

# Encoding LFG f-structures in the TL-LFG type system

Miltiadis Kokkonidis

Computational Linguistics Group  
University of Oxford

## Abstract

Type-Logical Lexical Functional Grammar is a new syntactic framework similar in nature to Type-Logical Categorical Grammar, but with a distinct Lexical Functional Grammar flavour. It is the overall setup of TL-LFG that makes it a close relative of TL-CG, but it is the information encoded in the type system that brings it even closer to LFG. The encoding of f-structure information in the TL-LFG type system is hereby discussed.

## 1 Introduction

Glue Semantics (Dalrymple et al., 1993; Dalrymple, 1999; Dalrymple, 2001) emerged as an attempt to provide an alternative to phrase-structure-driven semantic composition. For this, the commutative (order-insensitive) and resource-sensitive nature of Linear Logic (Girard, 1987) was ideal. Glue has always been the intermediary between syntax and semantics formalisms. Although originally designed for LFG, it has also been successfully integrated with LTAG (Frank and van Genabith, 2001), HPSG (Asudeh and Crouch, 2002), CG (Asudeh and Crouch, 2001), and CFG (Asudeh and Crouch, 2001).

First-Order Glue (Kokkonidis, 2007a) is a fairly recent development. It is the result of a radical redesign of the formal foundation of Glue that brings out the true nature of quantification in the type system. Quantification is and has always been first-order in Glue. However, in the currently popular system of Dalrymple et al. (1997), it appears to be second-order as its designers based that system on

a linear version of System F (Girard, 1989). They had to both introduce two sorts ( $e$  and  $t$ ) and restrict quantification to base types (base types of sort  $t$  to be precise) in order to get a system with the desired behaviour. First-Order Glue with base-type constructors (predicates)  $e/1$  and  $t/1$  did not emerge as a result of a number of ad-hoc restrictions and extensions; it is by its very nature that it has the correct formal properties for its intended purpose. As a result, the First-Order Glue system is simpler than its predecessor.

However, it also came with additional encoding power: in predicate logic, a predicate can have more than one individual as its argument and individuals can be variables, constants, or the result of applying a function from a number of individuals to a single individual. This additional encoding power may prove useful in Glue analyses interfacing syntax and semantics. However, it also inspired a use of that type system as a new syntax formalism on its own accord. The ability of First-Order Glue base-type constructors to involve more than one individual either directly (by having a higher arity) or indirectly (by having as its argument the result of a function taking multiple arguments) meant that arbitrarily hierarchical feature structures could be encoded.

A further development resulted in a type system more appropriate than G3 for the purpose of encoding syntactic (and other) information in its base types.  $G3_i$  (Kokkonidis, 2006) was meant to be a simplified version of G3 that would facilitate both the teaching and the implementation of Glue.  $G3_i$  differs from G3 in a number of ways. What is of particular interest here is the fact that in  $G3_i$ , unification is used instead of quantification. Whereas G3 can, given an appropriate typing context, form

an expression of target type, say,  $t(s)$ , or  $\forall\alpha. t(\alpha)$ , it can not do the same for type  $t(\alpha)$  given some underspecified  $\alpha$  which will become gradually more and more specified in the course of concurrent semantic and syntactic information composition. This is exactly what unification in  $G3_i$  permits.

A number of developments concerning the formal foundation of Glue Semantics, its type-system, lead to the latter being suitable not only for being an interface between a syntax and a semantics formalism (Glue), but also a variety of syntactic formalisms based on a simple first-order linear type system. TL-LFG (Kokkonidis, 2007c) is one such formalism. It combines  $G3_i$  with a simple mechanism for checking functional uncertainty, existential, and other constraints imported from LFG.

This paper focuses on an issue at the very heart of the design of TL-LFG, the encoding of syntactic features in the base types of  $G3_i$ . It is this encoding that turns  $G3_i$  into a grammar formalism with a recognisable LFG heritage. In section 2, an overview of the architecture of TL-LFG is given and the  $G3_i$  type-system is briefly presented. In section 3, the mapping from f-structures to first-order logic individuals as found in the type system is discussed. Finally, section 4 is a short summary.

