

# Towards a LIGHT Implementation of Fluid Construction Grammars

Liviu Ciortuz  
Department of Computer Science  
“Al. I. Cuza” University  
Iași, Romania  
Email: ciortuz@infoiasi.ro

Ștefan Panțiru  
Department of Computer Science  
“Al. I. Cuza” University  
Iași, Romania  
Email: spanțiru@infoiasi.ro

**Abstract**—The aim of this work in progress is twofold: first to find out a significant, as large as possible subset of the Fluid Construction Grammar (FCG) formalism that can be supported by an efficient implementation, and second to check whether the LIGHT platform which until now was used for running large scale unification grammars can be (and if so, how it will be) extended so as to support evolving grammars like those defined in the FCG framework. This paper justifies and documents some of the basic principles that we defined and partially implemented for translating FCG into LIGHT.

**Keywords**—Fluid Construction Grammar; unification; feature structures; parsing; translation.

## I. INTRODUCTION

Fluid Construction Grammar is a formalism which was conceived for studying the evolution of language [14], [8], [13], [12]. The “ALEAR” EU FP7 research project aims to prove that autonomous robots are able — while using the FCG formalism when playing grounded language games — to develop certain language features, some of which resemble well-known features in the human languages, for instance the noun cases, the verb aspects, the use of anaphora, etc [15], [10].

FCG grammars are currently created and tested on the Babel2 platform (hence the FCG2 version of the FCG formalism) which was implemented at Vrije Universiteit Brussel and the CS “Sony” Laboratory in Paris [11]. The Babel2 system was written in Lisp and is mainly intended for grammar development, analysis, debugging, etc. From the computational point of view, FCG grammars are interpreted in Babel2, therefore doing sentence parsing and generation with them is not very efficient.

Our goal is to offer an alternative, efficient implementation of FCG grammars, especially those grammars that become stable after a certain development time on Babel2. As FCGs are a particular kind of unification-based grammars, we proposed to use an already highly optimised system for parsing/generation with fairly unrestricted unification grammars, and to adapt it so as to host FCGs. Among different such existing systems, for instance PET [2], LKB [7], and LIGHT [5], we have chosen the latter one in order to reach the goal stated above.

The LIGHT system was implemented in C. It contains a broad range of unifiers that work in both interpreted and compiled manner, together with a versatile parser. LIGHT was extensively tested and finely tuned on several versions of a very large unification grammar, namely the ERGO (previously known as LinGO) HPSG grammar for English [9]. The LIGHT implementation also contains a significant number of optimisations among which we mention here a feature structure sharing mechanism, the quick-check pre-unification filter [6], and a specialised form for compiled rules (written as feature structures) to be used in active bottom-up chart-based parsing [3].

We aim to show that these capabilities of the LIGHT system are quite helpful in adapting it so as to work a different kind of grammars than the ones for which this system was configured until now. On the other hand, certain extensions to the LIGHT system are required for running FCGs. Some of these extensions will have to deal with the fact that the LIGHT system works basically with stable, i.e. non-evolving grammars, while the FCG formalism was especially intended to model phenomena that may occur in the evolution of language.<sup>1</sup> Among the needed extensions are a translator from the FCG syntax to the LIGHT syntax, and a generator. A first version of the FCG → LIGHT translator was already implemented and is currently under tests; this paper we will present later its main design ideas.

As the FCG formalism is still under active development, our strategy for offering an efficient implementation of FCG grammars within the LIGHT system is first to cover a significant number of case studies — FCG grammars translated into LIGHT with their functionality thoroughly tested — and later on, after having worked on a large enough number of such case studies, we will naturally arrive at a

<sup>1</sup>It should be noted here that prior to this project, the LIGHT system has already been extended with a learning module [4]. Its design was inspired by the Inductive Logic Programming (ILP) paradigm. While tested on a number of reduced-size grammars, this module was able to either recover some “damaged” grammars or to extend other grammars with some important rules, based on analysing the parses that have been associated (by a supervisor) to sentences in a given testsuite. Therefore, the LIGHT system already has certain capabilities to deal with emerging grammars. We intend to use these capabilities when the time will come for implementing in LIGHT the grammar repairing strategies that will be defined in the near future by the FCG designers.

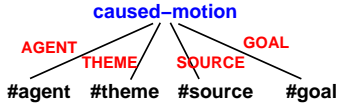


Figure 1. A very simple feature tree.

point when a large number (if not all) of the FCG grammars developed in Babel2 will also run on the LIGHT platform.

The next two sections will give a comparative view — LIGHT vs. FCG — on two of their main aspects, namely the specification of feature structures and the parsing strategies they employ. The comparisons will justify the introduction of three of the basic principles (designated below as PR1, PR2 and PR3) that we designed and use for translating FCG grammars into LIGHT.

