University "Alexandru Ioan Cuza" of Iaşi
Faculty of Computer Science

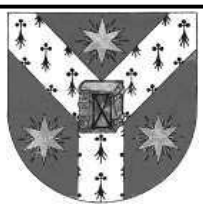
TECHNICAL REPORT

# FCGlight: Making the bridge between Fluid Construction Grammars and main-stream unification grammars by using feature constraint logics

**Liviu Ciortuz**

**TR 10-02,** February 2010

**ISSN 1224-9327**



**Universitatea "Alexandru Ioan Cuza" Iaşi**
**Facultatea de Informatică**
Str. Berthelot 16, 6600-Iaşi, Romania
Tel. +40-32-201090, email: bibl@infoiasi.ro

# FCGlight: Making the bridge between Fluid Construction Grammars and main-stream unification grammars by using feature constraint logics

Liviu Ciortuz

Department of Computer Science
"Al.I. Cuza" University, Iaşi, Romania,
ciortuz@info.uaic.ro

## Abstract

Fluid Construction Grammars (FCGs) are a flavour of Construction Grammars, which themselves are unification-based grammars. Its syntax is (only) up to certain extent similar to other unification-based grammars. However, it lacks a declarative semantics, while its procedural semantics is truly particular, compared to other unification-based grammar formalisms.

Here we propose the re-definition of a subset of the FCG formalism (henceforth called FCGlight) within the framework of (order-sorted) feature constraint logics (OSF-logic) that would assign FCG a rigorous (both declarative a and procedural) semantics, which is suitable for both parsing, generation and grammar learning.

In this way we will be able to compare FCG to other unification-based grammars. We will also get the advantage of associating it another classical paradigm for learning ("evolving") new grammars. This is learning in hierarchies (lattices) of concepts which exploits a partial order relation (generalisation / specialisation) between grammars, instead of the method currently used by FCG, which is (inspired by) reinforcement learning. This will enable a rather natural link with the linguistic background knowledge when devising the grammar repair strategies and a clear way to compare different grammars that could be learned by an agent at each step during the grammar evolution process.

In return, inspired by the FCG approach to grammar learning, we are able to define new ways for learning grammars in other unification-based formalisms. In particular, ILP-like learning of HPSGs can be substantially improved by using ideas borrowed from FCG:
– instead of using a given "golden" test suite (against which the learning is performed), this test suite is dynamically produced during the language game (played by two agents);
– the grammar learning process will be "grounded", something which was not considered before;
– we will learn also new rules (instead of modifying the given ones, as currently done in ilpLIGHT)
– learning new rules by generalising upon already learned rules, will also be a significant step forward.

## 1 Introduction

FCGlight identifies a subset of Fluid Construction Grammars [21] [11] [20], henceforth abbreviated as FCGs, which is suitable for implementation (currently under advanced progress) on the LIGHT platform [6].

The LIGHT system was developed for doing efficient processing of HPSG-like unification grammars, but it is by no means restricted to HPSG. LIGHT comprises two main levels feature structure (FS) unification,[1] and a control level (actually performing active bottom-up chart-based parsing) upon the unification level. An important number of optimisations were included in the LIGHT system; they were fine tuned so as to achieve efficient parsing with LinGO [12], the large-scale HPSG grammar [18] for English developed at CSLI, Stanford University.

ilpLIGHT [5] is an extension of the LIGHT system,[2] which was developed in order to present (a "proof of the concept" for) grammar learning in the OSF-logic framework [2] [3], based on the general idea of Inductive Logic Programming (ILP) [17]. ilpLIGHT has exemplified this approach by showing that it is capable to learn three basic HPSG principles: the Head principle, the Subcategorisation principle and the Saturation principle [8].

On one hand, the *objective* of this work is to open the way for further enhancing the grammar learning capabilities of the LIGHT system by developing the ilpLIGHT configuration into a new version, fcgLIGHT, whose conception is very much inspired by FCG. On the other hand, one could see this work as (leading to) defining FCGlight as a new flavour of unification grammars, which was derived from the FCG formalism and was transposed into the LIGHT setup, thus benefitting from a rigorous semantics (both a declarative and a procedural one) based on Aït-Kaci's OSF-logic.[3]

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

– redefine the procedural semantics of (the chosen subset of) FCG and associate it a declarative semantics which we claim was not clearly visible in the original setup;

– learning in the new framework could benefit from a natural partial order relation, expressed as generalisation/specialisation of grammars;

[1] In fact LIGHT implements several (currently 6) unifiers, of both typed and un-typed kind: *a.* non-compiled (lazy) typed unification, *b.* compiled (eager) typed unification, *c.* compiled (eager) typed unification specialised for active bottom-up chart-based parsing, and their un-typed counterparts $(a', b', c')$. For technical details, the interested reader should consult [9].

[2] Otherwise said: a configuration of the LIGHT system, which is centered around its **ilp** module.

