Notice

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

Ciortuz Liviu, Saveluc Vlad (2012). Fluid Construction Grammar and Feature Constraint Logics. In Luc Steels (Ed.), *Computational Issues in Fluid Construction Grammar*, 289–311. Berlin: Springer.

BibTeX:

```
@incollection{ciortuz2012fcg,
   Author = {Ciortuz, Liviu and Saveluc, Vlad},
   Title = {Fluid Construction Grammar and Feature Constraint Logics},
   Pages = {289--311},
   Editor = {Steels, Luc},
   Booktitle = {Computational Issues in {Fluid Construction Grammar}},
   Publisher = {Springer},
   Series = {Lecture Notes in Computer Science},
   Volume = {7249},
   Address = {Berlin},
   Year = {2012}}
```

Fluid Construction Grammar and Feature Constraint Logics

Liviu Ciortuz and Vlad Saveluc

Department of Computer Science, "Al.I. Cuza" University, Iaşi, Romania

Abstract. Fluid Construction Grammars (FCGs) are a flavor of Construction Grammars, which are themselves unification-based grammars. The FCG syntax is similar to that of other unification-based grammars only to a small extent. Additionally, up until now, FCG has lacked a comprehensively-defined declarative semantics, whereas its procedural semantics is truly particular compared to other unification-based grammar formalisms.
Here we propose the re-definition of a core subset of the FCG formalism (henceforth called FCG light) within the framework of order-sorted feature constraint logics (OSF-logic) that would assign FCG a rigorous semantics, both declarative and procedural, that is suitable for both parsing, production and grammar learning.
This new framework allows us to clearly compare FCG to other unification-based grammars. We will also have the advantage of associating FCG with another classical paradigm for learning ("evolving") new grammars, namely learning in hierarchies (lattices) of concepts. This learning technique exploits the natural partial order relation of generalization/ specialization between grammars. The learning method currently used by FCG, is (inspired by) reinforcement learning. We claim that learning in a hierarchy of grammar versions enables us to establish a rather natural link with linguistic background knowledge when devising the grammar repair strategies. It also sets a stage on which we may compare different grammars that could be learned by an agent at each step during the grammar evolution process.

1 Introduction

Here we present FCG light, a core subset of Fluid Construction Grammars introduced by [33] [15] [34] which is currently implemented on a simplified version of the LIGHT platform [9].

The LIGHT system was developed with the explicit aim of doing efficient processing of HPSG-like unification grammars[25], but it is by no means restricted to working with HPSGs.[1] LIGHT comprises several feature structure (FS) unifiers[2]

[1] HPSG stands for Head-driven Phrase Structure Grammars.

[2] More exactly, LIGHT currently has 6 unifiers, of both typed and un-typed kinds: *a.* non-compiled (lazy) typed unification, *b.* compiled (eager) typed unification, *c.* compiled (eager) typed unification specialized for active, bottom-up, chart-based parsing and their un-typed counterparts (a', b', c'). For technical details, the interested reader should consult [12].

and a control level (actually performing active bottom-up chart-based parsing) upon the unification level. An important number of optimizations were incorporated into the LIGHT system. In the past these optimizations were fine-tuned so as to achieve efficient parsing with ERG [17], the large-scale HPSG grammar for English developed at CSLI, Stanford University.

The present work leads to defining FCG light as a new flavor of unification grammars, which is derived from the FCG formalism and is transposed into the LIGHT setup, thus benefitting from a rigorous semantics (both a declarative and a procedural one) based on OSF-logic, which was introduced in [3] and [1].[3]

We argue that by using this logic-based semantics we are able to:

- redefine the procedural semantics of the chosen subset of FCG and associate it with a declarative semantics which we claim is not clearly visible in the FCG setup;
- gain certain benefits through this framework arising from the natural partial order relation between grammars defined via generalization/specialization;
- replace grammar learning, as a consequence of the above issue, in FCG (which is based on the reinforcement learning paradigm) through learning in a lattice of grammars. We argue that the latter paradigm is more naturally suitable for integrating linguistic background knowledge, and it leads to more efficient learning because it enables us to define certain *heuristics* that are very helpful for guiding the search for appropriate (rule) candidates in the grammar lattice;
- in certain conditions,[4] define parsing and production in FCG light in a declarative manner (as it is the case of unification grammars, in particular head-driven grammars).

This paper is organized as follows: Section 2 makes a review of the (core) FCG formalism from the feature constraint point of view. Section 3 goes through the details of FCG light language's definition, at both syntactic and semantic levels. Section 4 is concerned with grammar learning aspects in FCG light. Section 5 scrutinizes several tasks that we have planned, demonstrating further usefulness of FCG light.

[3] OSF-logic is closely related to Carpenter's logic of typed feature structures [5]. It has been associated with an abstract machine for compilation of FS unification [2], which was further extended by [10] to compiled typed FS unification. OSF is not concerned with appropriateness constraints; however, we have shown that in certain conditions one could automatically infer these constraints from the given input grammar [7]. For a good introduction to feature constraint logics, the reader should consult [31].

[4] When constraint reduction actions (see Section 3.1) are performed in "soft" mode, they do not affect logical entailment. This is why in FCG light the reduction operation is replaced (at unification level) by constraint marking for deletion (to be cared of by the parser/producer).

```
((?top-unit
  (tag ?meaning (meaning (== (read ?event)
                             (reader ?event ?agent)))))
((J ?verb-unit ?top-unit)
 ?meaning
 (referent ?event)
 (sem-cat (==1 (base-type ?event event)))))
<-->
((?top-unit
  (tag ?form (form (== (string ?verb-unit "cita")))))
 ((J ?verb-unit ?top-unit)
  ?form
  (syn-cat (==1 (pos verb)
           (gender ?agent ?agent-gender)
           (case ?object accusative)))))
```

Fig. 1. An FCG construction that acts as lexical entry for the Russian verb "cita", similar to the verb "risova" presented by [18], Chapter 3.

2 FCG revisited: A feature constraint-based perspective

Here we summarize the basic notions in the FCG formalism, relative to both its syntax and procedural semantics.

An FCG grammar is a set of structures called *constructions*, which are written in the FCG format, as exemplified in Figure 1. Unlike FCG authors [32] [14], here we give a *constraint*-based view on the definition of construction structures. To this aim, we need some basic notions of feature constraint logics. We briefly describe them here, yet without going into formal details. *Elementary constraints* considered here are of three kinds: sort constraints, feature constraints and equality constraints.[5]

It is useful to make the following *preliminary remark:*

In HPSG/LIGHT rules are expressed simply as FSs, enabling the user to employ a unique formalism for both the grammar rules and the structures to which they apply. However, in FCG formalisms, the *constructions* that express rules differ from the structures to which they are applied, namely the *coupled feature structures*. This *difference* is due to the fact that certain operations — beyond the level of deductive parsing and production — that have to be performed by the parser/producer are specified in the constructions representing rules.

[5] In FCG light, like in other formalisms, the equality constraints are not explicitly used at the syntax level. Instead they are automatically derived while building certain variable substitutions, i.e. during operations manipulating the FSs: subsumption, unification (computation of the GLB for two FSs), and generalization (LUB computation).