## 2 The general setup

When an LFG+Glue system (e.g. XLE) parses a sentence it produces a c-structure, an f-structure, and a Glue typing context for it. Then, the meanings are assembled by the Glue type system. When an TL-LFG system does the same, it takes as its starting point the target type and the typing context built up by looking up the words of the given string in the lexicon and collecting their semantic atoms. Then the  $\lambda$ -calculus terms that correspond to the semantics of the word sequence (they are the terms that have the target type) are built up by the typing system, but also at the same time the target type feature values are instantiated in a way that reflects the functional structure and contents of the input word sequence (expected to be a well-formed sentence or else parsing will fail).<sup>1</sup> An example follows.<sup>2</sup>

<sup>1</sup>Fragment parsing is also possible, but the setup assumed here is for sentence parsing.

<sup>2</sup>Kokkonidis (2007b) discusses modification and this particular example in more detail.

- (1) Every<sub>0</sub> candidate<sub>1</sub> supported<sub>2</sub> an<sub>3</sub> overly<sub>4</sub> optimistic<sub>5</sub> view<sub>6</sub> 7

The TL-LFG typing context for (1), let us call it  $\Gamma$ , made up of the seven semantics atoms is:

- (2)  $every_{0\dots1} : (e_{s'} \multimap t_{s'}) \multimap (e_s \multimap t_\alpha) \multimap t_\alpha$ ,  
 $candidate_{1\dots2} : e_{s'} \multimap t_{s'}$ ,  
 $supported_{2\dots3} : e_s \multimap e_o \multimap t_f$ ,  
 $an_{3\dots4} : (e_{o'} \multimap t_{o'}) \multimap (e_o \multimap t_\beta) \multimap t_\beta$ ,  
 $overly_{4\dots5} : (e_{o''} \multimap t_{o''}) \multimap (e_{o'} \multimap t_{o'})$   
 $\multimap (e_{o''} \multimap t_{o''}) \multimap (e_{o'} \multimap t_{o'})$ ,  
 $optimistic_{5\dots6} : (e_{o''} \multimap t_{o''}) \multimap (e_{o'} \multimap t_{o'})$ ,  
 $view_{6\dots7} : e_{o''} \multimap t_{o''}$

The target type is  $t_f$ . The typing system of Figure 1 is charged with the task of finding terms  $M$  such that  $\Gamma \vdash M : t_f$ . It discovers that the two ways meaning atoms can be combined into a meaning  $M$  for the entire sentence are:

*Reading 1*

- (3)  $every_{0\dots1}$   
 $(\lambda x. candidate_{1\dots2} x)$   
 $\lambda x. an_{3\dots4}$   
 $(\lambda y. overly_{4\dots5} optimistic_{5\dots6} view_{6\dots7} y)$   
 $\lambda y. supported_{2\dots3} x y$

$$[\sigma(\alpha) = \sigma(\beta) = f]$$

*Reading 2*

- (4)  $an_{3\dots4}$   
 $(\lambda y. overly_{4\dots5} optimistic_{5\dots6} view_{6\dots7} y)$   
 $\lambda y. every_{0\dots1}$   
 $(\lambda x. candidate_{1\dots2} x)$   
 $\lambda x. supported_{2\dots3} x y$

$$[\sigma(\alpha) = \sigma(\beta) = f]$$

Replacing the meaning placeholders  $every_{0\dots1}$ ,  $candidate_{1\dots2}$ , etc. with their intended meanings gives the semantics of the sentence. Each of the two readings ( $M$ ) represents the body of a function from the meanings of the semantic atoms to the meaning of the whole.

$$\begin{array}{c}
(\neg\text{Intro.}) \\
\frac{\Gamma, X : T \vdash E : T'}{\Gamma \vdash \lambda X. E : (T \neg T')} \\
(\neg\text{Elim.}) \\
\frac{\Gamma_1 \vdash A_1 : T'_1 \quad \dots \quad \Gamma_N \vdash A_N : T'_N}{F : T_1 \neg \dots \neg T_{N+1}, \Gamma_1, \dots, \Gamma_N \vdash F A_1 \dots A_N : T_{N+1}[\sigma]} \\
[T_1[\sigma]=T'_1[\sigma], \dots, T_N[\sigma]=T'_N[\sigma], \text{ and } T_{N+1} \text{ is a base type.}]
\end{array}$$

where  $\sigma$  is some total function  
from variables to individual denoting expressions  
such that for any variable  $V$ ,  $\sigma(V) \neq V$ .

