arg-1}(?r-2,?o-3) & \text{; the object that is moving} \\
\text{moving-away-arg-2}(?r-2,?o-1) & \text{; the object being moved away from} \\
\text{size}(\text{small},?o-3) & \text{; the moving object is small} \\
\text{material}(\text{paper},?o-3) & \text{; and its material is paper} \\
\text{physobj}(\text{table},?o-1) & \text{; the source object is a table} \\
\text{material}(\text{wood},?o-1) & \text{; and it is made of wood}
\end{align*}
\]

The different equalities between the objects acting as the arguments of predications are represented graphically as a semantic network (See Figure 3). One of the objects in this network is the topic of the utterance, for example, the event itself (i.e. ?r-2) as in “the moving away of the small paper” or the object which is moving (i.e. ?o-3) as in the utterance “the small paper moving-away from the wooden table”.

**Figure 3.** Semantic network representing the meaning of an utterance. Each node is a predication and the links represent referential equalities between the arguments.
To simplify the experiments, we use utterances with only content-carrying lexical items and only word order and phrase structure as the means for expressing syntactic structure, ignoring other syntactic devices like morphology, grammatical function words, agreement, etc. Thus, the utterance “a small paper moves away from a wooden table” would be rendered as “small paper moves-away-from wooden table”. There is of course no necessity to use English-like words, that is only done to make the utterances understandable for the reader, and there is no reason why the emergent grammar should be English-like, although English also makes heavy use of phrase structure.

2.2 Grammar representation

We use Fluid Construction Grammar (FCG) for the representation of the grammar (Steels, 2011). FCG views language processing in terms of operations over transient structures. A transient structure captures all that is known about a particular utterance being parsed or produced. In routine language processing, transient structures are expanded by the application of constructions in a process of matching (to see whether the construction can apply) and merging (to add information from the construction to the transient structure).

FCG represents transient structures in terms of feature structures, similar to many other computational formalisms in use today, such as HPSG (Pollard & Sag, 1994). A feature structure consists of a set of units which correspond to words and phrases, and features associated with these units (see appendix).

A construction is an association between meaning and function and form constraints. Lexical constructions associate one or more predicates and a semantic categorization of the predicates (equal to the attribute (i.e. type) of the predicate) with the occurrence of a word-string and lexical categories (i.e. parts of speech) of that word. Grammatical constructions, in this case restricted to phrase structure constructions, associate a pattern of units with semantic constraints with syntactic categories (lexical or phrasal categories) and word orders. Each construction has a score (between 0.0 and 1.0) which reflects the success of that construction in past language interactions.
Constructions in FCG consist of two parts. A conditional part (written on the right hand side) which specifies under which conditions a construction can become active and a contributing part (written on the left hand side) which specifies what the construction contributes to the transient structure. Constructions must be usable both in comprehension and production. So the conditional part is split into two 'locks'. A production lock (on top) which has to match with the transient structure in production and a comprehension lock (below it) which has to match in comprehension. When a lock fits with a transient structure all the information of the construction (the other lock and the contributing part) gets merged in. The fact that constructions in FCG can be used both for comprehension and production is crucial for insight learning because once the learner has been able to deduce the meaning (possibly indirectly), it can produce the same meaning himself with its own inventory and thus compare it to the utterance produced by the tutor.

A lexical construction for the word “paper” looks as follows:

```
?paper
referent: ?obj
args: {?obj}
sem-cat: material
potential-syn-cat: {adj, noun}
```

```
?paper
# meaning:
  {{material paper ?obj}}
# form:
  {{string ?paper paper}}
```

“Paper” has two potential lexical categories: adjective (as in “a paper basket”) and noun (as in “a small paper”). An example of a simplified phrasal construction is:

```
?np-unit
constituents:
  {?word-unit-1, ?word-unit-2}
sem-cat: material
syn-cat: noun-phrase
head: word-unit-2
referent: ?obj
```

```
?word-unit-1
sem-cat: size
referent: ?obj
potential-syn-cat: {adjective}
```

```
?word-unit-2
sem-cat: material
referent: ?obj
potential-syn-cat: {noun}
```
The details of FCG constructions are explained in the appendix and in the growing literature on FCG.

3. Repair processing

The consecutive application of constructions expands the transient structure to go from meaning to form or vice versa. But occasions will arise when no construction can be applied, particularly in language learning. The listener then moves to the meta-level to repair the situation and then consolidate the outcome of the repair.

3.1 Syntactic-based repair-operators

The listener can try to find a construction which is only partially matching, and either coerce units to fit into that construction, for example, coerce a word to have a particular lexical category, or accept a variant of the construction, for example, accept a slightly different word ordering. For the experiment reported here we have operationalized the following repairs, but our architecture is general enough to include many more cases:
+ **Lex-cat-coercion**: A construction is found that is semantically compatible but one or more words do not have the appropriate lexical category (as in the example of “googled” where a noun occurs in a context where a verb is expected). The repair-operator then adds the lexical category to the word-unit in the transient structure and the construction can apply.
+ **Ordering-Variation**: A construction is found that matches with the current situation, however there are constraints on the construction, specifically ordering, which are violated. The learner can then construct a variant of the existing construction with this new ordering.