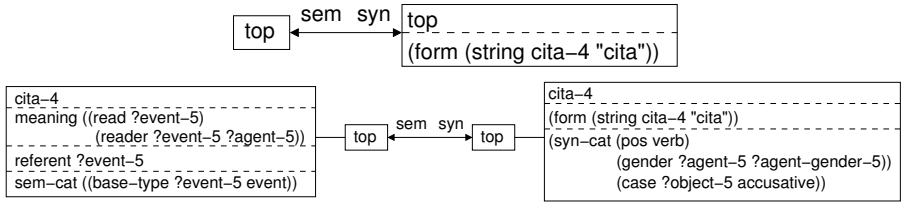

Fig. 2. Two simple couple feature structures. The lower CFS is obtained from the upper one by application of the construction given in Figure 1 in parsing mode.

This fact unfortunately leads to the disruption in FCG of the really nice correspondence between declarative and procedural semantics and also the orthogonality between the unifier level and the parser/producer level that characterize main-stream unification grammars, in particular HPSGs. This disruption seems to be the price to be paid by FCG for the benefit of offering the grammar writer the capacity to play with subtleties in the learning (evolvable) grammars.

Similar to HPSG/LIGHT, in defining FCG light we aim to reduce to a minimum (and, in certain conditions, even eliminate) the difference between the form of rules and the CFSs to which they apply, thus enabling our parser/producer to work very smoothly.

At this point, we can provide a set of informal *definitions* that can be seen as constraint-based alternatives to the ones that have been introduced in [34]:

A *coupled feature structure* (CFS) can be seen as a set of elementary constraints, partitioned into two disjoint subsets named *poles*. These poles are usually referred to as the *syntactic* pole and the *semantic* pole (more generally: the *left* pole and the *right* pole) of the given CFS. Each of the two poles further partitions its set of elementary constraints into several *units*. The units in a CFS are explicitly linked into a graph (usually a tree) using the multi-valued *features* `syn-subunits` (in the syntactic pole) and `sem-subunits` (in the semantic pole). Further on, each unit partitions its set of elementary constraints under several *slots*.

To illustrate the above notions, consider the CFS shown in the lower part of Figure 2, in which the `top-unit` has as sub-unit the `verb-unit`, and examples of slots in the latter unit are `syn-cat` and `sem-cat`.[6] The FCG constraints `(read ?event-5)` and `(base-type ?event-5 event)` correspond in OSF-logic to the elementary sort constraint #*event-5:read* and respectively the feature constraint #*event-5.base-type* $\Rightarrow$ *event*.

[6] The reader will see that this CFS can be obtained from the very simple CFS shown in the upper part of Figure 2 by the application of the FCG construction in Figure 1 in parsing mode.

CFSs and constructions in FCG are characterized by a certain, de facto user-specified correspondence between syntax and semantics. This correspondence starts with the `meaning` and `form` constraints in a top-unit and is down-propagated to the other units via parsing and production.

Basically, the set of elementary constraints that constitute a CFS can be treated in either *match*/subsumption mode or *merge*/unification mode. The actual way in which the elementary constraints are processed is dependent on the parsing or production process (not detailed here) carried by the application of construction rules [14] [4].

The notion of *construction* extends the definition given above for CFS by adding the following:

- two *operators*, namely:
  the *J operator* that (indirectly) indicates `syn`/`sem-subunits` constraints and also separates the merge zone from the match zone in a pole;
  the *tag operator* that indicates a substructure (set of elementary constraints) to be deleted and eventually moved elsewhere;
- several *restrictors* ('==1', '==0', etc.), which can be seen as meta-constraints to be checked while the match and merge operations are performed. For further details, the reader should consult [34].

In FCG, the scope of the match and merge operations is limited to slots. In FCG light we replace slots with features, whose values are implicitly $\top$-sorted, where $\top$ is the top sort in the sort hierarchy.[7] In FCG light the constraints associated with a slot must represent a rooted feature structure.

Similar to the LIGHT system, in FCG light we make no use of negation ((0==) in FCG), whereas the single-valued restriction ((1==) in FCG) is considered implicit for feature constraints.[8] Multi-valued features (designated using the =>> symbol) are restricted to SYN-SUBUNITS and SEM-SUBUNITS.[9] The treatment of these two features is reserved for the parser and the producer. In FCG light, features names are always capitalized.

Two additional features SYN and SEM, corresponding to the two poles in a CFS are introduced.[10] A syntactico-semantic graph, henceforth abbreviated as *syn-sem graph*, can be derived from each CFS. The notion of syn-sem graph is used in the sequel as an alternative/replacement for the notion of CFS.

From the restrictions and syntactic transformations listed above, it follows that a CFS in FCG can be naturally written as a FS in OSF/LIGHT.

[7] For these slot-derived features, appropriateness constraints [5] with corresponding new sort values can be further added.

[8] Therefore, the symbol $\Rightarrow$ in OSF functional constraints is dropped.

[9] Multi-valued features are found, for instance, in F-logic [21]. OSF-logic can be naturally extended so as to accommodate such features.

[10] These features are not (necessarily) shown in the sequel, if the sets of syntactic slots and semantic slots are disjoint.

production

precond.	♯*top-unit.MEANING* = ♯*event*, ♯*event:read*, ♯*event.READER* = ♯*agent*
reduction	♯*top-unit.MEANING* = ♯*event*, ♯*event:read*, ♯*event.READER* = ♯*agent*
main	♯*top-unit.SEM-SUBUNITS* ∋ ♯*verb-unit*, ♯*verb-unit.MEANING*=♯*event*, ♯*event:read*, ♯*event.READER* = ♯*agent*, ♯*verb-unit.REFERENT* = ♯*event*, ♯*verb-unit.SEM-CAT* = ♯*1*, ♯*1.BASE-TYPE*(♯*event*) = ♯*2*, ♯*2:event*

parsing

precond.	♯*top-unit.FORM* ∋ ♯*verb-unit*, ♯*verb-unit.STRING* = ♯*1*, ♯*1:"cita"*
reduction	♯*top-unit.FORM* ∋ ♯*verb-unit*, ♯*verb-unit.STRING* = ♯*1*, ♯*1:"cita"*
main	♯*top-unit.SYN-SUBUNITS* ∋ ♯*verb-unit*, ♯*verb-unit.STRING* = ♯*1*, ♯*1:"cita"*, ♯*verb-unit.SYN-CAT* = ♯*2*, ♯*2:verb*, *verb:pos*, ♯*2.GENDER*(♯*agent*) = ♯*agent-gender*, ♯*2.CASE*(♯*object*) = ♯*3*, ♯*3:accusative*

Fig. 3. The sets of elementary constraints used in production and respectively parsing with the "cita" lexical entry given in Figure 1. The hash symbol (#) introduces variables, while the colon (:) precedes the sort of a variable (or a supersort of a sort). The symbols = and ∋ designate values for single-valued features and respectively multi-valued features. Feature names are written in upper case.

3 FCG light language definition

Here we formally introduce in Subsection 3.1 the syntax of the FCG light subset of FCG, basically showing how constructions in FCG get translated into the OSF/LIGHT syntax (which in the past also supported HPSG grammars). Then, in Subsection 3.2, we introduce the two basic aspects of FCG light semantics — the declarative one and the procedural one —, that further support the FS subsumption and FS unification operations used in parsing, production and grammar learning.