## II. FEATURE STRUCTURES IN FCG AND LIGHT

The *feature structure* (FS) notion is central to unification grammars. Here we will informally define this notion and then we will present the way in which each of the two systems, LIGHT and FCG specify FSs. Thus some basic differences between the two systems will be outlined, making us able to draw the first two principles in translating FSs from FCG into LIGHT.

A feature structure is a syntactical expression which describes a *feature tree*, like the one shown in Figure 1. Such trees are single-rooted, and their edges are labeled with *features* (like AGENT, THEME, SOURCE), while their nodes are designated by constants (also called *sorts*, like caused-motion) or *variables* (#agent, #theme, #source). Variables can be restricted to a certain sort by using an is-a type declaration (#cm:caused-motion). Unrestricted variables are implicitly *top-sorted* (see below).

In a LIGHT grammar, the sorts are organised into an inferior semi-lattice  $\mathcal{S}$ , which means that it contains a *top* sort ( $\top$ ) and a *bottom* sort ( $\perp$ ), and for every two sorts  $s_1$  and  $s_2$  there is a unique greatest lower bound of them, usually denoted as  $glb(s_1, s_2)$ .<sup>2</sup> When the glb of two sorts  $s_1$  and  $s_2$  is the bottom sort ( $\perp$ ), it is said that  $s_1$  and  $s_2$  are incompatible.

FCG does not use a sort hierarchy, which is equivalent to say that it works with a set of constants which is a flat hierarchy of sorts.<sup>3</sup>

The partial order relation on  $\mathcal{S}$  can be naturally extended to a partial order relation on the set of all FSs that can be defined over  $\mathcal{S}$ . This new relation is called *subsumption* and is usually denoted  $\sqsubseteq$ . It can be shown that for any two given

<sup>2</sup>More precisely: let  $\preceq$  be reflexive and transitive closure of the partial order relation  $<$  in the semi-lattice  $\mathcal{S}$ . If  $u = glb(s_1, s_2)$ , then the following two properties are satisfied: 1.  $u \preceq s_1$ ,  $u \preceq s_2$ , and for every  $v$  such that  $s_1 \preceq v$ , and 2.  $v \preceq s_2$ , it follows that  $v \preceq u$ .

<sup>3</sup>However, we will translate certain unary predicates used in FCG grammars as sorts in LIGHT.

```
#cm:caused-motion
[ AGENT #agent,
  THEME #theme,
  SOURCE #source
  GOAL #goal ]
```

Figure 2. The LIGHT representation of the FS in Figure 1.

```
((caused-motion #:?cm-18)
 (agent #:?cm-18 #:?agent-11)
 (theme #:?cm-18 #:?theme-11)
 (source #:?cm-18 #:?source-8)
 (goal #:?cm-18 #:?goal-8))
```

Figure 3. The FCG representation of the FS in Figure 1.

FSs over  $\mathcal{S}$ , there is a unique glb of them with respect to the  $\sqsubseteq$  relation, and this glb is named the *unification* (result) of the two given FSs. If the unification result for two FSs is  $\perp$  (more exactly, the FS having the root node marked with the bottom sort  $\perp$ ), then the two FSs are said to be non-unifiable. Equivalently, it is said that the unification of these two FSs fails.

The LIGHT representation of FSs is straightforward, as it is shown in Figure 2. The way in which FCG represents FSs is not so smooth; it differs from LIGHT in two main respects:

1. As shown in Figure 3, a FS is written in FCG as a list of so-called *constraints*, or more precisely *elementary constraints* as defined in OSF-logic. This logic has been adopted as the logical background of LIGHT [1]. The elementary constraints employed in FCG are feature constraints and sort constraints. As a consequence, a list of units in FCG can represent multi-rooted FSs. Here naturally follows our first principle in translating FCG into LIGHT.

**PR1:** At FS level, we only translate those (maximal) lists of constraints (in the given FCG grammar) that represent single-rooted FSs.<sup>4</sup>

2. While a LIGHT grammar is (basically) made of a set of lexical FSs and a set of rule FSs, an FCG grammar is a set of coupled feature structures. A *coupled feature structure* (henceforth abbreviated, cFS) is made of two lists of *units* (FCG creators called these two lists *poles*) that separate the semantics features from the syntactic features of a given FS.<sup>5</sup> The reader can easily see this distinction in the example shown in Figure 4; the two poles are separated by the item  $\langle - \rangle$ . We can formulate now the second principle for translating FCG into LIGHT.

**PR2:** A cFS in FCG is translated in LIGHT as one single-rooted FS with two “reserved” features defined at the root level, namely SEM and SYN; their values are respectively the semantic and the syntactic poles/FSs in the given cFS.

<sup>4</sup>Up to now, we do not see this as a real limitation, as further examples will hopefully show it.