[3] OSF-logic is closely related to Carpenter's logic of typed feature structures [4]. It has been associated with an abstract machine for compilation of FS unification [1], which was further extended by Ciortuz to compiled typed FS unification [7]. 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. For a good introduction to feature constraint logics, the reader should consult [19].

– as a consequence of the above issue, one can replace grammar learning in FCG (which was based on the reinforcement learning paradigm) with learning in a lattice of grammars (further enhanced to the ILP paradigm). We will argue that the latter paradigm leads to more efficient learning, and it is more suitable for integrating linguistic background knowledge. It enables one to define *heuristics*, which are very useful for guiding the search for appropriate (rule) candidates in the grammar lattice;
– in certain conditions,[4] we will be able to define parsing and generation in FCGlight in a declarative manner (as it was for unification grammars, in particular for head-driven grammars); further on we will be able to clarify the relationship between parsing in FCG and classical methods like chart-based parsing.

The remaining part of this paper is organised as follows: Section 2 goes through the details of FCGlight language's definition, at both syntactic and semantic levels. Section 3 is concerned with grammar learning aspects in FCGlight.[5] Section 4 scrutinizes several tasks that we have already made preparations for, demonstrating further usefulness of FCGlight.

## 2 FCGlight language definition

Subsection 2.1 will summarize the basic notions in the FCG formalism, relative to both its syntax and (procedural) semantics. Subsection 2.2 will formally introduce the syntax of FCGlight; it will basically show how constructions in FCG get translated into the LIGHT/OSF format. Subsection 2.3 introduces the FCGlight semantics (both declarative and procedural) that will support FS subsumption, FS unification, parsing, generation and learning.

### 2.1 FCG brief overview: Syntax and procedural semantics

An FCG grammar (and also an FCGlight grammar) is a set of structures called *constructions*, which are written in FCG format, as exemplified in Figure 1. We will show later how this format can be translated into LIGHT format.

Here, unlike the FCG authors, we will give a *constraint*-based view on the definition of construction structures. To this aim, we will need the following basic notions taken from OSF-logic (not formally detailed here):

*Elementary constraints* considered in the sequel are of three kinds: sort constraints, feature constraints and equality constraints.[6] It is also useful to make the following

[4] When constraint reduction actions (see section 2.2) are performed in "soft" mode, then they do not affect logical entailment. This is why in fcgLIGHT reduction is replaced (at unification level) by constraint marking for deletion (indefinitely delayed or, alternatively, to be exploited by the parser/generator).

[5] Concerning the implementation, this is still "work in progress".