3.1 FCG light syntax

Now we will (re)define for FCG light the notion of construction by getting it as close as possible to the notion of FS in OSF/LIGHT. (Re)defining the notion of construction for FCG light requires getting it as close as possible to the notion of FS in OSF/LIGHT.

production

precond.	`#top-unit[ MEANING =>> #event:read` `[ READER #agent ] ]`

reduction	`#top-unit[ MEANING =>> #event:read`
	`[ READER #agent[ TO-BE-REDUCED + ],`
	`TO-BE-REDUCED + ] ]`

main	`#top-unit`
	`[ SEM-SUBUNITS =>> #verb-unit`
	`[ MEANING #event:read`
	`[ READER #agent ],`
	`REFERENT #event,`
	`SEM-CAT top`
	`[ BASE-TYPE( #event ) event ] ] ]`

parsing

precond.	`#top-unit[ FORM =>> #verb-unit[ STRING "cita" ] ]`

reduction	`#top-unit[ FORM =>> #verb-unit[ STRING "cita"`
	`[ TO-BE-REDUCED + ],`
	`TO-BE-REDUCED + ] ]`

main	`#top-unit`
	`[ SYN-SUBUNITS =>> #verb-unit`
	`[ STRING "cita",`
	`SYN-CAT verb:pos`
	`[ GENDER( #agent ) #agent-gender,`
	`CASE( #object ) accusative ] ] ]`

Fig. 4. The OSF rooted terms (FSs) corresponding to the sets of elementary constraints identified for the "cita" lexical entry (Figure 1), that have been shown in Figure 3. Compared to Figure 3, here above we did not show OSF variables that occur only once. Also, multi-valued feature constraints corresponding to the J operator were added; see the `SYN-SUBUNITS` and `SEM-SUBUNITS` features.

Definition: In FCG light, a *construction* is a set of elementary (i.e. atomic) constraints which is divided into the following two (not necessarily disjoint) triplets of sets:

- a set of *precondition* constraints, a set of constraints marked for *reduction* actions and the set of *main* constraints used for parsing,
- a set of *precondition* constraints, a set of constraints marked for *reduction* actions and the set of *main* constraints used for production.

In FCG light we explicitly impose the following *restrictions:*[11] For both parsing and production, the constraints to be reduced must constitute a subset of the

[11] These demands are generally met by FCG grammar writers.

precondition set of constraints, and the set of main constraints must be disjoint from the precondition set.

In order to give an *exemplification* of the above definition of construction in FCG light, the sets of elementary constraints that build up the "cita" construction — which was given in FCG format in Figure 1 — are presented in Figure 3. Further on, Figure 4 shows these sets of constraints represented as rooted FSs in OSF/LIGHT format.[12]

In many cases, it is possible to actually get rid of constraint reduction, which is why in the current implementation of FCG light we opted for a "soft" treatment of reduction. In other words, the set of constraints designated for reduction are *marked* at unification and/or subsumption level by using the reserved feature TO-BE-REDUCED that takes boolean values. The parser and the producer subsequently analyze these markings. Such treatment in FCG light is absolutely sufficient for reproducing a quite elaborate example of learning in FCG, such as the one described in Gerasymova's MS thesis [18].

Instead of having one construction/form treated in two different ways during parsing and respectively production (as it is done in FCG), in FCG light we explicitly associate each construction with two rules that are treated in exactly the same manner — the subsumption, reduction and unification sequence — in both parsing and production. The *general form* of a parsing or production rule that corresponds to an FCG light construction is

$$\mu : -\psi; \alpha.$$

where μ is the main set of constraints, ψ is the precondition, and α is a set of to-be-reduced constrains. Here, μ, ψ and α should be seen as rooted FSs.

If the reduction actions are implemented in soft mode, and α' is the marked FS (using the TO-BE-REDUCED feature) that corresponds to α, then the rule $\mu : -\psi, \alpha$ becomes $\mu, \alpha' : -\psi$.

The latter can be even written as

$$\mu' : -\psi.$$

where μ' is the unification result for μ and α'.

The *algorithm* responsible for getting these two FCG light rules is presented in Figure 6. It has three main steps, each step translating/transforming in a certain way the output of its preceding step. Before we will comment on them, we show via an *example* what these steps are supposed to do. When applied on the "cita" construction given in Figure 1 the 'initial' translation step in this algorithm builds the sets of elementary constraints shown in Figure 3. In the 'intermediate' translation step, these sets of constraints are put under the form of (single-)rooted FSs, as shown in Figure 4. These FSs will be subject to a number of simple operations in the 'final' translation step, and their ultimate form (for this example) is given in Figure 5.

The idea behind the first part of the 'initial' translation step of our FCG-into-LIGHT algorithm is the following: Consider an arbitrary FCG construct

[12] The correspondence between sets of elementary constraints and (multi-rooted) feature structures should be familiar to the reader acquainted with feature constraints logics.

```
/* cita ; production */
#top-unit
[ SEM-SUBUNITS =>> #verb-unit
                   [ REFERENT #event,
                     MEANING #event:read
                             [ READER #agent ],
                     SEM-CAT top[ BASE-TYPE < #event, event > ],
                     STRING <! "cita" !>,
                     SYN-CAT verb
                             [ GENDER < #agent, #agent-gender >,
                               CASE < #object, accusative > ] ],
  SYN-SUBUNITS =>> #verb-unit,
  ARGS < #top-unit[ MEANING #event:read
                           [ READER #agent ] ] > ]

/* cita ; parsing */
#top-unit
[ SYN-SUBUNITS =>> #verb-unit
                   [ SYN-CAT verb
                             [ GENDER < #agent, #agent-gender >,
                               CASE < #object, accusative > ],
                     REFERENT #event,
                     MEANING #event:read
                             [ READER #agent ],
                     SEM-CAT top[ BASE-TYPE < #event, event > ] ],
  SEM-SUBUNITS =>> #verb-unit,
  ARGS < #top-unit[ FORM =>> #verb-unit[ STRING <! "cita" !> ] ] > ]
```

Fig. 5. The two FCG light rules that are associated to the construction "cita" given in Figure 1. The `ARGS` feature designates the right hand side (RHS) of the rule. The syntax <!!! > is a "sugar-ed" notation for difference lists. Using such a special structure is a very convenient way to replace the (interpretable) constraint `meets` used in FCG.

and assume that we want to obtain an OSF/LIGHT rule that corresponds to the application of this construct in parsing mode. Among all the elementary constraints in which this construction is decomposed, those which are subject to (FCG) matching will be placed in the *precondition* part of the to-be-created LIGHT rule for parsing. Similarly, the constraints used for (FCG) merging will be put into the rule's *main* part. The (FCG) tag-ed constraints will be placed firstly into the rule's *reduction* part and secondly wherever the tag re-appears. The remaining part of the 'initial' translation step is concerned with building the two (parsing and production) rules out of the sets of elementary constraints that have just been built. As formalized above, each rule is of the form RHS :− LHS.