Figure 1: TL-LFG Type-Inference Rules

However, the focus of this paper is syntax, or more precisely functional syntactic structures and how they can be encoded in the type system of Figure 1. That system is based on unification rather than universal and existential quantification. While the premises can be straightforwardly interpreted as having all the variables in their types implicitly universally quantified, the interpretation of the variables in the target type is a bit more open ended. Both an interpretation assuming implicit universal quantification and another one assuming implicit existential quantification are possible, and both are useful. All non-instantiated<sup>3</sup> variables of the target type as originally specified can be thought of as universally quantified and all instantiated ones as existentially quantified. For the most part, since a complete f-structure is built, it will be the existential interpretation that will be used.

Of the many f-structures that appeared in the typing context, the final one,  $f$ , will be presented below. It includes as its parts also  $s$  and  $o$ . Intermediate f-structures  $o, o, o'', o'''$ , and  $s'$  did not make it to the final type i.e. they are not included in  $f$ .

In this example, the target type  $t_f$  is specified as:<sup>4</sup>

$$t_f: \begin{bmatrix} {}_0 \text{ Every } \dots \text{ view } {}_7 \\ \text{PRED } v \\ \text{SUBJ } s \\ \text{OBJ } o \\ \vdots \end{bmatrix}$$

*Convention:* Variables are written in italic, whereas constants are written in plain font style.

<sup>3</sup>A variables  $V$  is called instantiated if  $\sigma$  maps it to a non-variable or an instantiated variable.

<sup>4</sup>Instead of writing the SPAN feature in full, the span start and end values are written in the first line of the feature structure containing it. In examples, the words within the span are written in between the span start and end values as a reminder of what part of the word sequence the f-structure corresponds to.

However, as each of the two scope readings (the one where all companies supported a different overly optimistic view and the other where they all supported the same overly optimistic view) is obtained on the meaning side, the boxed variables will be instantiated also, resulting in the f-structure (ignoring tense, case and other features) for the sentence shown in Figure 2.<sup>5</sup>

The encoding of the types that appear in the standard AVM-based presentation of TL-LFG types into the actual first-order system on which it is based will be discussed next.

### 3 Encoding syntactic information in the base-types

The very core idea behind the Curry-Howard isomorphisms between proofs in a logic and terms in a related typed  $\lambda$ -calculus is that logic formulas correspond to types and proofs to  $\lambda$ -calculus terms. In a type system based on first-order logic, the base types correspond to predicates supplied with the the correct number of individuals. Individuals can be variables, constants, or functions from a number of individuals to individuals.

When working with first-order logic, it is possible to express something like ‘every candidate votes for their party’ using a one-place predicate *candidate*/1, a two-place predicate *vote*/2 and a single-argument function *party-of*/1<sup>6</sup> that is meant to map an individual belonging to a party to the party that individual belongs to (also interpreted as an individual):

$$\forall x. \text{candidate}(x) \rightarrow \text{love}(x, \text{party-of}(x)).$$

When working with a type system such as that of TL-LFG, it is also possible to use predicates taking multiple individuals as arguments and functions from tuples of individuals to a single individual. This is the one additional ability First-Order Glue came equipped with in its bid to replace the earlier Glue type system (Dalrymple et al., 1997) that actually lead to the development of TL-LFG. An example of a type in G3, the original type system for First-Order Glue, is:

$$\forall \alpha. c_\alpha \rightarrow l_{\alpha, p(\alpha)}.$$

<sup>5</sup>The analysis for modification assumed here is not the standard LFG analysis.

<sup>6</sup>A function  $f$  or predicates/base-type constructor  $p$  of arity  $n$  will be written  $f/n$  and  $p/n$  respectively. Functions and predicates/base-type constructors of the same name but different arity are unrelated.