[6] Equality constraints are not used explicitly at syntax level; they are automatically derived while building (variable) substitutions, i.e. during operations manipulating the constructions/FSs: subsumption, unification (G.L.B. computation), and generalisation (L.U.B. computation).

```
((?top-unit
  (tag ?meaning (meaning (== (read ?event)
                             (reader ?event ?agent)
                             (event-type ?event for-a-while)))))
((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 word "cita", as presented in [14].

*Preliminary remark:* Unlike HPSG/LIGHT where rules are expressed just as FSs — therefore letting the user to employ a unique formalism for both the rules and the structures on which they apply —, in the FCG formalism the *constructions* (expressing rules) differ from the structures on which they are applied, namely *coupled feature structures*. This *difference* is due to the fact that certain operations to be performed by the parser/generator are explicitly expressed in the constructions that represent rules.[7] In FCGlight we aim to reduce at minimum (and in certain conditions we even eliminate) this difference, and this enables the parser/generator to work very smoothly.

*Informal definition:* A coupled feature structure (CFS) can be seen as a set of elementary constraints, partitioned into two disjoint subsets named *poles*, usually referred to as the *syntactic* pole and the *semantic* pole (more generally: the *left* pole and the *right* pole). Each of the two poles groups its elementary constraints into several *units*. Units are linked into graph (usually tree) structures 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*/features. Examples of such slots are SYN-CAT and SEM-CAT.

*Important remark:* CFSs and constructions in FCG are characterised by a certain, *de facto* user-specified correspondence between syntax and semantics.

[7] This leads to breaking the so nice correspondence between declarative and procedural semantics and also the orthogonality between the unifier level and the parser/generator level that characterise the main-stream unification grammars. This is the price paid by FCG for the benefit of offering the grammar writer the capacity to play with subtleties in the learning (evolvable) grammars.

This correspondence is extended further down to MEANING and FORM constraints.

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

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

- two *operators*:
  - the *J operator* that (indirectly) indicates SYN/SEM-SUBUNITS constraints and also separates the merge zone from the match zone in a pole, and
  - the *tag operator* that indicates a substructure (set of elementary constraints) to be deleted and eventually moved elsewhere, and
- several *restrictors* ('1==', '0==', etc.), which can be seen as meta-constraints to be checked while doing match or merge. For further details, the reader should consult [20].

*Important remark:* In FCG, the match and merge operations are slot-limited. As a consequence, FCG lacks the automatic constraint reinforcement mechanism used in LIGHT/OSF-logic for single-valued features: $X.f \doteq Y$ and $X.f \doteq Z$ implies $Y \doteq Z$. In our opinion, this is why the operator (1==) is often used in writing FCG constructions. Similarly to the LIGHT system, in FCGlight we will make no use of the negation (0==). In FCGlight multi-valued features will be restricted to SYN-SUBUNITS, SEM-SUBUNITS, and also the FORM and MEANING constraints used as preconditions in parsing or generation (see the next subsection).

### 2.2 FCGlight: Syntax

It is immediate that a CFS can be written as a FS by introduction of the additional features SYN and SEM.[8] What we get is a syntactico-semantic graph (henceforth abbreviated as syn-sem graph). The notion of *syn-sem graph* will be used in the sequel as an alternative/replacement for the notion of CFS.

Now we will redefine (for FCGlight) the notion of *construction* by getting it as close as possible to the notion of FS. We will show that in order to do this, apart from indicating the preconditions and the main (body of) constraints to be used for parsing and respectively generation, only (explicit) reduction of constraints is needed.

*Definition:* In FCGlight, a *construction* is a set of elementary *specifications* — i.e. atomic constraints and actions — divided into the following (non necessarily disjoint) sets:

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

*sem.*

| | |
|---|---|
| *precond.* | `#top-unit[ MEANING =>> #event:read`<br>`  [ READER #agent,`<br>`  EVENT-TYPE for-a-while ] ]` |
| *producer actions* | `REDUCE: #top-unit[ MEANING =>> #event:read`<br>`  [ READER #agent,`<br>`  EVENT-TYPE for-a-while ] ]` |
| *main* | `#top-unit`<br>`[ SEM-SUBUNITS =>> #verb-unit`<br>`  [ MEANING  #event:read`<br>`  [ READER #agent,`<br>`  EVENT-TYPE for-a-while ],`<br>`  REFERENT #event,`<br>`  SEM-CAT  top`<br>`  [ BASE-TYPE( #event ) event ] ] ]` |

*syn.*

| | |
|---|---|
| *precond.* | `#top-unit[ FORM =>> #verb-unit[ STRING "cita" ] ]` |
| *parser actions* | `REDUCE: #top-unit[ FORM =>> #verb-unit[ STRING "cita" ] ]` |
| *main* | `#top-unit`<br>`[ SYN-SUBUNITS =>> #verb-unit`<br>`  [ STRING "cita",`<br>`  SYN-CAT verb:pos`<br>`  [ GENDER( #agent ) #agent-gender,`<br>`  CASE( #object )  accusative ] ] ]` |

**Fig. 2.** The sets of elementary specifications for the lexical entry in Figure 1, shown here in OSF format. The double-headed arrow (`=>>`) stands for multi-valued features as found for instance in F-logic [15]. OSF-logic can be naturally extended so as to accommodate such features. The hash symbol (`#`) introduces variables, while the colon (`:`) precedes the sort of a variable.

– preconditions constraints, (constraints marked for) reduction actions, and the 'main' (body of) constraints for *parsing*, and
– preconditions constraints, (constraints marked for) reduction actions, and the 'main' (body of) constraints for *generation*.

In FCGlight we explicitly make the following *demand:*[9]

For both parsing and generation, the constraints to be reduced constitute a subset of the 'precondition' set of constraints, and the set of 'main' constraints is disjoint from the 'precondition' set.[10]

[9] It seems to us that in practice, this demand is generally met in writing FCG grammars.

[10] Up to those constraints which are moved from one set to another (by translating into FCGlight the effect of the FCG tag operator).

```
/* 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,
                               EVENT-TYPE for-a-while ],
                     SEM-CAT top[ BASE-TYPE < #event, event > ] ],
  SEM-SUBUNITS =>> #verb-unit,
  ARGS < #top-unit[ FORM =>> #verb-unit[ STRING <! "cita" !> ] ] > ]