Input: a construction given in FCG format
1. The *initial* translation step
   - by following the guidelines for construction application in FCG,
     build the sets of elementary OSF constraints corresponding to
     precondition, *reduction* and *main* body,
     for parsing and respectively production
   - then, for parsing do the following:
     place the parsing *precondition* and *reduction* constraints
     into the RHS of the newly to-be-created (parsing) rule
     the rest, i.e. the constraints corresponding to the parser's J actions, and
     all stuff in the left/semantic pole,
     including the producer's J constraints
     but not its *reduction* constraints
     is placed into the LHS part of the new (parsing) rule
   - for production: proceed similarly.
2. The *intermediate* translation step:
   for each rule of the two rules resulted from the 'initial' step,
   - in the LHS unify the FSs having the same root identifier
   - check whether the FSs in the *precondition*
     is a connex tree, i.e. single-rooted FS
   - do the same for the LHS part
   - do the same for the *reduction* part if necessary.
3. The *final* translation step:
   for each rule of the two rules resulted from the 'intermediate' step,
   - put *reduction* actions unto soft form
     i.e. mark constraints for reduction, using the TO-BE-REDUCED feature
   - replace the ("interpretable") feature MEETS with difference lists;
     constrain accordingly the variables in the difference lists
   - transform features whose names are non-atomic terms
   - extract sort (s1:s2) declarations
   - use the reserved feature ARGS to designate the rule's RHS.

Output: the two LIGHT rules obtained above.

Fig. 6. The FCG-into-LIGHT translation algorithm which, starting from a construction given in FCG format, obtains the two rules to be used in FCG light for parsing and respectively production. For instance, for the "cita" construction which was given in Figure 1, the two FCG light rules produced by this algorithm are those shown in Figure 5. The sets of elementary constraints (represented as FSs) into which that construction was de-composed (see Step 1 from above), were previously presented in Figure 3. They were further down translated as FSs, and the result of Step 2 was shown in Figure 4.

Something important is to be explained here: The application of the J operator in FCG will correspond in FCG light to (checking and enforcing) certain elementary constraints. These constraints will be expressed using the reserved features SYN-SUBUNITS for parsing, and respectively SEM-SUBUNITS for production. For instance, the FCG code `(J ?verb-unit ?top-unit)` in the left/semantic pole of the "cita" construction given in Figure 1 will be translated as the constraint `?top-unit.SEM-SUBUNITS` ∋`?verb-unit`.[13]

The 'intermediate' step of our algorithm puts sets/conjunctions of constraints under the form of rooted FSs. FCG light, which is oriented toward efficiency of unification and subsumption, imposes that these (*precondition*, *reduction* and *main*) FSs be single-rooted.

The 'final' translation step is concerned with a. soften-ing the constrain reduction, i.e. marking the constraints which must be reduced; b. replacing "interpretable", i.e. procedurally defined FCG predicates like MEETS with non-procedural ones; c. transforming non-atomic feature names, and d. extracting is-a relationships between sorts (unary predicates in FCG).

We think it is useful to show how we transform constraint features identified by non-atomic terms (see Step 3 in Figure 6). For instance, the constraint

```
CASE( #object ) accusative
```

becomes:

```
CASE <#object, accusative>.
```

One final remark: The reader will note that in those two FCG light rules in Figure 5 we purposely omitted the (markings responsible for) constraint reduction. This is due to the fact that in FCG light the parser/producer itself (and not the unifier) takes care of constraint reduction. We should also add that in FCG light constraint reduction is restricted to the scope of `form` and `meaning`.

To summarize this section, we say that a *grammar* in FCG is a set of constructions (written in FCG format), each one of which is automatically translated using the algorithm in Figure 6 into a pair of rules (in OSF/LIGHT format), one for parsing and the other for production.

3.2 FCG light semantics

The OSF-logic provided the *declarative semantics* to the LIGHT system. It does the same for FCG light — if reduction actions are implemented in soft mode — both at the FS unification and FS subsumption levels and at the deductive control level over FSs, i.e. parsing and production [28] [30].[14]

[13] In FCG light, where the use of multi-valued feature constraints is limited to the reserved SYN-/SEM-SUBUNITS features, the parser/producer will fully take care of them. In this way, the unification/subsumption procedure is exempted from this (inefficiency causing!) overhead.

[14] We mention that unlike LIGHT when used for the HPSG ERG grammar, FCG light works with the un-typed counterpart of OSF-logic, since FCG does not impose appropriateness constraints on FSs.

In FCG light all rules are unary rules, unlike the LIGHT system which supports both binary and unary rules. However, the argument — or the pre-condition, or the right hand side (RHS) — of each rule is not necessarily a phrase structure. Instead, it is the description of a rooted subgraph in the syn-sem graph created during parsing and production. The same is true about the rule's left hand side (LHS). More precisely, one of the *restrictions* that we impose on FCG light grammars is the following: all units specified in a construction should constitute a rooted graph, where edges are defined via the SYN-SUBUNITS and SEM-SUBUNITS features.

Concerning the *procedural semantics*, as outlined in Section 3.1, each construction in FCG light is associated with two *rules*, one for parsing and the other for production. Unlike LIGHT, for which the application of rules is fully unification-based, in FCG light the unique argument of a rule is checked for compatibility (with a syn-sem graph) by using FS *subsumption* (match, in the FCG formulation). The rule's LHS is treated via FS *unification*.[15]

In FCG light, the parsing and production can be partial. For parsing, this means that we drop off the usual requests that *i.* all lexical entries should be defined in the given grammar, and *ii.* a syntactic tree should be built so as to span the whole input sentence and to be subsumed by the grammar's start symbol. Just as for HPSG, there is usually no designated start symbol in FCG grammars. The function of such a symbol is taken by the *top-unit*, to which all the other units (morphological, lexical, syntactic or semantic) get linked in one way or another.

Given τ, the syn-sem graph whose root is the *top-unit* and whose arcs are given by the features SYN-SUBUNITS and SEM-SUBUNITS, an FCG light rule of the form $\mu :- \psi, \alpha$ is applied as follows:

> if subsume(τ', ψ), and σ is the corresponding most general substitution,
> then
> perform the reduction of the constraints $\alpha\sigma$ and
> unify($\mu\sigma, \tau''$)

where τ' is an arbitrary (single-rooted, maximally connected) subgraph of τ, and τ'' is the subgraph of $\tau\sigma$ whose root is identified by the root of $\tau'\sigma$.

It should be noted that a rule application has not necessarily a unique outcome, since τ' is not always unique. Here above, the unification operation is seen as a constraint satisfaction problem in OSF-logic. Such a problem is solved using for instance the so-called clause normalization procedure. If $\mu\sigma$ and τ'' are trees, then the usual FS unification algorithm can be used and the result is unique (up to variable renaming). Subsumption is also regarded in the classical way.

The following simple algorithm formalizes the way *parsing* is done in FCG light:

[15] For other perspectives on the unification and merge operations in FCG, the reader may consult [29] and [16].

Input:
a grammar G, and
τ the (initial) syn-sem graph corresponding to
the FORM of a given input sentence.
Procedure:
as long as parsing rules in G apply successfully on τ,
build up the corresponding new syn-sem graph(s).