3.2 Semantics-based repair-operators

When no partially matching constructions could be found, it is possible to use the situation model and combine units for words or phrases based on semantics. For the present experiment, we have included an example of such an operator:
+ **Build-Hierarchy**: When a relational word is encountered, i.e. a word which introduces a predicate with more than one argument, such as `moves-away-from`, and no constructions are available to integrate it in the transient structure, the repair-operator looks in the world-model to detect which object plays which role and then combines the units for these objects into a new hierarchical unit. The repair-operator also needs to decide which of the arguments is going to be the referent and hence which argument will be the head of the phrase. The referent is determined by the structure of the meaning that needs to be expressed. Specifically, the referent is the entity that is on the shortest path towards the topic of the utterance as a whole. For example, in the sentence “the ball on the table”, the referent of the unit based on the relational word “on” is the ball, because that is the topic of the utterance as a whole, whereas in “He wants the ball (to be) on the table” the referent is the on-relation itself.

New lexical and phrasal categories are created and expanded by the learner as a side effect of these repair operators. For example, Build-Hierarchy either creates new lexical categories for the words linked into a new construction or it reuses existing lexical categories and thus changes their categorical definition.

### 3.3 Consolidation

When an utterance could be successfully parsed after one or more repairs, the learner activates consolidation-operators that integrate the insights that were obtained into his construction inventory. In some cases this is straightforward, for example, lex-cat-coercion can be consolidated by adding the relevant lexical category to the potential lexical categories of a word. In other cases, more complex ‘linguistic engineering’ is required. Before a change is made to a construction it is first copied and only then additions and changes are made, because the existing construction may still be useful in other syntactic contexts.

### 3.4 Alignment

The repair-operators and learning-operators are hypotheses made
by the learner about the language of the tutor and mistakes such as overgeneralizations are unavoidable. The learner therefore needs an additional mechanism to progressively discard wrong hypotheses based on further input. We have achieved this by updating the score of constructions using the well known lateral inhibition learning rule. Knowing which constructions $c_i$ need an increased score is easy: they are the constructions that were used on the path towards the final transient structure. We use the following update rule: $\sigma_{ci} \leftarrow \sigma_{ci} \cdot (1 - \gamma) + \gamma$, with $\gamma = 0.2$ a constant.

Determining which competing constructions $c_j$ need to be decreased is more difficult. First of all, this set includes all constructions that started off a wrong branch in the search space during comprehension, i.e. a branch which was not on the path to the final solution. Second, the listener can produce himself the utterance based on the meaning deduced from the comprehension process and possibly further feedback from the tutor in the case of miscommunication and thus find all constructions that start off a wrong branch while producing. The scores of competing constructions need to be decreased as well. We use the following update rule: $\sigma_{cj} \leftarrow \sigma_{cj} \cdot (1 - \gamma/2)$.

4. Results

The appendix gives concrete examples how routine language processing as well as insight problem solving and learning works. In this section we report two experiments exercising the cognitive architecture in Figure 2 and the repair- and consolidation-operators described in the previous section on a larger scale. The tutor is initialized with a lexicon of 40 lexical constructions and a grammar with 30 grammatical constructions. The tutor grammar includes adverbs, adjectives, nouns, verbs, prepositions, pronouns and relative pronouns as well as noun phrases of different levels of complexity, verb phrases, main clauses and relative clauses. The tutor produces a stream of utterances each describing a particular topic (object or event) in a scene. Some example utterances are “Paul sees (the) red carpet (that) Emilia wants”, or “Paul believes (that) Emilia wants (the) carpet on (the) big wooden table”. The learner is initialized with the same lexicon (because we focus here on the acquisition of grammar) and endowed with
the various operators described above, but WITHOUT any grammatical constructions nor any syntactic categories (neither lexical nor phrasal).

In each concrete interaction (further called a language game), tutor and learner share the same situation model (represented as a semantic network as in Figure 3). The tutor then chooses a topic (one object to be referred to) and produces a description of this topic using his lexicon and grammar. Then the learner tries to parse the utterance and interpret the extracted meaning against the situation model. The interaction is successful when the learner was able to identify the topic originally chosen by the tutor. This may involve both routine and meta-level processing. Each experiment is carried out for 5 different tutor-learner pairs, using random choices from a set 20 situations, so that results can be compared.

4.1 Experiment 1. Lexical categories given

The first experiment tests the architecture and specific learning operators assuming that the learner already knows the lexical categories, i.e. parts of

![Figure 4. Grammar learning with lexical categories known. 800 language games are shown (x-axis). The learner reaches the same number of grammatical constructions (namely 30 on right y-axis) and a total alignment (left y-axis), demonstrating that it successfully acquired the grammar. The shading around the lines represent the confidence interval of 5 runs.](image)
speech, of the words in the lexicon. Moreover grammatical constructions are limited to those having only one semantic category and one syntactic category per subunit. The task is to learn the grammatical constructions. Figure 4 shows the results, measuring communicative success, inventory size (the number of constructions of the learner) and alignment (how often the learner’s reproduction of the meaning of an utterance is equal to the utterance of the tutor).