/* cita ; generation */
#top-unit
[ SEM-SUBUNITS =>> #verb-unit
                   [ REFERENT #event,
                     MEANING #event:read
                             [ READER #agent,
                               EVENT-TYPE for-a-while ],
                     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,
                              EVENT-TYPE for-a-while ] ] > ]
```

**Fig. 3.** The two FCGlight rules which are associated to the construction "cita" given in Figure 1. The ARGS feature designates the RHS of the rule.

The syntax <! !> is a sugared notation for difference lists. We *note* here that this is a very convenient way to replace the (interpretable) constraint 'meets' used in FCG.

We also make the *remark* that, when getting the above rules, we dealt with constraint reduction in the soft way. That means the feature TO-BE-REDUCED was not used. This is because the constraints to be reduced are identical to those that make up the preconditions, and this is true for all lexical constructions given in [13]. In such a case, it is the duty of the parser/generator to take actions that compensate for its use, i.e. to keep track of the constraints that (otherwise) should have been reduced.

*Important remark:* Our opinion is that in many cases one can actually get rid of constraint reduction. In FCGlight until now we only do a soft treatment of reduction, i.e. we mark constraints for reduction via the subsumption and unification mechanisms, using the reserved feature TO-BE-REDUCED. The parser/generator analysis these markings (if they are indeed needed). Such a treatment is absolutely sufficient for reproducing in FCGlight a quite elaborated example of learning in FCG like the one described in Gerasymova's MS thesis [13].

In order to give an *exemplification* of the above definition of construction in FCGlight, the sets of specifications (constraints and actions) that build up the "cita" construction which was given in Figure 1 in FCG format are presented in Figure 2 in LIGHT/OSF format. For convenience, FSs were used instead of sets of elementary constraints. The reader will note that in Figure 3, the rules corresponding to the construction shown in Figure 1, the reduction actions are omitted.

*Important note:* Basically, what makes a *key difference* between FCG and its FCGlight subset (at syntactic level) is that in FCGlight the constraints associated to a slot must represent a single-rooted feature structure.

It is natural to associate each construction in FCGlight format two *rules*, one for parsing and one for generation. The two LIGHT rules which are built for the "cita" construction given in FCG syntax in Figure 1, starting from the sets of specifications in Figure 2 are shown in Figure 3. The algorithm for getting these LIGHT rules is presented in Figure 4.

To *summarize* this subsection: the general form of a parsing or generation rule that corresponds to an FCGlight construction is

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

where $\mu$ is the 'main' set of constraints (written as a rooted FS), $\psi$ is the 'precondition', and $\alpha$ is a set of (parser and respectively generator) reduction 'actions'.

If the reduction actions are implemented in soft mode, and $\alpha'$ is a FS marked so as to correspond to the actions $\alpha$, then the rule $\mu : -\psi, \alpha$ becomes $\mu, \alpha' : -\psi$.

### 2.3 FCGlight: Semantics

FCGlight inherits for free the declarative semantics that makes the backbone of the LIGHT system, namely (one based on) OSF-logic, both at the FS unification and subsumption level, and at the control level over FSs (e.g. parsing and generation) if reduction actions are implemented in soft mode.[11]

Unlike the LIGHT system which supports binary and unary rules, in FCGlight all rules are unary rules. However, the argument (pre-condition, or right hand side (RHS)) of each rule is not a phrase structure; it is the description of a

[11] Unlike LIGHT for LinGO, FCGlight works with the un-typed counterpart of OSF-logic.

1. the 'initial' translation step:
   starting from the FCG format of the given construction,
   (and by following the guidelines for construction application in FCG)
   build the corresponding sets of elementary specifications
2. the 'intermediate' translation step:
   – for parsing:
      starting from the 'initial' translation result and
      put the parsing precondition and
         the parser REDUCE actions
      into the RHS of the newly created (parsing) rule
      the rest, including
            the parser J actions (taken without the J operator itself), and
            all stuff in the 'sem' pole
                  including the generator's J actions, but
                  except the generator REDUCE actions
         is put it into the LHS part of the new (parsing) rule
   – for production: proceed similarly
3. the 'pre-final' translation step:
   – in the LHS of each rule resulted from the 'intermediate' step,
      unify the FSs having the same root identifier
   – assuming that the FSs in the 'precondition' of each rule
         that resulted from the pre-final translation step represent a connex tree
      put it as such (i.e under the form of a single-rooted FS)
   – the same for the LHS part of the two rules
   – do the same for the REDUCTION actions if necessary
4. the 'final' translation step:
   – put REDUCTION actions unto soft form
      (i.e. mark constraints for reduction, using the TO-BE-REDUCED feature)
   – introduce difference lists for the STRING feature
   – replace the ("interpretable") feature MEETS with constrains between variables
         in the difference lists introduced above
   – transform features whose names are terms
      Example: CASE( ♯object ) accusative
      becomes: CASE <♯object, accusative>
   – extract sort (s1:s2) declarations
   – finally get LIGHT rules,
      using the feature ARGS to designate the rule's RHS part.

**Fig. 4.** The algorithm which, starting from a construction given in FCG format, obtains the two FCGlight rules to be used in parsing and respectively generation. For instance, for the construction which was given in Figure 1, the two FCGlight rules produced by this algorithm are those shown in Figure 3. The sets of elementary specifications into which that construction is de-composed (see step 1 from above), were presented in Figure 2.

subgraph in the syn-sem graph created during parsing/generation. The same is true about the rule's left hand side (LHS).[12]

Unlike LIGHT, in fcgLIGHT the unique argument of a rule is checked for compatibility (with a syn-sem subgraph) by using FS *subsumption* (match, in the FCG formulation). The rule's LHS will be treated via FS *unification*.

*Important remark:* It should be clear by now that, instead of having one construction/form treated in two different ways during parsing and respectively generation (as it is done in FCG), in FCGlight we explicitly associate each construction two rules which are treated in unique manner (remember the subsumption, reduction, and unification sequence) in both parsing and generation.

Unlike LIGHT, in FCGlight the parsing and generationcan be partial.[13] For parsing this means that we drop off the 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. In fact there is usually no designated start symbol in FCG grammars. Its place is somehow taken by a so-called *top unit*, to which all the other units (morphological, lexical, syntactic or semantic) get linked in a way or another.

*Definition (rule application):* Given $\tau$, the syn-sem graph whose root is the *top unit* and whose arcs are given by the features SYN-SUBUNITS and SEM-SUBUNITS, a rule of the form $\mu : -\psi, \alpha$ is executed as follows:

if subsume( $\psi, \tau'$ ), and $\sigma$ is the corresponding most general substitution,
then
    perform the actions expressed by $\alpha\sigma$ and
    return unify( $\mu\sigma, \tau''\sigma$ )
else return false.

where $\tau'$ is an arbitrary subgraph of $\tau$, and $\tau''\sigma$ is the subgraph of $\tau$ whose root is identified by the variable $root(\tau''\sigma)$.

The following simple algorithm formalises the way *parsing* in done FCGlight:

*Input:*
    a grammar $G$, and
    $\tau$ the (initial) syn-sem graph corresponding to
        the FORM of a given input sentence.
*Procedure:*
    as long as parsing rules in $G$ apply succesfully on $\tau$ or its subgraphs,
        build up the corresponding new syn-sem graphs.

[12] Here comes another *restriction* for FCGlight grammars: all units specified in a construction should constitute a rooted graph, via the SYN-SUBUNITS and SEM-SUBUNITS features.

[13] Generation is not yet implemented for full LIGHT, but it is already done in the FCGlight prototype.

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 no such graphs result, then (partial) graphs may be considered.

Similarly one can formalise generation in FCGlight.

## 3 Learning in FCGlight

Unlike FCG where learning is reinforcement-based, i.e. each construct receives a weight (between 0 and 1) which is increased when the construction is used in successful parsing/generation, and it is decreased otherwise, in FCGlight (and ilpLIGHT) the learning is hierarchy/ILP-based, i.e. it amounts to searching in a space partially ordered by a generalisation/specialisation relation between grammars.

Unlike ilpLIGHT, in the framework of which the learning is done in *off-line*/batch mode, by using a "golden" test suite given to the learner by the supervisor/teacher (see Figure 5),[14] learning in FCGlight is done in *on-line* (i.e. interactive) mode, via language games played by 2 agents (see Figure 6), like in FCG. Said it otherwise, in FCGlight the test suite is dynamically completed, via communication between the two agents.