Concerning the parsing *output:* syn-sem graphs that span the whole input are (eligible for) the output; the user may impose additional constraints on them. If there is no graph that satisfies these constraints, then other (partial) graphs may be considered.

Similarly one can formalize production in FCG light.

4 Grammar learning in FCG light

In FCG, the learning process is reinforcement-based, i.e. each construct receives a weight which is a real number between 0 and 1, and it is increased whenever the construction is used in successful parsing/production. If unsuccessful, the weight is decreased. One of our *objectives* has been to explore in FCG light a different paradigm for defining strategies for construction learning (as used, for instance, in grammar repairing), compared to the paradigm that is currently used by FCG.

In FCG light, learning is based on searching in a lattice of grammar versions, i.e. it amounts to searching in a *version space* which is partially ordered by means of a generalization/specialization (actually subsumption-based) relation between grammars, as illustrated in Figure 7 [22]. The version space is meant to induce a certain discipline while searching for new grammars (and, at lower level, new rules) during the learning process.[16] For a discussion on using lattices for learning in Embodied Construction Grammars (ECG) and comparison with FCG, the reader is referred to [6].

Learning in the FCG light system is performed in *on-line* (i.e. interactive) mode, via language games played by 2 agents, as in FCG. For a schematic view on the functional architecture used by FCG light for learning, the reader should see Figure 8.

In order to be able to go into more detail when explaining the learning strategy used by FCG light, we give in Figure 9 the pseudo-code of the procedure

[16] One could merge the two learning paradigms by associating each rule (of each grammar) in the version space a weight and then updating it, as done in FCG. As a consequence, this new, composed paradigm would be a generalization of the two previous ones. One could think of the rule weights in the current implementation of FCG light as being initially set to 1; removing a rule from a grammar amounts to setting its weight to 0. As for FCG, in the new, compound paradigm one would have to keep track of the generalization/specialization relationships between newly created rules and the previously existing ones, and similarly between different grammar versions.

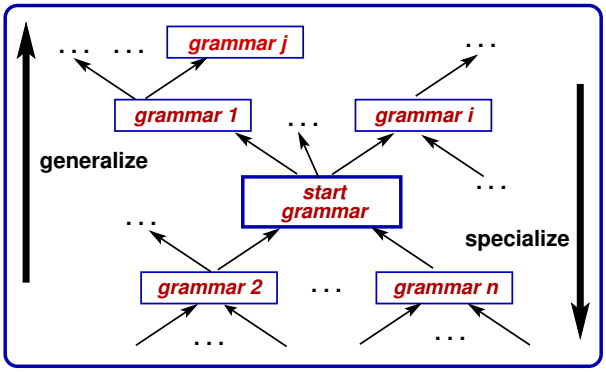

Fig. 7. Illustrating the notion of version space for the process of learning grammars in in the FCG light system. Upward arrows signify the generalization relation between grammars. Conversely we have the specialization relation. During the grammar learning process, the 'start grammar' can be generalized for instance to 'grammar 1' (which in turn can be generalized to 'grammar j'), or can be specialized to 'grammar 2' or 'grammar n'.

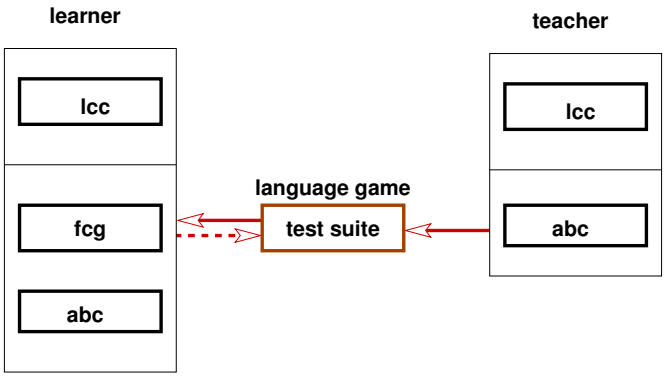

Fig. 8. Schematic view on the learning architecture in FCG light. Here **lcc** (a name which is an abbreviation for: LIGHT into C Compiler) designates the module in the LIGHT system that is in charge with the pre-processing and compilation of the input grammar; **abc** is the parser's name (abbreviation for: Active, Bottom-up Chart-based), while **fcg** is the learner module of FCG light. The dotted arrow corresponds to questions that the learner may ask the teacher in order to guide his or her search for better inferred rules/grammars.

Target grammar: ;
the given given in Chapter 3 of Gerasymova's MS thesis [18];
Input/start grammar: ;
obtained from the target grammar by deleting for instance the lexical construction for "po-" and the associated 'mapping rule' and 'semantic' rule.
Language game: ;

1. choose a *setup*, for inst. Misha read for-a-while; Masha read ongoing;
   teacher: generate a question, for instance "kto pocital?"
   learner:
   parse the question,
   get the corresponding *meaning*,
   try to disambiguate it wrt the given *setup*,
   if disambiguation is successful, then go to Step 1,
   otherwise (since in this case 'po-' is unknown to him/her),
   send to the teacher the *failure* message;
   teacher: reveal the correct answer to the above question ("Misha");
   learner:
   after correlating the CFS previously obtained by parsing
   with the meaning pointed to by the teacher's answer,
   induce a *holophrasis* construction (corresponding to 'pocita');
2. after performing a number of times the Step 1,
   use the procedure given in Figure 10 to
   generalize over the learned holophrases, and then
   extract new construction rules
   by *decomposing* the generalized holophrasis.

Fig. 9. The pseudo-code for the language game strategy which was designed for learning for Russian verb aspects as presented inGerasymova's MS thesis [18], Chapter 4. The teacher agent uses the 'target' grammar, while the learner agent starts the game with an altered version of it, called the 'start' grammar.

that implements in FCG lightthe language game presented by Gerasymova in Chapter 4 of her MS thesis [18]. This language game strategy supports the learning of an FCG grammar for the Russian verb aspects. During the language game played by the two agents, a number of *holophrasis* constructions are learned (see Step 1 in Figure 9). The generalization procedure which is called (see Step 2 in Figure 9) to elaborate new rules based on the previously produced holophrases is given also in pseudo-code in Figure 10.

Gerasymova's *target grammar*, which was translated in FCG light following the guidelines that have been presented in Section 3, is able to parse and produce sentences like "Misha pocital", "Masha narisoval lico", "kto pocital?".[17] In the beginning of the language game, from the target grammar — which is used

[17] These sentences translate into English as "Michael read for a while", "Masha has drown the face", and "Who read for a while?" respectively.

Input: a set of holophrases produced (or, even the CFSs from which they originated) during the application of Step 1 of a series of language games played according to the strategy presented in Figure 9;
Output: a set of pair of constructions (each one made of a lexical entry and a lexical rule) corresponding to each prefix and its associated event-type;
Procedure: • Group (the CFSs that correspond to) all holophrases that have
– the same prefix (for example 'po-'),
– the same value for the EVENT-TYPE feature for the event which is associated to the respective verb's occurrence (for example, 'pocita', 'porisova' etc. correspond to the event-type 'for-a-while');
• for each such group g
generalize: apply the *least upper bound* (LUB) operation on the CFSs (representing the holophrases) in the group g; let's denote it LUB(g);
decompose: split the *generalized holophrasis* construction corresponding to LUB(g) into a *lexical construction* and a *lexical rule*;

to get this done, it is necessary to introduce a new ('AKTIONSART') feature; this feature makes the link between the prefix and the verb's EVENT-TYPE value. For example:

SYN-CAT top[AKTIONSART delimitative];
SEM-CAT top[AKTIONSART(#event) delimitative];
replace g in the current grammar with the above created rules.