<sup>5</sup>Note that sharing of substructures between the two FSs is made possible by the usage of variables.

```

<coupled-feature-structure:
((#:top-206
 (tag #:?meaning-90
  (meaning (== (carpet #:?r-56)))
  (sem-cat (==0 (pom category))))
 (j #:?carpet-unit-49 #:?top-206)
 (referent #:?r-56)
 #:?meaning-90
 (sem-cat
  (==1 (pom category)
   (category (inanimate))
   (number sing))))))
<-->
((#:top-206
 (tag #:?form-104
  (form
   (== (string #:?carpet-unit-49 "carpet")))
  (syn-cat (==0 (pof common-noun))))
 (j #:?carpet-unit-49 #:?top-206)
 #:?form-104
 (syn-cat (==1 (pof common-noun))))>

```

Figure 4. Description of the lexical entry for the word *carpet* in FCG.

```

carpet-le
[ SEM top
  [ REFERENT #r-56,
    MEANING #r-56:carpet,
    SEM-CAT top
      [ CATEGORY < inanimate >,
        NUMBER sing ] ],
  SYN top
    [ SYN-CAT common-noun ],
    FORM < "carpet" > ]

```

Figure 5. Description of the lexical entry for the word *carpet* in LIGHT.

As an example for the application of this principle, Figure 5 gives the LIGHT correspondent of the FCG cFS shown in Figure 4.

### III. PARSING IN FCG AND LIGHT

This section outlines the main differences in doing parsing with LIGHT and respectively FCG grammars. In the latter case, we will briefly explain the use of the J and tag operators. Their understanding is crucial in further extending the set of principles that underline the translation of FCG grammars into LIGHT.

Parsing in the LIGHT system is currently done in an (active) bottom-up chart-based manner. It is required that each word/token in the input sentence is associated at least one (lexical) feature structure in the grammar. Using rule FSs, these lexical FSs are combined via unification, in a deductive-like bottom-up manner, until (at least) one instance of a rule FS subsumes the whole input sentence. This idea is illustrated by the example given in Figure 6.

Rule FSs in LIGHT have a “reserved” feature, ARGS, whose value is represented by the list of arguments taken by that rule. No “knowledge” is encoded into any rule FS regarding how it will be related to other FSs — apart its direct arguments/descendants — in the syntactic tree to be produced during parsing.

Unlike LIGHT, the FCG formalism is not fully declarative, in the sense usually employed for deductive parsing. The rule

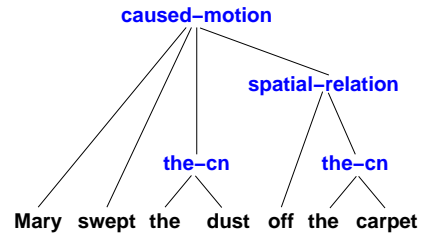


Figure 6. The parsing tree obtained by LIGHT for the sentence *Mary swept the dust off the carpet*.

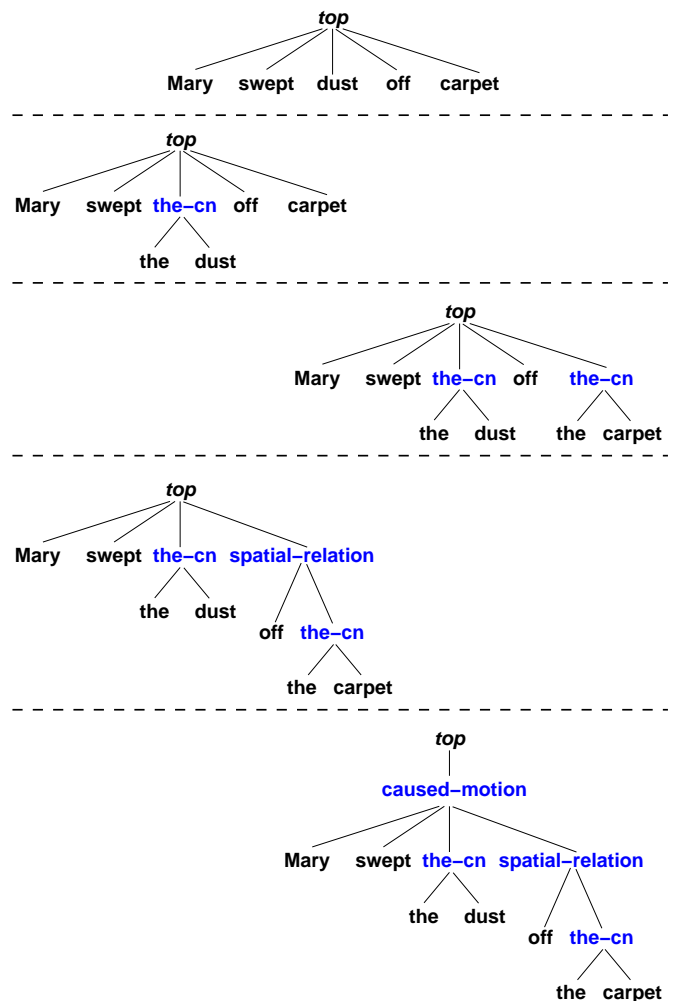


Figure 7. The steps performed by FCG when parsing the sentence *Mary swept the dust off the carpet*.