Our *objective* here is to find/explore in FCGlight new construction learning (aka grammar repair) strategies compared to those used in FCG. For instance, while in ilpLIGHT we were restricted to learning (generalisations and/or specialisations) *within* rules, in fcgLIGHT we will be able to invent new rules.

In Figure 7 we sketch an *exemplification* of such alternative learning strategies, based on a modified version of the grammar learning example presented by Gerasymova in her MS thesis [13], chapter 4. The generalisation work (see step 2) over holophrases learned there (see step 1) is detailed in Figure 8.

Concerning Figure 7, we make the following important **remark** in relation to the mapping rule and the semantic rule which were eliminated from the *target grammar* (together with the 'po-' lexical construction) in order to get the *start grammar*. These two rules could be easily combined, and so they will we considered in the sequel; therefore we will not stick to the 3-step learning scheme (inspired from Tomasello) used by Gerasymova. Instead we will use a 2-step strategy for building lexical entries, as it is common in the LIGHT system.[15]

Moreover, concerning the step 1 in Figure 7, we should mention that in [13], the 'pocita' holophrasis was obtained in fact after having generalised on the endings '-l' and '-la' (something which she does not mention). This holophrasis is obtained by *i.* starting from the LUB (generalisation) of the two syn-sem trees corresponding to parsing the sentences "Misha pocital" and "Masha pocitala", and *ii.* further generalising it by eliminating the subject (Misha/Masha/kto).

[14] I.e. a set of sentences with associated parsing trees and eventually other informations.

[15] We are not interested here in maintaining the relationship between what we learn and the learning schemata discussed in Tomasselo's work, as it was done in Gerasymova's thesis. We consider this too difficult/complex an endeavour to be followed by the learning agents.

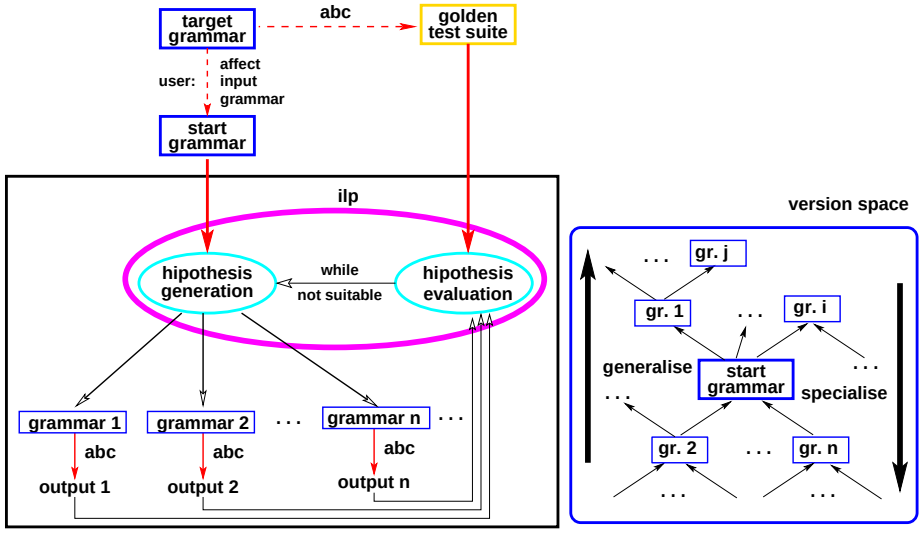



**Fig. 5.** Learning in ilpLIGHT. Left: The functional schema of the ilpLIGHT extension/configuration of the LIGHT system. **abc** and **ilp** are the parser and respectively the learner modules of LIGHT. Right: Illustrating the version space for grammars learned by ilpLIGHT. Upward arrows signify the generalisation relation between grammars. It is reverse to specialisation.

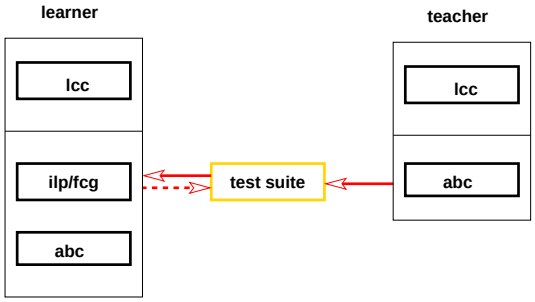



**Fig. 6.** Learning in fcgLIGHT. **lcc** (an abbreviation for: LIGHT into C Compiler) is a module in the LIGHT system that is in charge with the pre-processing and compilation of the input grammar. The dotted red arrow corresponds to (presumable) questions that the learner may ask the teacher in order to guide his search for better inferred rules/grammars. Compared to ilpLIGHT, here the learning is done in on-line manner: the test suite is dynamically produced by (or, under the supervision of the teacher ("speaker", in FCG).

**target grammar:**
as in Gerasymova's MS thesis [13], chapter 3
**input/start grammar:**
the target grammar without: the lexical construction for "po-", the mapping rule and the semantic rule.
**language game:**
1. setup: Misha read for-a-while; Masha read ongoing
   speaker/teacher: generate "kto pocital?"
   hearer/learner:
   parse "kto pocital?",
   get the corresponding *meaning*,
   try to disambiguate it wrt the given *setup*,
   get *failure*,
   send the message "I give up" to the speaker
   speaker: "Misha"
   hearer: learn the 'pocita' holophrasis/rule
2. after several such cycles:
   generalise (i.e. meta-learn) over the learned holophrases, as shown by the pseudo-code given in Figure 8.

**Fig. 7.** The pseudo-code for (the main steps in) the language game which was designed for learning for Russian verbs' aspect presented in [13], chapter 4.

The steps *a, b* and *c* in Figure 8 correspond to three learning subtasks in the work presented Gerasymova's MS thesis [13], which are performed there in a different order order (*c, a, b*) and by rather different means then those used by us, which we claim are simpler.

The reader should remember that holophrases are constructions, and each construction gets translated in FCGlight as two rules, one for parsing and one for generation. Therefore, when computing in FCGlight their LUB (generalisation), as required at the step *c.* in Figure 8, we will have to carefully work on the parsing and respectively the generation rules corresponding to such constructions.

We also *note* that will (have to) explicitly express at each learning step (i.e. at each construction invention) the **heuristics** that — eventually based on background linguistics knowledge — allow for choosing that (certain) construction among the (possibly many) different ones which are situated in between the *most specific* and the *most general* ones, compatible with the current/resulted test suite.