4.2 Experiment 2. Lexical categories not given

The second experiment assumes that the learner does not know any a priori lexical categories. This obviously makes the learning problem harder. Also, grammatical constructions can have more than one semantic category per slot, which means that constructions can be more general. They still have only one syntactic category for slots, whereas individual words can have several lexical categories (syncretism). Because constructions can be more general, we end up with a smaller inventory.

Results for inventory size and alignment are shown in Figure 5. The graph also shows syntactic and semantic ambiguity encountered by the learner. Let $B^*$ be the number of branches in the search space that are blocked on semantic grounds, i.e., where grammar introduced co-referential link that are not valid in the current context. Let $H$ be the total number of hypotheses explored by the semantic meta-operators before getting a result (for example extend or build a group, or build a hierarchical unit based on a relational word); and $H_F \subseteq H$ the subset of hypotheses in $H$ that fit with the utterance form. We define syntactic ambiguity as in equation (1):

$$\frac{B^* + H_F}{\#\text{words}}$$

and semantic ambiguity as in equation (2)

$$\frac{H}{\#\text{variables}}$$

whereby n-ary relation-words introduce n variables and attribute word (which introduce a unary relation) 1 variable.
Figure 5. Grammar learning without lexical categories known. The learner reaches fewer grammatical constructions (25) and a total alignment, although this takes somewhat longer compared to Figure 4. The graph shows 800 language games and the confidence interval for 5 runs using random situations from the same set in each run.

Figure 6. MDS plot showing how the categories of the learner converge to the categories of the tutor. The tutor categories are labeled with recognizable names, such as ‘noun’ or ‘prep’, whereas the learner categories are labeled as syn-cat-6, syn-cat-11, etc.
Thanks to the increase in grammar both types of ambiguity get drastically reduced, which proves that the key function of grammar, namely to dampen combinatorial search, is progressively better achieved as the grammar grows.

Figure 6 shows that the lexical categories of the learner converge to those of the listener. Categories cannot directly be compared so we use an indirect way. First, we construct a vector $V_i$ for each category $c_i$: $V_i = [... b(w_j) ... ]$ where $b(w_j) = 0$ if the word $w_j$ is assigned $c_i$ in the lexicon and 1 otherwise. The categorial dimensions are then reduced to a 2-dimensional plot using a standard well-known Multidimensional Scaling algorithm (Cox & Cox, 2001).

5. Conclusion

The paper explored in how far grammar learning can be modeled as insight learning which consolidates the outcome of insight problem solving. The computational simulations showed that this is possible requiring (i) a strong semantic component to make meaning-driven learning possible, (ii) a meta-level architecture similar to SOAR and (iii) appropriate diagnostics, repair, consolidation and alignment operators.

The experiments reported here use generated situation models, but we have also carried out experiments on robotic agents that extract their situation model using perception and scene recognition (Spranger et al., 2012), and that works just as well. Tutor and learner do not necessarily have the exact same world model but as long as it is sufficiently similar, insight grammar learning is feasible. We note also that the approach does also work just as well for virtual world models, i.e. dialogs about objects which are not shared or objects that are not directly sensed. Often human language remains ambiguous (for example for the attachment of prepositional phrases) which then gets resolved using semantics. However the proposed system contains a full semantic component to which language processing can fall back.

Of course, the model is much simpler compared to the full complexity of the resources that human language learners can bring to bear on language acquisition, specifically we have not integrated a strong component for
inference although such components exist. However, the advantage of a simpler system is that we could develop and test the basic architecture in computational terms. Moreover this method allows us to causally examine the impact of each operator in detail, and thus supports empirical research into which operators are available to human language learners.

Finally, we stress again that the point of the paper is to introduce the general architecture for insight grammar learning and validate it. There are many ways in which we can proceed further. The first one is to extend the tutor grammar with many more constructions that fall within the scope of the diagnostics and repairs currently already implemented. Second, the diagnostics and repair strategies so far are strongly geared towards phrase structure grammar learning, but there is of course much more going on in language grammars, for example agreement (such as subject-verb agreement), marker systems (such as case marking), or anaphora which would require additional diagnostics and repairs. Finally, the representation of meaning can be made more complex to support additional linguistic phenomena, such as the intentional use of concepts (as in ‘the beautiful dancer’ when it means the dancing is beautiful versus the dancer is beautiful) or additional conceptualization information, such as figure/ground (to distinguish ‘John met Mary’ vs. ‘Mary met John’). These extensions add broader language coverage and would further validate the proposed architecture.