Fig. 10. The procedure for generalizing over (CFSs corresponding to) the holophrases learned during a series of language games for acquiring the Russian verb aspects.

as such by the 'teacher' agent —, the syntactic construction 'po-', and the associated 'mapping' rule and 'semantic' rule were eliminated in order to get the *start grammar*, which is to be used (in the beginning of the language game) and later on improved by the 'learner' agent. Skimming through the execution of the procedures given in Figures 9 and 10 during this language game allows us to make a couple of *remarks*:[18]

First, concerning Step 1 in the procedure given in Figure 9:

In FCG light a holophrasis (like 'pocita') is obtained by generalizing the CFSs obtained during parsing. More specifically, this holophrasis construction is obtained by:

– applying the LUB operation on the syn-sem CFSs that have been obtained after parsing slightly similar parsed sentences like "Misha pocital" and "Masha pocitala"; the LUB operation *generalizes* them with respect to the subject (Misha, Masha, kto) and the endings (-l/-la) indicating the perfect tense;[19]

[18] The reader may find useful to follow the explanations below by taking also a look at the whole picture of the learning process, as shown in Figure 13.

[19] [18] does not explicitly call this a generalization. Nor does it explicitly names the LUB operation.

– then *inducing* a rule construction from the generalized (via LUB) CFSs obtained above, this is a construction which when starting from the FORM ('pocita') gets the associated MEANING (read for-a-while) and vice-versa.

Here above it became evident an *advantage* that FCG light has over the FCG approach, due to the fact that we use a feature constraint logics as support: the LUB operation is a well known operation defined on feature structures with a well defined correspondent in feature logics.

Second, concerning Step 2 in the language game strategy outlined in Figure 9: Here we do not stick (strictly) to the 3-step learning scheme used by Gerasymova, inspired by [35], so to produce a syntactic rule, a mapping rule and a semantic rule. We use instead a two-step strategy for building a lexical entry *and* a lexical rule. This is common practice for the HPSGs that have been implemented in the LIGHT system. We mention that the lexical construction and the lexical rule produced in the *decompose* step in Figure 10 correspond respectively to the syntactic rule and (a slightly simplified version of) the composition result of the mapping rule and the semantic rule in [18].[20]

The generalization procedure given in Figure 10 is a significantly revised version of the one in [18]. Concerning its application in the afore mentioned language game, the reader should note that due to the one-step rule decomposition (producing a 'syntactic' construction and a 'combined' rule), it is enough to use the AKTIONSART feature at the verb's SYN-CAT level. Further rule splitting/decomposition of the 'combined' rule is possible, and so a 'mapping' rule and a 'semantic' rule can be obtained (à la Tomasello).

After the work performed by the generalization procedure in Figure 10, a specialization task can be performed on constructions for prefixed verbs, as presented in Gerasymova's MS thesis [18].[21] This task, which consists of two steps, namely A and B in Figure 11, can be performed by alternative means compared to those used in [18]. We claim that in FCG light these means are simpler, more diverse and more naturally fitted into the framework.

Indeed, in OSF-logics it is easy to consider/introduce new sorts by generalizing/grouping some already existing sorts. This is for instance the case of 'non-ongoing' when learning Russian verb aspects, as implemented at Step B in Figure 11 and illustrated in Figure 12. Also, instead of introducing new features — like ASPECT, at Step A in Figure 11 —, one could opt for redefining the sort hierarchy. In our example, we mean introducing the sort 'perfective-verb' as a subsort to the sort 'verb'.

[20] We are not interested here in maintaining the relationship between what we automatically learn in FCG light and the learning schemata discussed in Tomasello's work, as it was done in Gerasymova's thesis for the following reason: while being useful for didactic presentation purposes, we consider it a rather too difficult and complex endeavor to be conveniently followed (as such) by autonomous learning robots.

[21] Alternatively, this specialization can be applied before generalization. In such a case it would work on syn-sem CFSs directly, not on the rules created afterwards based on these CFSs.

A. By simply analyzing the association of *(prefix, event-type)* pairs associated to verbs during the language game, it can be inferred that:

not all verbs can get prefixed (with 'po'-like prepositions), therefore:

- to distinguish between those verbs that accept prefixes and those that don't, introduce a *new sort*, 'perfective', and
- add (it as the value of) a *new feature*, 'ASPECT' at the verb's SYN-CAT level:

  SYN-CAT top[ASPECT perfective];

B. By simply analyzing the values of the EVENT-TYPE feature for events corresponding to the verbs whose ASPECT is 'perfective' it follows that:

only events whose EVENT-TYPE value is different from 'ongoing' are associated (as meaning) to those verbs, therefore:

- invent a *new sort*, name it for instance 'non-ongoing', and
- add (it as the value of) a *new feature*, 'VIEW' at the verb's SEM-CAT level, for all perfective verbs, i.e. those that can be prefixed:

  SEM-CAT top[VIEW(#event) non-ongoing].

Fig. 11. The procedure for specializing over verbs while learning in FCG light the Russian verb aspects. Like in the generalization procedure, our approach is slightly different from the one presented in [18]. A couple of remarks could be added here, namely 1. for step *A*: the complementary sort to 'perfective' would be 'non-perfective', and 2. for step *B*: in the FCG light's sort hierarchy, the 'non-ongoing' sort will be the parent sort for all values taken by the EVENT-TYPE feature which are different from 'ongoing', as graphically illustrated in Figure 12.

After having presented the procedures that support the learning process (Figures 9-11), we add the *remark* that they make explicit a *heuristics* responsible for guiding the learning agent — based on *background* linguistics *knowledge* — to rightly choose a (certain) construction among the (possibly many) different ones which are situated in the lattice of rule versions between the *most specific* and the *most general* constructions that are compatible with the sentences to be learned in the current language game.

For Gerasymova's example, we could be formulate as follows the simple linguistic reason that supports the learning of the construction for the 'po-' lexical entry and the corresponding lexical rule via generalization on the 'pocital'-like holophrases (see Step 2 in Figure 9):

> Because 'po-' is a prefix morpheme inside the word 'pocital', the grammar learning process should concentrate on elaborating the relationship between the newly learned holophrasis and the already existing constructions or CFSs for the other two morphemes that compose that word, namely the verb root ('cita') and the ending ('-l'/'-la').

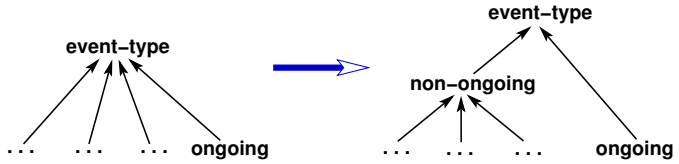

Fig. 12. Example of sort signature refinement in FCG light. Such a refinement can be commanded during the specialization procedure, see Figure 11.