For instance, a linguistic motivation that supports that generalisation step *ii.* which was mentioned above when we talked about the formation of the 'pocital' holophrasis, could be the following one: 'po-' is a morpheme/prefix inside the word 'pocital', therefore one should concentrate on its relationship to the other morphemes that compose that word, namely the root 'cita' and the ending '-l'/'-la'.

*a.*

[*Note 1:* by simply analysing the association of pairs (prefix, verb),] it can be seen that:

- not all verbs could get prefixed (with 'po'-like prepositions), therefore:
- to distinguish between those that accept (such a prefix) and those that don't, introduce a new sort/feature value ('perfective') and
  - add (it as the value of) a new feature, 'ASPECT' at the verb's SYN-CAT level:

    SYN-CAT top[ ASPECT perfective ];

[*Note 2:* the complementary sort (to 'perfective') would be: 'non-perfective'.]

*b.*

[*Note 3:* by using an algorithm for identification of association rules on triplets (EVENT-TYPE, prefix, verb) or rather — given the outcome produced at the above point *a.*, by simply analysing the values of the EVENT-TYPE feature for events corresponding to the verbs whose ASPECT is 'perfective'] it will follow that:

- only events of type non-'ongoing' are associated (as meaning) to verbs that get prefixed, therefore:
- invent a new sort/feature value, 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 verbs that can be prefixed:

    SEM-CAT top[ VIEW( ♯event ) non-ongoing ];

[*Note 4:* in the FCGlight's sort hierarchy, the 'non-ongoing' sort should be the parent of all values for the EVENT-TYPE feature which are different from 'ongoing'];

*c.*

- group all holophrases with the same prefix (for example 'po-'), the same verb and the same value for the feature EVENT-TYPE for the verb's associated event);
- generalise each such group $g$ over the verb (and then replace $g$ in the current grammar) by applying the *least upper bound* (L.U.B.) operation on the FSs representing the holophrases the $g$ group;
- the construction corresponding to such a *generalised holophrasis* can be split-up into a *lexical construction* (FCG: syntactic rule) and a *lexical rule* (which in FCG is the composition of two — mapping and semantic — rules);
  - to get this done, it is necessary to introduce a new ('AKTIONSART') feature for the verb (and eventually the associated event); it makes the link between the prefix and the EVENT-TYPE value. For example:

    SYN-CAT top[ AKTIONSART delimitative ];
    SEM-CAT top[ AKTIONSART( ♯event ) delimitative ].

[*Note 5:* I suggest that it is enough to use the AKTIONSART feature at the verb's SYN-CAT level.]

**Fig. 8.** (Meta-)learning over holophrases. Constructions needed to parse/generate sentences that involve the Russian verbs' *aspect*, in a significantly different manner than [13].

## 4 Further work

We intend to apply such (and other) strategies on slightly different language games. It is expected for instance that learning the clitic pronouns in the Romanian language — a rather difficult issue for non-native speakers — would be possible if the teacher presents the learner with sets of sentences with identical (or only slightly different) meanings, as shown in Figure 9.

Mihai vede soldatul.
*Michael sees the soldier.*
Mihai îl vede pe soldat.
*Michael him sees on the soldier.*
Mihai îl vede pe el.
*Michael him sees on he.*
El îl vede.
*He him sees.*

Mihai o vede pe Maria.
*Michael her sees on Maria.*
El o vede pe ea.
*He her sees on she.*
El o vede.
*He her sees.*

Mihai vede soldaţii.
*Michael sees the soldiers.*
El îi vede pe soldaţi.
*He them sees on the soldiers.*
El îi vede pe ei.
*He them sees on they.*
El îi vede.
*He them sees.*

Mihai vede fetele.
*Michael sees the girls.*
El le vede pe fete.
*He them sees on the girls.*
El le vede pe ele.
*He them sees on they.*
El le vede.
*He them sees.*

Mihai vede un camion.
*Michael sees a truck.*
El îl vede.
*He it sees.*
⋆ El îl vede pe el.
⋆ *He it sees on it.*

Mihai vede o minge.
*Michael sees a ball.*
El o vede.
*He it sees.*
⋆ El o vede pe ea.
⋆ *He it sees on it.*

Mihai îi înmânează Mariei cartea.
*Michael gives Mary the book.*
El i-o înmânează Mariei.
*He her it gives Mary.*
El îi înmânează ei cartea.
*He her gives her the book.*
El îi înmânează cartea.
*He her gives the book.*
El i-o înmânează.
*He him it gives.*

Maria îi înmânează lui Mihai cartea.
*Mary gives Michael the book.*
Ea i-o înmânează lui Mihai.
*She him it gives to Michael.*
Ea îi înmânează lui cartea.
*She him gives to Michael the book.*
Ea îi înmânează cartea.
*She him gives the book.*
Ea i-o înmânează.
*She him it gives.*