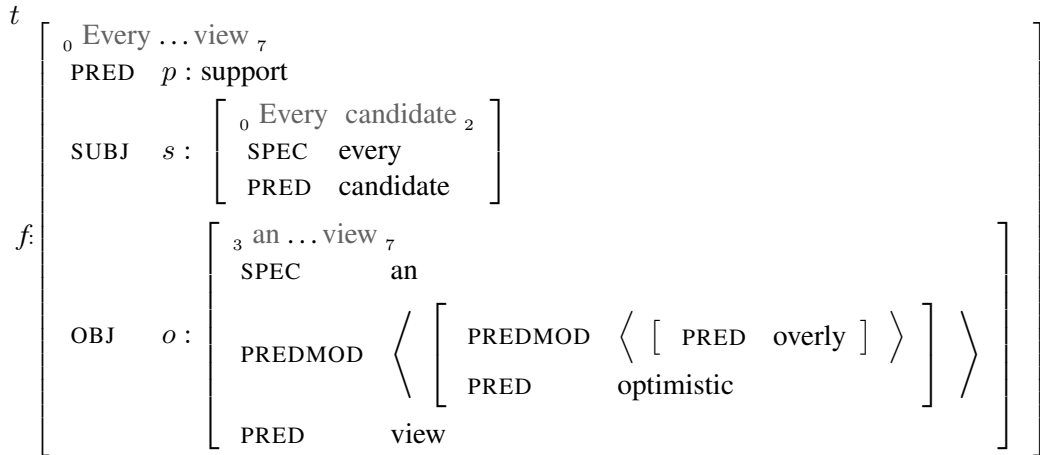


Figure 2: The instantiated target type for sentence (1).

However, TL-LFG has adopted  $G3_i$ , a different version of the First-Order Glue type system, one based on unification rather than quantification, so in TL-LFG the following type would be used instead

$$c_\alpha \rightarrow l_{\alpha,p(\alpha)}.$$

In the above example a number of base constructors appeared. For the purposes of the following discussion, the base constructors of interest will be  $t/1$  (truth values) and  $e/1$  (entities). They both take a single argument which will be an encoded feature structure. The encoding that will be described serves to demonstrate the feasibility of the endeavour and may not be the most efficient one. For example, different groupings of features could lead to slightly more efficient implementations. However, this will not be the highest priority in the following discussion.

### 3.1 Spans

The idea of numbering positions within the word sequence borrowed from chart parsing helps deal with word order. It is the ability to have base-type constructors of higher arity that inspired the whole programme of using linear first-order type systems originally meant for Glue Semantics as the main formal foundation for syntax formalisms as it makes it possible to encode spans. So, ignoring all other syntactic features, it would be possible to have  $t(0, 7)$  as the type of sentence (1). However, it is preferable to introduce a function *span* that takes the span start

and the span end as arguments. The advantage doing something like this is that this in effect groups multiple features together into one. This can be quite convenient when specifying lexicon entries. It also means that the Glue analysis of quantification can be transferred intact into TL-LFG; indeed, this was the original motivation for this grouping. So instead of  $t(0, 7)$ , a better type for sentence (1) would be  $t(\text{span}(0, 7))$ . However, SPAN will be one of many attributes within TL-LFG feature structures. They will be treated as having as their value a feature structure of two attributes: START and END.

### 3.2 Feature Structures

Function symbols play a very important role in the encoding of complex hierarchical feature structures. It is possible to use a single function symbol, let us call it *fstr*/ $N$  (where  $N$  is the number of different attributes such as PRED, SPEC, SUBJ, OBJ, CASE, NUM etc. that we want to be able to use) where each attribute would be occupying one of the  $N$  argument positions. If that attribute was not intended to be a part of that particular f-structure then it will be given the special value  $\perp$ . In LFG an f-structure is a (potentially partial) function from attributes (feature names) to their values. In a sense what this idea corresponds to is saying that such a function can be made total (by adding  $\perp$  to its codomain and mapping all previously unmapped attributes to it). This function can be represented by an  $N$  tuple where  $N$  is the (finite) number of attributes that can appear

in an f-structure. For an example, the f-structure value of a CASE feature for a noun that has either accusative or dative case in a language with cases NOM, ACC, GEN, DAT would look something like this:<sup>7</sup>

$$fstr(\perp, \perp, \dots, \overbrace{-, \alpha, -, \delta}^{\text{casemarking}}, \dots, \perp, \perp).$$

As not all attributes appear in every f-structure, if that kind of encoding were to be used in an TL-LFG implementation, it would result into too much wasted space and time (for unifications of fields that are not meant to be used).