Acknowledgements

The research reported here has been funded by an ICREA Research fellowship to LS and a Marie Curie Integration Grant EVOLAN. The project has received further funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under grant agreement no 308943, which is the FET-OPEN Insight project. We are indebted to Paul Van Eecke and Katrien Beuls for comments.
References


7. Appendix I. Technical details and examples

The goal of this appendix is to provide more technical details about the implementation, partly through explanations in the text and through a ‘web demo’, which is accessible through this www.biologiaevolutiva.org/lsteels/demos/J-Cog-Scie-2016/. Although the details provided here are not important to understand the main point of the paper, they clarify and prove that the proposed system is fully operational. Moreover they provide valuable information for readers competent in computational linguistics. The examples given here have been simplified as much as possible in order to remain comprehensible and fit within available space limitations. The reader is strongly advised to consult other publications on FCG (referenced in the bibliography) to understand these examples more thoroughly.

The examples use English-like words, but it is important to keep in mind that we are not targeting a treatment of English and we use only simplified expressions. For example, there is no morphology so that all occurrences of ‘see’ are rendered as “sees”, we may also use different word orders than English.

7.1 Transient structures

As explained in the main text, a transient structure represents all information about an utterance being parsed or produced. A unit has a name which is chosen to make sense by human readers. Names are mostly generated by the FCG-system itself. New names are made whenever a new token of a particular type is needed. For example, noun-1, noun-2, ... are all tokens of the type noun. The names do not have any meaning beyond their role in the overall system, although an FCG-designer can influence which names get created. Variables are names with a question mark in front, as in ?noun, ?giver, ?agent, ?number, etc. These variables get bound by the matching and merging which is at the core of FCG. There can be as many units as needed. There is only one special unit, called ‘root’. It can be considered as an input/output buffer. In comprehension, the root starts out with a description of the form of
the utterance. In production, it starts with the meaning that needs to be expressed and the topic.

There is no ordering in which features appear within units in a transient structure and a feature may be absent. The examples to be developed here employ the following features, which we can categorize as semantic, syntactic and control features (see Figure 7).

### 7.1.1 Semantic features

**Args**: The arguments of the meaning are the entities that are made accessible to other units, so that it is possible to handle compositional semantics. They are specified as a sequence. For example, the arguments of ‘cog-action(see,r-1), see-arg-1(r-1,o-1), see-arg-2(r-1,o-2)’ are [r-1,o-1,o-2]. We occasionally need to index the argument list and then use indices from 0 to n based on the argument sequence. The 0th argument always introduces the relational entity, in this case r-1, the 1st argument is
here o-1, and the 2nd argument o-2.

**Referent:** One of the arguments of the meaning of a unit acts as its referent. This is the object the unit refers to. For example, in the case of ‘a gift from Miquel’, the meaning of ‘gift’ is ‘phys-action(give,r-2), give-arg-1(r-2,?giver), give-arg-2(r-2,o-1), give-arg-3(r-2,Miquel)’ and the referent is ‘o-1’ (the gift). In the case of ‘the giver of a book to Miquel’, the referent of the same predication would be ‘?giver’, i.e. the giver.

**Meaning:** This feature contains the set of predications expressed by the unit. As already explained in the text, we follow a variant of standard second order typed predicate calculus. A predicate with a single argument, e.g. P(x), is denoted as T(P,x), where T is the type of P. For example, ‘red(o-1)’ is represented as ‘color(red,o-1)’, where color is the type and red the specific attribute. A predicate with multiple arguments, e.g. R(x,y,z), is transformed as T(R,u), R-arg-1(u,x), R-arg-2(u,y), R-arg-3(u,z), with T the type, R the relation, and u the relation instance. R-arg-1, R-arg-2, ... introduce the different arguments of the relation which always take the relation instance u and the filler of the argument as arguments. For example, ‘see(o-1,o-2)’ is represented as ‘cog-action(see,r-1), see-arg-1(r-1,o-1), see-arg-2(r-1,o-2)’. r-1 is here the reification of the relation itself, i.e. the seeing event. o-1 is the agent who is seeing and o-2 the object being seen.

**Sem-cat:** This feature contains the type of the predicate. For example, if a unit has the meaning ‘color(red,o-1)’ then its type is ‘color’.

### 7.1.2 Syntactic features

**Form:** The form-feature of a unit contains a set of predications circumscribing the form of the unit. It uses the names of units as argument, because these descriptions are not local to a single unit. When parsing, form descriptions first appear in the root unit, even before any units for individual words have been made. The first predicate that can be used for describing form is ‘string’, which takes two arguments: the unit and the string itself, as in ‘string(paul-15, “paul”)’ which specifies that unit ‘paul-15’ has the associated string “Paul”. The second predicate, called ‘meets’ specifies whether two units are directly adjacent to each
other, for example, ‘meets(paul-15, wants-17)’ specifies that the form
associated with the unit paul-15 (which covers the word “Paul”) precedes
the unit named ‘wants-17’.