It turns out that this specific *relationship* can be identified by a *heuristics* that generalizes twice on the CFSs corresponding to learned holophrases, namely: firstly generalizing on the ending '-l'/'-la' (by simply using the LUB operation, see Step 1 in Figure 9), and secondly generalizing on the verb (followed by application of the rule decomposition procedure, see Figure 11). The "invention" of holophrasis constructions — starting from CFSs relating morphemes/words unknown to the learner to the meaning that the teacher points to — followed by rule creation is the essence of the grammar learning process during this language game. A synthesis of learning in FCG light the aspect of Russian verbs is shown in Figure 13.

Finally we should note that different language games in FCG and FCG light require different language strategies. Therefore learning in such a setup is not general-purpose. Defining and implementing a set of useful, wider-range learning strategies should be the focus of further research.

5 Conclusion and further work

This chapter introduces FCG light, a core subset of FCG, which is (re)defined using as framework a feature constraint logic, namely OSF-logic. The latter provides FCG light with a well-defined semantics and allows its clear comparison with other unification-based formalisms. We showed how FCG light can be implemented by using a simplified version of LIGHT — a platform on which HPSG grammars have previously been implemented. For further details on the actual implementation, the reader is referred to [27]. In order to proof-check the functionality of our FCG light's implementation we reproduced the experiment for learning the FCG grammar of Russian verb's aspects [18]. Instead of using reinforcement-based learning as done in the current implementation of FCG, we opted for learning in a lattice/hierarchy of different grammar versions. This lattice is naturally provided by the OSF-logic setup by exploiting the specialization/generalization relationship among grammars. Building on this experiment, in our recent paper [13] we have shown how to model in FCG a Slavic-based phenomenon present in a regional dialect of the Romanian language (more exactly, a certain verbal aspect), and how to model in FCG the transformation

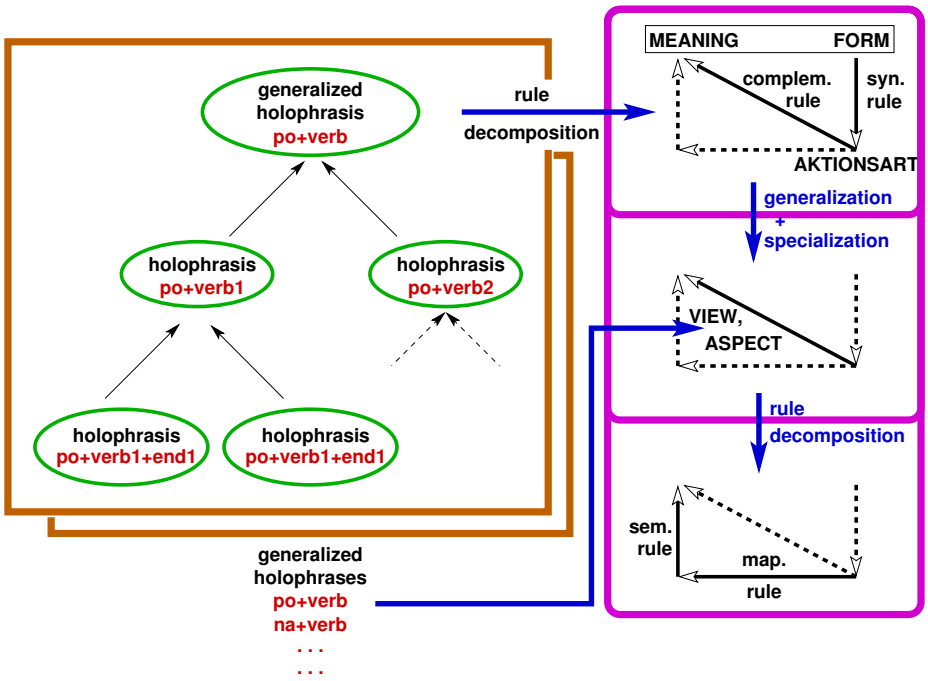

Fig. 13. An overview on grammar repairing, i.e. rule learning in FCG light, as done during the language game for acquiring the Russian verb aspects. For the rule composition scheme (acting on syntactic, mapping and semantic rules) please refer to [19].

that presumably takes place in a child's brain when "learning over" that Slavic construction a Latin-rooted phrasal construction in modern Romanian.

Apart from our experiment and that of Gerasymova's, both using FCG for learning phenomena related to Slavic languages, there is already another one done for Polish [20].

We intend to apply such, and other, learning strategies to learn the clitic pronouns in the Romanian language, which is a rather difficult issue for non-native speakers. The result of the Romanian clitics' formation in FCG light could then be compared, for instance, to the HPSG description of these clitics as done in the Paola Monachesi's PhD thesis [23].

Also, inspired by the FCG approach to grammar learning, we are now able to suggest new ways for learning grammars in other unification-based formalisms. In particular, we aim to test these ideas on ilpLIGHT [8]. This is an extension/configuration of the LIGHT system which adapted the learning paradigm of

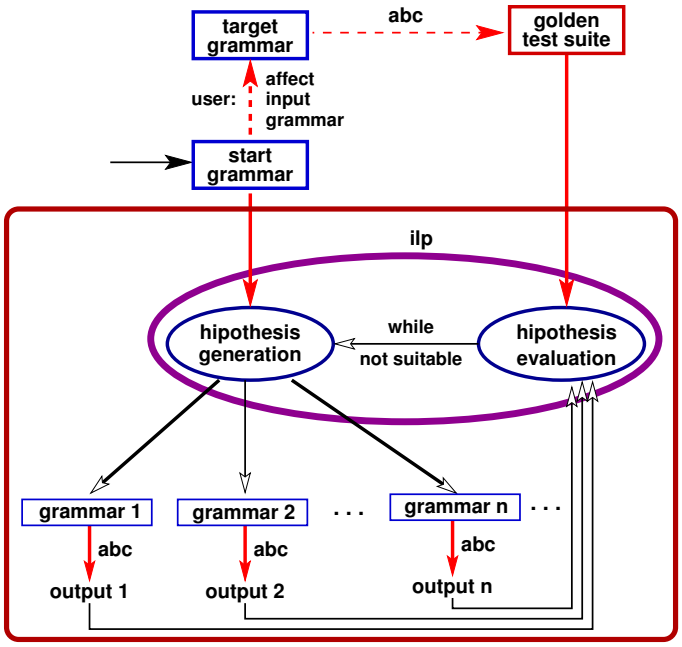

Fig. 14. The learning architecture of the ilpLIGHT extension/configuration of the LIGHT system. The **abc** and **ilp** modules are the parser and respectively the ILP-based learner components of LIGHT.

Inductive Logic Programming (ILP, [24]) so as to work with HPSG-like unification grammars.

In ilpLIGHT, the learning process — also based on searching in a lattice of grammar versions, as in FCG light — is done in *off-line*/batch mode, by using a "golden" test suite given to the learner by the supervisor/teacher.[22] For the learning architecture of ilpLIGHT, the reader is referred to Figure 14. [11] has demonstrated that it is possible to induce each of three basic HPSG principles — the head principle, the subcategorization principle and the saturation principle, as presented by [25] — given that the grammar contains the other two principles and a simply annotated test suite is provided.