A better encoding would use different function symbols to group different kinds of f-structure.<sup>8</sup> With a specialised function for the case feature structure, the encoding of the f-structure value of a CASE feature for a noun that has either accusative or dative case would look like something like this instead:

$$casefstr(-, \alpha, -, \delta).$$

If we wanted to remind ourselves the position of the attributes, then instead of having their values bare in the relevant position, we could use them as arguments to unary functions bearing the name of the attribute. This would result into the following expression for the same example feature structure:

$$casefstr(nom(-), acc(\alpha), gen(-), dat(\delta)).$$

This is the actual encoding we will be using here as it is effortless for readers, especially those familiar with LFG. If we wanted to be fully explicit in how we write out the above as an attribute-value matrix (AVM) we would write:

$$\begin{bmatrix} \text{NOM} & - \\ \text{ACC} & \alpha \\ \text{GEN} & - \\ \text{VOC} & \delta \end{bmatrix}$$

For attributes included in the AVM depiction of an f-structure but having no explicit value, a different fresh variable would be used as the value of each.

<sup>7</sup>The analysis of Dalrymple et al. (2006) is used here.

<sup>8</sup>This proposal means that in essence there are different types of f-structure, let us call them f-types. We could define a type-system within the type system to deal with this fact. However, relevant as this endeavour is, it is also outside the scope of this paper.

So, assuming the variables  $\alpha$  and  $\delta$  did not appear anywhere else, they could be omitted:

$$\begin{bmatrix} \text{NOM} & - \\ \text{ACC} & \\ \text{GEN} & - \\ \text{VOC} & \end{bmatrix}$$

In open AVMs, attributes that do not appear are treated as if they did appear but with a fresh variable as their value. An open AVM is one with a vertical ellipsis inside it marking the fact that there are more attributes in the f-structure (with underspecified values). So we could also have used the following AVM depiction for the same f-structure:

$$\begin{bmatrix} \text{NOM} & - \\ \text{GEN} & - \\ \vdots & \end{bmatrix}$$

AVMs without the vertical ellipsis are *closed*. Any omitted attributes are taken to have the value  $\perp$ . In LFG, all AVMs are closed as they represent the end result of finding an f-structure that is the minimum solution to the given constraints. In TL-LFG there are also open AVMs as they correspond to LFG defining equations; a bunch of defining equations in LFG will not necessarily mention every possible attribute, but that does not mean that these attributes will not exist, as another bunch of defining equations could always specify their values.

### 3.3 Lists

In TL-LFG there is the notion of an empty list ( $\perp$ ), which coincides with the generic notion of empty value, and a list consisting of a head element *head* and a tail list *tail* which can quite simply be represented by the following AVM:

$$\begin{bmatrix} \text{HEAD} & \text{head} \\ \text{TAIL} & \text{tail} \end{bmatrix}$$

However, a special notation is used for lists, so the above is written as follows:

$$\langle \text{head} \mid \text{tail} \rangle.$$

For notational convenience,  $\langle x_1, \dots, x_m \mid X \rangle$  can be written instead of  $\langle x_1 \mid \langle \dots, \langle x_m \mid X \rangle \dots \rangle \rangle$  and  $\langle x_1, \dots, x_m \rangle$  can be written instead of  $\langle x_1 \mid \langle \dots, \langle x_m \mid \perp \rangle \dots \rangle \rangle$ .

LFG has a notion of a set of f-structures. One advantage of supporting lists rather than sets in TL-LFG (other than the fact they can be encoded in the type system quite trivially) is that they preserve the order elements are placed in them and can be accessed and manipulated in an order-sensitive way. If sets prove to be a useful concept they may always be introduced into TL-LFG, but as the design of TL-LFG takes the line of dispensing with unnecessary formal machinery, the need will have to be demonstrated first. Neither modification, nor case agreement, nor conjunction seem to necessitate sets for their analysis in TL-LFG.