**Syn-cat:** This feature contains the syntactic category of the unit. These
are either lexical categories (parts of speech) associated with nouns or
phrasal categories (such as noun-phrase) associated with phrases.

**Potential-syn-cat:** This feature, only used with lexical units, contains
a list of the possible syntactic categories. Most words have multiple
associated categories and which one is valid in a particular case is
chosen by the grammatical construction. There is no ranking between the
potential syn-cats.

**Subunits:** The hierarchical structure of units is also represented
explicitly using a feature called subunits. It contains an ordered list of
the constituents of a unit. The subunits-feature is used to draw tree-like
structures in the FCG-interface.

**Head-of:** The head-of feature contains the name of the hierarchical parent
unit for which the unit functions as the head in the linguistic sense, i.e.
the constituent that determines many features of the phrase as a whole
such as its syntactic features like number or semantic features like color.
For example, the head of a noun-phrase is the noun, and hence the head-
of feature of a noun unit would be filled with the name of the noun-phrase
concerned.

**Boundaries:** During processing (particularly in production), the exact
position of a unit (a word or a phrase) may not yet be known. So we use
an explicit representation of hierarchical order adapted from dependency
grammars and chart parsing. Every unit X has a feature called boundaries,
which specifies the boundaries of that unit in terms of two predicates:
‘from-left(Y)’ and ‘from-right(Z)’. Y is the leftmost unit of the phrase and
Z the rightmost unit. For a single word, these boundaries are equal to the
word-unit itself. For example, in the phrase “big ball moves-away-from
table”, the unit for big, let’s say big-12, has the following boundaries:
from-left(big-12) and from-right(big-12). Suppose the unit for ball is ball-
55, then the boundaries of the noun-phrase with big-12 and ball-55 has
the following boundaries: from-left(big-12), from-right(ball-55).
7.1.3 Control features

Control features are used for bookkeeping and controlling the flow of construction application. In the present examples only one feature, called ‘footprints’ is used.

**Footprints:** When a construction applies, it leaves a footprint behind that it was used. More than one construction can apply to the same unit, progressively building more of the syntactic and semantic structure. Each footprint is a pair, with the position of the unit in the pattern (starting from 1 for the first slot) and the identifier of the construction itself (explained below).

7.2 Constructions

Construction schemas use the same features as those used in transient structures and discussed in the previous paragraph. There are two types of constructions for the present experiments: lexical constructions that define the schemas for words, and phrasal constructions that define the schemas for phrases.

Lexical constructions are called X-cxn where X is the name of the string involved. For example, ‘table-cxn’ is the name of the construction for ‘table’. A lexical construction associates a meaning (a set of predications) with a form (a string). The meaning is the production lock (it has to be there in production for the construction to trigger) and the form is the comprehension lock (it has to be there in comprehension). Both of them are written on the right side of the arrow. If the transient structure fits with the lock then the contributing part merges the following features. Semantically, we get the arguments (args), the referent, and the semantic category (sem-cat). Syntactically, we get the potential lexical categories (potential-syn-cat). An example for the table-cxn is as follows:

\[
\begin{align*}
?\text{table} & \quad \text{←} \quad ?\text{table} \\
\text{args: [?o]} & \quad \text{# meaning:} \\
\text{referent: ?o} & \quad \{(\text{physobj table } ?o)\} \\
\text{potential-syn-cat: } \{\text{noun}\} & \quad \text{# form:} \\
\text{sem-cat: physobj} & \quad \{(\text{string } ?\text{table table}\})
\end{align*}
\]
A phrase defines a sequential pattern with a number of slots which are indexed from 1 to m. Each slot requires a particular semantic and syntactic category and introduces a particular argument of the relation covered by the pattern. One of the slots is responsible, on the semantic side, to introduce the referent, and on the syntactic side to function as the head of the phrase.

An example of a compact diagram to represent a construction is as follows:

```
| sem-cat:  person  cog-action(0,1,2)  physobj |
| arguments: 1  0  2 |
| syn-cat: NP transitive-verb NP |
```

cog-action transitive-verb-phrase

This construction builds a transitive-verb-phrase with two noun phrases and a transitive verb. Slots correspond to columns, and the sequential order is from left to right. The first line introduces the semantic categories of each slot. Here there are three slots with semantic categories ‘person’, ‘cog-action’ and ‘physobj’ respectively. The cog-action has 3 arguments: the action itself (0), the actor (1), and the object of the action (2). The next line specifies which slot fills which argument. For example, the first NP fills argument 1 of the cog-action. The + indicates which slot is the head, which implies that the corresponding argument (in this case 0, i.e. the action itself) is the referent of the phrase being built by this construction. The next line contains the syntactic category (lexical or phrasal) of each slot. Under the line, we see the semantic category (cog-action) of the parent phrasal-unit that this construction will build as well as its phrasal category (transitive-verb-phrase).

The links between the meaning to be expressed and the utterance produced through this construction is shown in the following diagram on the next page.