We suggest that ilpLIGHT can be substantially improved by using certain ideas borrowed from FCG:

– instead of using a given "golden" test suite (on which parsing is performed and against which the progress of grammar learning is checked), this test

[22] The test suite is a set of sentences with associated parsing trees and eventually other informations.

suite can be dynamically produced during the language game played by two agents;
- the grammar learning process can be "grounded", something which, up to our knowledge, was not considered before for HPSGs;
- new rules can be learned by generalizing upon several already learned rules, instead of simply modifying one or at most two rules, as is currently done in ilpLIGHT, thus constituting a significant step forward.

Upgrading the ilpLIGHT system so to do parsing and production with SBCG grammars [26] would further enhance the possibilities to compare FCG with other construction-based systems.

Acknowledgements: This work has been done in the framework of the European FP7 research project "ALEAR" and its sister project "ALEAR 37EU" funded by the Romanian Ministry of Education and Research". The authors wish to thank Joachim De Beule and Kateryna Gerasymova for their useful comments on an earlier draft of this chapter.

Bibliography

[1] Aït-Kaci, H., Podelski, A., Goldstein, S.: Order-sorted feature theory unification. Journal of Logic, Language and Information 30, 99–124 (1997)
[2] Aït-Kaci, H., Di Cosmo, R.: Compiling order-sorted feature term unification. Tech. rep., Digital Paris Research Laboratory (1993), pRL Technical Note 7
[3] Aït-Kaci, H., Podelski, A.: Towards a meaning of LIFE. Journal of Logic Programming 16, 195–234 (1993)
[4] Bleys, J., Stadler, K., De Beule, J.: Search in linguistic processing. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[5] Carpenter, B.: The Logic of Typed Feature Structures – with applications to unification grammars, logic programs and constraint resolution. Cambridge University Press (1992)
[6] Chang, N., De Beule, J., Micelli, V.: Computational construction grammar: Comparing ECG and FCG. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[7] Ciortuz, L.: Expanding feature-based constraint grammars: Experience on a large-scale HPSG grammar for English. In: Proceedings of the IJCAI 2001 co-located Workshop on Modelling and solving problems with constraints. Seattle, USA (2001)
[8] Ciortuz, L.: A framework for inductive learning of typed-unification grammars. In: Adriaans, P., Fernau, H., van Zaanen, M. (eds.) Grammatical Inference: Algorithms and Applications, LNAI, vol. 2484, pp. 299–301. Springer-Verlag, Berlin, Germany (2002)
[9] Ciortuz, L.: LIGHT — A constraint language and compiler system for typed-unification grammars. In: KI-2002: Advances in Artificial Intelligence. vol. 2479, pp. 3–17. Springer-Verlag, Berlin, Germany (2002)

[10] Ciortuz, L.: LIGHT AM – Another abstract machine for feature structure unification. In: Oepen, S., Flickinger, D., Tsujii, J., Uszkoreit, H. (eds.) Efficiency in Unification-ased Processing, pp. 167–194. CSLI Publications, The Center for the Study of Language and Information, Stanford University (2002)
[11] Ciortuz, L.: Inductive learning of attribute path values in typed-unification grammars. Scientific Annals of the "Al.I. Cuza" University of Iasi, Romania, Computer Science Series pp. 105–125 (2003)
[12] Ciortuz, L.: Parsing with Unification-Based Grammars — The LIGHT Compiler. EditDan Press, Iasi, Romania (2004)
[13] Ciortuz, L., Saveluc, V.: Learning to unlearn in lattices of concepts: A case study in Fluid Construction Grammars. In: Proceedings of SYNASC 2011. pp. 160–167. IEEE Computer Society, Timişoara, Romania (2011)
[14] De Beule, J.: A formal deconstruction of Fluid Construction Grammar. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[15] De Beule, J., Steels, L.: Hierarchy in Fluid Construction Grammar. In: Furbach, U. (ed.) Proceedings of KI-2005. pp. 1–15. No. 3698 in Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin (2005)
[16] Fernando, C.: Fluid Construction Grammar in the Brain. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[17] Flickinger, D., Copestake, A., Sag, I.A.: HPSG analysis of English. In: Wahlster, W. (ed.) Verbmobil: Foundations of Speech-to-Speech Translation, pp. 254–263. Artificial Intelligence, Springer-Verlag (2000)
[18] Gerasymova, K.: Acquisition of aspectual grammar in artificial systems through language games (2009), Humboldt Universitaet zu Berlin, Germany, MS thesis
[19] Gerasymova, K.: Expressing grammatical meaning with morphology: A case study for Russian aspect. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[20] Höfer, S.: Complex declension systems and morphology in Fluid Construction Grammar: A case study of Polish. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[21] Kifer, M., Lausen, G., Wu, J.: A logical foundation of object-oriented and frame-based languages. Journal of the ACM 42(4), 741–843 (1995)
[22] Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
[23] Monachesi, P.: A grammar of Italian clitics. Ph.D. thesis, Tilburg University (1995), iTK Dissertation Series 1995-3 and TILDIL Dissertation Series 1995-3
[24] Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. Journal of Logic Programming 19,20, 629–679 (1994)
[25] Pollard, C., Sag, I.A.: Head-driven Phrase Structure Grammar. CSLI Publications, Stanford (1994)
[26] Sag, I.A.: Sign-Based Construction Grammar: An informal synopsis. In: Boas, H., Sag, I.A. (eds.) Sign-Based Construction Grammar. CSLI Publi-

cations, The Center for the Study of Language and Information, Stanford University (2010), version of August 23, 2010
[27] Saveluc, V., Ciortuz, L.: FCGlight, a system for studying the evolution of natural language. In: Proceedings of SYNASC 2010. pp. 188–193. IEEE Computer Society, Timişoara, Romania (2010)
[28] Shieber, S.M., Schabes, Y., Pereira, F.: Principles and implementation of deductive parsing. Jornal of Logic Programming pp. 3–36 (1995)
[29] Sierra, J.: A logic programming approach to parsing and production in Fluid Construction Grammar. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[30] Sikkel, K.: Parsing Schemata. Springer Verlag (1997)
[31] Smolka, G.: Feature-constraint logics for unification grammars. Journal of Logic Programming 12, 51–87 (1992)
[32] Steels, L.: A first encounter with Fluid Construction Grammar. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[33] Steels, L., de Beule, J.: A (very) brief introduction to Fluid Construction Grammar. In: ScaNaLU '06: Proceedings of the Third Workshop on Scalable Natural Language Understanding. pp. 73–80. Association for Computational Linguistics, Morristown, NJ, USA (2006)
[34] Steels, L., De Beule, J.: Unify and merge in fluid construction grammar. In: Vogt, P., Sugita, Y., Tuci, E., Nehaniv, C.L. (eds.) EELC. Lecture Notes in Computer Science, vol. 4211, pp. 197–223. Springer (2006)
[35] Tomasello, M.: Construction grammar for kids (2007), *Constructions, [Online]*