LFG's notion of a set is rather awkward, but many would say also convenient; an LFG set is a proper set plus a feature structure with some attributes (Dalrymple, 2001). Unlike LFG sets, an TL-LFG list is a pure list. If additional features are to be associated with it, both those features and the list will have to be grouped together inside an f-structure that serves this purpose. The notation for an LFG set betrays the intuition that a set with features is really an AVM with a hidden attribute which is a pure set.

$$\left[ \begin{array}{l} \text{NUM pl} \\ \left\{ \left[ \begin{array}{l} \text{CASE } c : [\dots] \\ \dots \end{array} \right], \left[ \begin{array}{l} \text{CASE } c \\ \dots \end{array} \right] \right\} \end{array} \right]$$

LFG set attributes can be distributive or non-distributive. The latter are simply attributes of the containing AVM. The former are also attributes of the containing AVM, albeit hidden and with the additional constraint that their value is the same as the value of that same attribute within each element of the set. Such distributivity will have to be explicitly encoded in an TL-LFG grammar if desired. It can be argued that this makes things clearer from a formal perspective.

$$\left[ \begin{array}{l} \text{NUM pl} \\ \text{CASE } c : [\dots] \\ \text{CONJ } \left\langle \left[ \begin{array}{l} \text{CASE } c \\ \dots \end{array} \right], \left[ \begin{array}{l} \text{CASE } c \\ \dots \end{array} \right] \right\rangle \end{array} \right]$$

## 4 Conclusions

The simple encoding presented here shows how f-structure and span information can be encoded into the base types of  $G3_i$ , the main formal foundation of TL-LFG. This encoding permits TL-LFG to deal with f-structure and word order and as a result it enables TL-LFG to support the key aspects of the LFG + Glue approach to syntax and the interface to semantics with a fraction of the formal machinery: the Glue typing system is but a small fraction of the arsenal the LFG + Glue combination uses, whereas in TL-LFG,  $G3_i$ , a type system originally developed for Glue Semantics, is the main formal tool for dealing with both syntax and the syntax-semantics interface.

The encoding used here is a proof of concept; as the idea is that AVMs are what end-users of the formalism will be dealing with, the details of the encoding are left to the discretion of the designers of any particular implementation of TL-LFG.

## References

- Ash Asudeh and Richard Crouch. 2001. Glue semantics: A general theory of meaning composition. Talk given at Stanford Semantics Fest 2, March 16, 2001.
- Ash Asudeh and Richard Crouch. 2002. Glue semantics for hpsg. In Frank van Eynde, Lars Hellan, and Dorothee Beermann, editors, *Proceedings of the 8th International HPSG Conference*. Stanford, CA., CSLI Publications.
- Mary Dalrymple, John Lamping, and Vijay Saraswat. 1993. LFG semantics via constraints. In *Proceedings of the Sixth Meeting of the European ACL*, pages 97–105, University of Utrecht. European Chapter of the Association for Computational Linguistics.
- Mary Dalrymple, Vineet Gupta, Fernando C.N. Pereira, and Vijay Saraswat. 1997. Relating resource-based semantics to categorial semantics. In *Proceedings of the Fifth Meeting on Mathematics of Language (MOL5)*, Schloss Dagstuhl, Saarbrücken, Germany, August. An updated version was printed in (Dalrymple, 1999).
- Mary Dalrymple, Tracy Holloway King, and Louisa Sadler. 2006. Indeterminacy by underspecification. Poster presented at the LFG06 Conference.

- Mary Dalrymple, editor. 1999. *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach*. MIT Press.
- Mary Dalrymple. 2001. *Lexical Functional Grammar*. Number 42 in Syntax and Semantics Series. Academic Press.
- Anette Frank and Josef van Genabith. 2001. LI-based semantics construction for ltag- and what it teaches us about the relation between LFG and ltag. In *Proceedings of the LFG01 Conference*. CSLI Publications.
- Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science*, 50:1–102.
- Jean-Yves Girard. 1989. *Proofs and Types*. Cambridge University Press.
- Miltiadis Kokkonidis. 2006. A simple linear first-order system for meaning assembly. In *Proceedings of the Second International Congress on Tools for Teaching Logic*, Salamanca.
- Miltiadis Kokkonidis. 2007a. First-order glue. *Journal of Logic, Language and Information*. To appear.
- Miltiadis Kokkonidis. 2007b. Scoping and recursive modification in Type-Logical Lexical Functional Grammar. Unpublished manuscript.
- Miltiadis Kokkonidis. 2007c. Towards a functional type-logical theory of grammar. In Miriam Butt and Tracy Holloway King, editors, *Proceedings of the LFG07 Conference*. CSLI Publications. Forthcoming.