A construction identifier contains the same information as the construction diagram in terms of a list of specifications for each slot 0, ... , n:

```
((sem-cat_0 syn-cat_0 arg_0 is-head-p_0) ... )
```
sem-cat and syn-cat are the semantic and syntactic categories of a possible slot-filler, arg is the numerical index of the argument and is-head-p is a

meaning: person(Paul,01) cog-action(sees, ev1, o1, o2) physobj(table,o2)

sem-cat: person cog-action(0,1,2) physobj
arguments: 1 0 2
syn-cat: NP transitive-verb NP
utterance: Paul sees table
cog-action transitive-verb-phrase

Boolean which specifies whether the slot is the head of the phrase built by the construction (+) or not (-). There can only be one head and only one referent.

Here is an example of a construction identifier which conveys the same information as the diagram above:

((person np 1 -)
 (cog-action transitive-verb 0 +)
 (physobj np 2 -))

This construction defines a sequential pattern with three slots. The first requires a person/NP, the second one a cog-action/transitive verb, and the third one a physobj/NP. 1, 0, and 2 are the indices of the respective arguments of each slot. The cog-action/transitive-verb is introducing the head and hence the referent.

```
?transitive-verb
  head-of: ?transitive-verb-phrase
  footprints:
  {2 (person np 1 -)
   (cog-action transitive-verb 0 +)
   (physobj np 2 -))}

?transitive-verb-phrase
  referent: ?event
  sem-cat: cog-action
  syn-cat: pred
  subunits:
  {?subject, ?transitive-verb, ?object}

?subject
  footprints:
  {1 (person np 1 -)
   (cog-action transitive-verb 0 +)
   (physobj np 2 -))}

?object
  footprints:
  {3 (person np 1 -)
   (cog-action transitive-verb 0 +)
   (physobj np 2 -))}
```
There are details necessary for operationalizing this construction which are not expressed in the construction diagram or in the construction identifier. To see those, we have to look at the FCG-definition. For the same construction, it is as above.

In the experiments reported here, one agent (the tutor) is given an inventory of lexical and phrasal constructions. The learner also gets the same lexical constructions, however without knowing their lexical categories, and has to acquire the phrasal constructions and lexical categorizations in an incremental fashion through situated interactions.

### 7.3 Examples

The web-demonstration contains the following examples.

1. Routine processing of a clause: This illustrates parsing and production of a simple clause with a main transitive verb and a subject and object and without learning.

2. Routine processing of a Noun phrase: This illustrates parsing and production of a Noun phrase with a relative clause, still without learning.

3. Learn first hierarchy level: The next example shows how a new construction for a single noun gets built from scratch, showing how

```plaintext
?subject
referent: ?var-589
sem-cat: phsyobj
syn-cat: np
boundaries: {from-left(?2-left), from-right(?2-right)}
syn-cat: np
sem-cat: phsyobj

?transitive-verb
args: [?var-587, ?var-588, ?var-589]
sem-cat: cog-action
potential-syn-cat: {transitive-verb}
referent: ?event
sem-cat: cog-action
potential-syn-cat: {transitive-verb}

?object
referent: ?var-588
sem-cat: person
syn-cat: np
boundaries: {from-left(?0-left), from-right(?0-right)}
syn-cat: np
sem-cat: person

?transitive-verb-phrase
∅
# boundaries: {from-left(?0-left), from-right(?2-right)}
# form: {meets(?0-right, ?transitive-verb), meets(?transitive-verb, ?2-left)}
```
units at the first hierarchical level are formed.

4. Learn higher levels: shows how a new construction of a relational noun phrase gets built from scratch, showing how units at higher levels get formed.

5. Lex-cat-coercion: A particular word is coerced to have an additional lexical category so that an existing construction in which the word did not fit can be reused.

6. Ordering-variation. A construction does not fit because an ordering constraint is violated, for example two constituents occur in another ordering. The constraint can then be relaxed and if this relaxation lead to a successful interaction it can be stored as a construction variant.

7. Extension: The extension of a given construction to create a new variant with an additional unit.

There are two types of linguistic interactions: formulation by the tutor and comprehension by the listener. When this involves learning, the same utterance is comprehended again to see whether the constructions were learned and it is also produced to test that the learner is now capable to produce the utterance again from the same meaning. The web demonstration shows for each interaction: (1) Production of an utterance by the tutor. (2) Comprehension of the utterance the learner, if the learner could not handle the utterance then there is a learning process. (3) (i) input (the meaning in case of production, the situation model and the utterance in case of comprehension), (ii) the processing including possible diagnostics and repairs, and (iii) consolidations taking place after repairs.

7.3.1 Routine processing of a clause

The first example on the accompanying web-demo illustrates routine processing: The tutor produces the utterance “Paul sees (the) ball” from a representation of its meaning and then the learner parses the same utterance back to reconstruct the meaning. Through the web interface we see first the meaning, then the construction-set being used (which can be clicked to see the list and the details of each construction) and then the
different steps in processing.

We look at formulation first. The speaker gets a model of the situation which he attempts to describe completely.