**Fig. 9.** A small corpus of sentences intended to be used for learning Romanian clitics in FCGlight. The starred sentences are ungrammatical.

We expect that, in a similar way to the previous example, holophrases including clitics (îl, o îi, le) and case markers ('pe', 'lui') will be learned, and also the features associated to them (GENDER, NUMBER, DEFINITE, AGREEMENT). The result of the Romanian clitics formation in FCGlight could then be compared for instance to the HPSG description of these clitics as done in the Paola Monachesi's PhD thesis [16].

It is expected that after having completed several such learning tasks, we will be able to formulate several common learning (and meta-learning) strategies. Such a strategy is already visible in the *generalisation-specialisation* alternation during grammar learning, as shown in both ilpLIGHT demo for learning three basic HPSG principles [8] and Gerasymova's FCG example of learning Russian verb aspects.

**Acknowledgements:** Several ideas presented in this work have been outlined in the recent paper [10]. This work has been done in the framework of the European FP7 research project "ALEAR".

## References

1. H. Aït-Kaci and R. Di Cosmo. Compiling order-sorted feature term unification. Technical report, Digital Paris Research Laboratory, 1993.
2. H. Aït-Kaci and A. Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, 16:195–234, 1993.
3. H. Aït-Kaci, A. Podelski, and S.C. Goldstein. Order-sorted feature theory unification. *Journal of Logic, Language and Information*, 30:99–124, 1997.
4. B. Carpenter. *The Logic of Typed Feature Structures – with applications to unification grammars, logic programs and constraint resolution.* Cambridge University Press, 1992.
5. L. Ciortuz. A framework for inductive learning of typed-unification grammars. In P. Adriaans, H. Fernau, and M. van Zaanen, editors, *Grammatical Inference: Algorithms and Applications*, volume 2484 of *Lecture Notes in Artificial Intelligence*, pages 299–301. Springer-Verlag, Berlin, Germany, 2002. 6th International Colloquium: ICGI 2002, Amsterdam, The Netherlands.
6. L. Ciortuz. LIGHT — a constraint language and compiler system for typed-unification grammars. In *KI-2002: Advances in Artificial Intelligence*, volume 2479, pages 3–17, Berlin, Germany, 2002. Springer-Verlag. Proceedings of the 25th Annual German Conference on AI, Aachen, Germany.
7. L. Ciortuz. LIGHT AM – another abstract machine for feature structure unification. In S. Oepen, D. Flickinger, J. Tsujii, and H. Uszkoreit, editors, *Efficiency in Unification-ased Processing*, pages 167–194. CSLI Publications, The Center for the Study of Language and Information, Stanford University, 2002.
8. L. Ciortuz. Inductive learning of attribute path values in typed-unification grammars. pages 105–125. Scientific Annals of the "Al.I. Cuza" University of Iasi, Romania, Computer Science Series, 2003.
9. L. Ciortuz. *Parsing with Unification-Based Grammars — The* LIGHT *Compiler.* EditDan Press, Iasi, Romania, 2004.
10. Liviu Ciortuz and Ştefan Panţiru. Towards a LIGHT implementation of Fluid Construction Grammars. In IEEE Computer Society, editor, *Proceedings of ComputationWorld '09*, pages 511–514, Athens, Greece, 2009.

11. Joachim De Beule and Luc Steels. Hierarchy in Fluid Construction Grammar. In U. Furbach, editor, *Proceedings of KI-2005*, number 3698 in Lecture Notes in Artificial Intelligence, pages 1–15, Berlin, 2005. Springer-Verlag.
12. Daniel P. Flickinger, Ann Copestake, and Ivan A. Sag. HPSG analysis of English. In Wolfgang Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*, Artificial Intelligence, pages 254–263. Springer-Verlag, 2000.
13. K. Gerasymova. Acquisition of aspectual grammar in artificial systems through language games, 2009. Humboldt Universitaet zu Berlin, Germany, MS thesis.
14. K. Gerasymova, L. Steels, and R. van Trijp. Aspectual morphology of russian verbs in fluid construction grammar. In N.A. Taatgen and H. van Rijn, editors, *Proceedings of the 31th Annual Conference of the Cognitive Science Society*, pages 1370–1375. Cognitive Science Society, 2009.
15. M. Kifer, G. Lausen, and J. Wu. A logical foundation of object-oriented and frame-based languages. *Journal of ACM*, May 1995.
16. P. Monachesi. *A grammar of Italian clitics*. PhD thesis, Tilburg University, 1995. ITK Dissertation Series 1995-3 and TILDIL Dissertation Series 1995-3.
17. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
18. C. Pollard and I. Sag. *Head-driven Phrase Structure Grammar*. CSLI Publications, Stanford, 1994.
19. G. Smolka. Feature-constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87, 1992.
20. Luc Steels and Joachim De Beule. Unify and merge in fluid construction grammar. In Paul Vogt, Yuuga Sugita, Elio Tuci, and Chrystopher L. Nehaniv, editors, *EELC*, volume 4211 of *Lecture Notes in Computer Science*, pages 197–223. Springer, 2006.
21. Luc Steels, Joachim De Beule, Nicolas Neubauer, and Joris Van Looveren. Fluid Construction Grammars. In *Proceedings of the International Conference on Construction Grammars*, 2004.