cFSs that compose a grammar incorporate information not only on which *preconditions* have to be checked, and how the rule will eventually *create* different argument cFSs, but also on how some arguments are to be a priori *disconnected* from their parents nodes and *re-connected* to other nodes

in the syntactic tree like in the so-called word grammars. In order to specify these (procedural) issues, FCG employs two operators: J and tag.

As shown in Figure 7, parsing in FCG is done in a combined push-down and bottom-up manner. (One can also see it as merging phrase-based and word-based analysis.) A *top* cFS<sup>6</sup> is initially fed with certain (mainly, vicinity) constraints related to the words/tokens in the input sentence. The application of a lexical cFS requires the a priori satisfaction of the constraints specified in its syntactic pole, in conjunction with constraints in the syntactic pole of the *top* cFS. Upon successful satisfaction of these constraints, the J declarations in the lexical cFS instruct the parser how the lexical cFS is to be linked to the *top* cFS. Similarly, a J declaration in a non-lexical cFS tells the parser that certain arguments/units will be linked to another (mother) unit, and that the later one will eventually relate to the *top* cFS.

The tag operator usually enhances the effect of the J operator: it identifies/marks a subtree in a cFS and asks the parser to erase it from that place. The J operator may eventually reattach that subtree inside another FS.<sup>7</sup> The third principle for translating FCG into LIGHT is as follows.

**PR3:** We only translate in LIGHT those cFSs that make use of the J and tag operators in a manner that corresponds to the LIGHT parsing strategy.<sup>8</sup>

#### IV. CONCLUSION AND FURTHER WORK

We are in the process of extending the LIGHT system, which was priorly used for large-scale non-evolving unification grammars, so as to host end efficiently run Fluid Construction Grammars, which are essentially evolving grammars. Here we presented the basic principles employed by the FCG → LIGHT translator that is currently in a step-by-step extension and test process. Further on we will take most of the non-declarativeness (but also the stuff concerning the morpho-lexical and semantic analysis) out of FCG cFSs, thus simplifying their unification. The control level above unification, i.e. the parser, will be put in charge with the rest.

#### REFERENCES

- [1] H. Ait-Kaci, A. Podelski, and S.C. Goldstein. Order-sorted feature theory unification. *Journal of Logic, Language and Information*, 30:99–124, 1997.
- [2] U. Callmeier. PET — a platform for experimentation with efficient HPSG processing techniques. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):99–108, 2000.
- [3] L. Ciortuz. On compilation of head-corner bottom-up chart-based parsing with unification grammars. In *Proceedings of the IWPT 2001 International Workshop on Parsing Technologies*, pages 209–212, Beijing, China, 2001.
- [4] 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.
- [5] 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.
- [6] L. Ciortuz. On two classes of feature paths in large-scale unification grammars. In H. Bunt, J. Carroll, and G. Satta, editors, *New Developments in Parsing Technologies*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2004.
- [7] A. Copestake. *The (new) LKB system*. CSLI, Stanford University, 1999.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] Martin Loetzsch, Pieter Wellens, Joachim De Beule, Joris Bleys, and Remi van Trijp. The Babel2 manual. Technical Report AI-Memo 01-08, AI-Lab VUB, Brussels, Belgium, 2008.
- [12] Luc Steels. Language games for autonomous robots. *IEEE Intelligent Systems*, 16(5):16–22, September 2001.
- [13] 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.
- [14] Luc Steels, Joachim De Beule, Nicolas Neubauer, and Joris Van Looveren. Fluid Construction Grammars. In *Proceedings of the International Conference on Construction Grammars*, 2004.
- [15] Remi van Trijp. The emergence of semantic roles in Fluid Construction Grammar. In A. D. M. Smith, K. Smith, and Ferrer, editors, *Proceedings of the 7th International Conference on the Evolution of Language*, pages 346–353. World Scientific, March 2008.

<sup>6</sup>This is not to be confused with the *top* (T) sort in LIGHT.

<sup>7</sup>The dependencies between FSs corresponding to words will be recorded by the SEM-SUBUNITS and SYN-SUBUNITS “reserved” features.

<sup>8</sup>Again, we do not see this as a real limitation, provided that the LIGHT system can host other parsing strategies than the current (active, bottom-up, chart-based) one.