- The *meaning* to be expressed (which is in these experiments equal to the situation model) is shown first. It is a semantic network with the cognitive action see (r-1) and two arguments: arg-1 is a person Paul (o-3) and arg-2 is a physical object table (o-3). The links show the reoccurrence of the same entity as argument of different predications.

- Then the *construction inventory* is shown, which can be clicked on to show the contents.

- The *initial structure* contains the initial transient structure. It can be seen in full by clicking ‘expand’. The transient structure contains only the root unit with two features: for the meaning to be expressed and for the topic, which is r-1.

- The *process steps* show the different steps in processing. The final node is in darker green. Each box shows one or more construction applications. If there is more than one application, clicking on the box will separate them out. One click at the box shows the transient structure at that point and clicking again on the header of the box shows more details: the transient structure before construction application, the construction itself, and then the transient structure after construction application. Each of these can be expanded further so that details become visible. We are in production mode so the production lock of the construction has to match and if so the comprehension lock and the contributing part have to merge. The binding-lists that are constructed in matching and merging are shown as well. Clicking again on the box header will collapse the box.

- The *final structure* gives the final transient structure.

- *Utterance* shows first the utterance and *structure* the same utterance with parentheses, indicating the phrasal boundaries.

Next is an example of comprehension. The listener gets an input phrase and the situation model shared with the speaker.
• The utterance to be comprehended, ”paul sees table” is shown first. The situation model shows the situation model as a network.

• The initial structure contains the initial transient structure which contains the root unit with one feature called form that contains the different strings and which string meets (i.e. directly precedes) another string.

• The process steps shows the different steps in processing, as before but now all constructions are applied by matching the comprehension lock first.

• The final structure gives the final transient structure.

• Meaning shows the meaning network derived from the final transient structure.

• Interpretation shows how the bindings in matching this meaning network against the situation model.

• Interpreted Topic shows the topic that could be derived.

7.3.2 Routine processing of modifiers

The second illustrates still routine processing, but now for a noun-phrase with an extensional modifier. An extensial modifier conveys more meaning that further restricts the referent introduced by the head noun. It can be in the form of prepositional noun-phrases (the block on the table), participles (the block seen by Paul), or relative clauses (the block that Paul saw). The present examples concerns the latter with a phrase like ‘(the) table (that) Paul sees’. We see the same items through the web interface.

7.3.3 Build-hierarchy: unary relation

Now we look at the first example of learning. It is a very simple example to show the basic idea of the proposed architecture, based on diagnostics, repairs, and consolidation. The example is for learning the first level of the hierarchy, in this case a noun-phrase containing just a single noun. The noun is “table” which is first produced by the tutor. The meaning of this noun is based on a predicate ‘physobj(table, ?o1)’ and introduces the
referent ?o1. The learner will have to build a construction that introduces the equivalent of a noun-phrase with only this noun as constituent.

First the demo shows how the tutor produces the utterance “table”. Then we can see the learner. You should look at the processing steps of the learner. The final step is a repair-event and its internals can be inspected by clicking twice on it (clicking only once shows the transient structure after the repair)

*Diagnostic:* When all constructions have been applied, the listener evokes a diagnostic called *any-dangling-relations?* which detects whether there are arguments of a relation which are not linked to the relation itself. This diagnostic also triggers in the null-case when the relation is just used to refer to a single argument, without any other arguments expressed. This is the case here and the diagnostic triggers for the table-unit.

*Repair:* If the *any-dangling-relations?* diagnostic triggers, a *Build-Hierarchy* repair event takes place, which consults the situation model and creates a new unit grouping all elements of the relation.

*Consolidation:* The consolidation consists in building a new construction that groups the different expressions that introduce the relation and its arguments, including the null-case where there is only one expression which introduces a unary relation and has this single argument as referent. For each of the arguments in the relation a slot is created with a particular syntactic category, which is a new category if no categories were associated with the filler yet. In this case, there is a new syntactic category for the “table”. It functions like “noun” in the tutor’s grammar. The different constraints coming from the tutor’s utterance, in particular the ordering constraints, are absorbed in the new construction. Consolidation takes place after all possible repairs have been carried out and the utterance could be treated completely by the learner. In the present case, the learner acquires a construction with a single-slot identifier, more concretely one of the kind: `((physobj syn-cat-i 0 +))`, where i is the number generated for this new category.

After learning, there is a test whether the learner acquired the right constructions, in other words whether routine processing can now handle the input utterance, but also whether the learner is now able to produce himself the utterance, given the same meaning as the tutor originally
expressed. These tests can also be inspected in the demo, and they are successful.

### 7.3.4 Build-hierarchy: Build Hierarchy Relational

We now look at a similar example, based on the word “sees”. This word introduces a predicate with two arguments (for the seeer and the seen). We also introduce ambiguity to make the example more interesting. “sees” can both act as the main verb “sees” but also as the particle “seen-by”. The first case is then where ‘sees’ is the main verb in a clause (as in ‘paul sees (the) table’) and the second case is where ‘sees’ is acting like a participle, (as in ‘(the) table (that is) sees (actually seen by) Paul’).

There are different word orders to distinguish between the same uses of the word. In the first case, the topic of the whole meaning is the relation itself, in the second case it is one of the arguments of the relation, namely the table.

In the first example, the web-demo shows first the formulation process by the tutor. Notice in the processing-steps that there are two hypotheses now, one where the 0th argument of the see-event, namely the see-event itself, is the referent of the transitive verb-phrase, and a second one where the second argument, namely the table, is the see-event. However here the topic of the tutor is the see-event (r-1) so only that branch in the search tree leads to a successful utterance.

Now let us look at the learner. He is parsing the utterance and arrives at a point (the final step) where there are three possible repair actions: (i) where the referent is the see-event, (ii) where the referent is the first argument (paul), (iii) where the referent is the second argument (the table). The learner at this point does not know the topic yet. So he picks one at random. If this is the topic that the listener confirms is the one he had in mind, then there is a consolidation event for the successful repair. Let us inspect this learning event.

**Diagnostic** The diagnostic is the same as for the previous example, namely *any-dangling-relations?*. This diagnostic indeed triggers for the unit based on the word ‘sees’ because the relation between the predicate ‘see’ and its arguments is not expressed, also the referent is unknown.
**Repair** A group-unit is introduced and all components are incorporated in that unit. It is in fact the same repair action as in the previous example, but now encompassing more units. Notice the tight interaction here between semantics (which tells the learner which objects are hanging together) and the introduction of syntax.

**Consolidation** The consolidation is also the same as in the previous case. We see that sees is assigned a new syntactic category and that a phrasal construction is built.

The next interactions show that the listener is now able to comprehend and produce the same utterance without having to acquire a new construction. It is of particular importance to observe that production was learned at the same time as comprehension.

### 7.3.5 Lex-cat-coercion: Coercion of lexical category

The next example shows how lex-cat-coercion takes place. The inventory of constructions of the learner now contains all previous ones, plus a lexical construction for the verb ‘wants’. This construction has no potential lexical categories yet. The example begins with the speaker producing the utterance “Paul wants table”. Then the listener comes in action. You should click on the third box in processing steps to reveal the repair-event. Clicking twice on this repair-event header shows the details.

**Diagnostic** The any-dangling-relation? again triggers and detects that the argument structure introduced through the word ‘wants’ are not yet expressed.

**Repair** But a construction is found that almost matches, namely:

The construction does not match because the the semantic category of

| sem-cat: | person | cog-action(0,1,2) | physobj |
| arguments: | 1 | 0 | 2 |
| syn-cat: | NP | transitive-verb | NP |
| | cog-action | transitive-verb-phrase |

‘want’ is a cognitive action (and the current transitive verb construction expects a physical action) and it does not have the required potential...
syntactic category (the equivalent of transitive-verb). But if the wants-unit is coerced to include transitive-verb, then the construction would match. So a repair event changes the transient structure to insert this change. Then processing continues and the transitive-verb-phrase construction can apply.

**Consolidation** Consolidation requires only that the lexical construction for ‘want’ has now an additional potential syntactic category namely the category required to use the phrasal construction.

### 7.3.6 Ordering-variation: Learning a construction variant with a new ordering

The next example shows how a variant of a construction can be learned by copying the existing construction but changing some of its constraints. This new construction is then competing with the existing one. The example concerns a learner that has acquired the SVO ordering of constituents, as in ‘Paul wants (the) table’. But now the learner gets another sentence, ‘Paul (the) table wants’, which shows an SOV ordering. Perhaps there is a change going on in the language or there is a dialectal variation.

The web demo shows first that the tutor produces the SOV ordering. The learner is unable to parse this utterance and triggers a meta-level process (last box in the processing steps).

**Diagnostics** The *any-dangling-relations?* triggers again because the argument-structures of the see-event introduced by ‘sees’ is not expressed.

**Repair** However now there is a construction that is partially matching. The ordering constraints on the constituents in the transitive-verb-phrase construction are relaxed (by introducing new variables for the meets-predications on the transitive-verb-unit) and then a match can be found that takes into account all other constraints. Processing can continue after this fix and now the argument structure is completely expressed.

**Consolidation** Repair takes place by building a new construction that is entirely equal to the previous one that was matching partially, but now a new ordering is included.
7.3.7 Extension: Learning a construction variant by incorporating a new unit

The last example is a special case of hierarchy building that was already illustrated in the earlier examples (7.3.3. and 7.3.4.). Here the tutor produces the utterance “table block”, and both words refer to the same object. The learner has already a construction that can build a single noun phrase over “table” (from the example in 7.3.3) and “block” should in principle be treated with the same construction. However, given that there is already a corresponding noun phrase, the solution provided by the Build-Hierarchy repair and consolidation consists in extending the single noun phrase to a combined noun phrase, i.e., a noun phrase with two constituents at the same level.