A Parser for LTAG and Frame Semantics

David Arps, Simon Petitjean
Heinrich-Heine-Universität Düsseldorf
david.arps@uni-duesseldorf.de, petitjean@phil.uni-duesseldorf.de

Abstract

Since the idea of combining Lexicalized Tree Adjoining Grammars (LTAG) and frame semantics was proposed (Kallmeyer and Osswald, 2013), a set of resources of this type has been created. These grammars are composed of pairs of elementary trees and frames, where syntactic and semantic arguments are linked using unification variables. This allows to build semantic representations when parsing, by composing the frames according to the combination of the elementary trees. However, the lack of a parser using such grammars makes it complicated to check whether these resources are correct implementations of the theory or not. The development of larger resources, that is to say large-coverage grammars, is also conditioned by the existence of such a parser. In this paper, we present our solution to this problem, namely an extension of the TuLiPA parser with frame semantics. We also present the frameworks used to build the resources used by the parser: the theoretical framework, composed of LTAG and frame semantics, and the software framework, XMG2.

Keywords: Parsing, Grammar, Syntax, Semantics, Tools, Systems, Applications.

1. Introduction

The development of linguistic resources such as precision grammars and lexicons is a complex and time consuming task. Even though the challenges offered by these tasks often come from the size of the resources to develop, the difficulty can be increased by the lack of tools processing this type of data and allowing to test it. This is the case of grammars using both the formalism of Lexicalized Tree Adjoining Grammars and frames as semantic representations. For example, the description of the syntax-semantics interface in (Zinova, 2017) uses a workaround (simulating the lexical insertion while creating the grammar, and not during parsing), as no parser for such a type of resource was available at the time. This paper introduces a parser which uses grammars of this type, based on an existing parser for LTAG: TuLiPA.

In Section 2., we provide a short explanation of the framework and show how it interacts with semantic parsing. In Section 3., we summarize the compact description of our resources in a metagrammar. In Section 4., we summarize the architecture of the TuLiPA parser that we are using. In Section 5., we introduce our new extension of TuLiPA to handle frame semantics, and give an example of use of this parser in Section 6..

2. Semantic Parsing with LTAG

2.1. Tree Adjoining Grammars

A Tree Adjoining Grammar consists of a set of elementary trees, from which larger trees are built using substitution and adjunction. Nodes in elementary trees have terminal or non-terminal labels. In a lexicalized TAG, every elementary tree has at least one terminal node. In a fully derived tree, all leaf nodes have terminal labels and all internal nodes have non-terminal labels. Elementary trees are either initial trees or auxiliary trees. The latter have one leaf node marked as a foot node by an asterisk. The foot node and the root node of the auxiliary tree have the same label. All trees may have leaf nodes that are marked as substitution nodes by an arrow. A derivation starts with an initial tree.

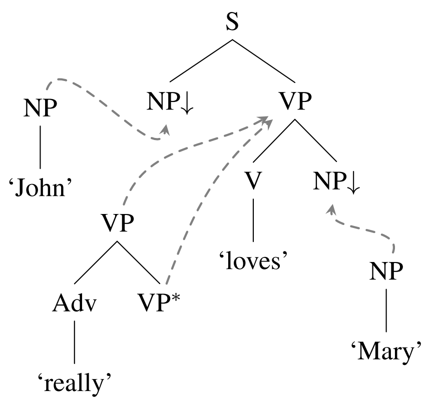
Two operations are used to combine trees during a derivation. In substitution, the root node of an elementary tree replaces a non-terminal leaf node of another tree, where both nodes have the same label. For adjunction, an auxiliary tree is adjoined to a target node in another tree. The root node of the auxiliary tree replaces the target node, and the part of the target tree below the target node is attached to the foot node in the auxiliary tree. An example is shown in Figure 1. In1a, the elementary trees for 'John' and 'Mary' are substituted at the subject and object argument slots in the initial tree for 'loves'. The adverbial modifier 'really' is adjoined at the VP node. The resulting derived tree is shown in 1b.

Elementary trees represent the constructional meaning of their lexical items in that they have substitution nodes for all arguments of the anchor. Recursive phenomena like modification are modelled by adjunction, which leads to long-distance dependencies in derived trees that were local in one elementary tree (Joshi and Schabes, 1997). Thus, all elementary syntactic structures can carry semantic information locally. Morphosyntactic information and links to the semantic representation are stored in feature structures at the nodes of elementary trees (Vijay-Shanker and Joshi, 1988). The acceptance of a derivation is determined by unification of those feature structures.

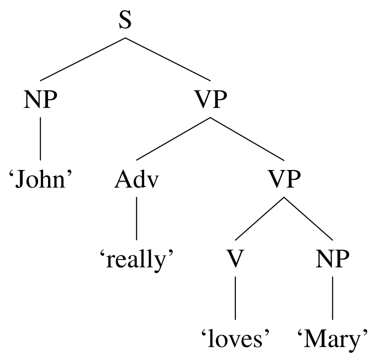
2.2. Frames and semantic parsing

In the syntax-semantics interface for LTAG proposed by (Kallmeyer and Osswald, 2013), semantic frames are represented as base-labelled, typed feature structures. They can be understood as a straightforward representation of the semantic and conceptual knowledge about a situation, while having good computational properties as their composition relies on the unification of attribute-value structures. LTAG elementary trees are paired with frames by using unification variables, as shown in Figure 2.

Here, the elementary tree for 'loves' is paired with a frame of type *love*, which has two attributes: an actor and a theme. This frame, labelled by e, is the semantic representation associated to the verb, therefore the variable e is shared with the feature structure of the syntactic node VP. The variables associated to the actor and the theme in the frame are shared with the two NP nodes of the syntactic tree (where they are values of the attribute I). The elementary trees for 'John'



(a) Elementary trees



(b) Derived tree

Figure 1: A derivation in LTAG.

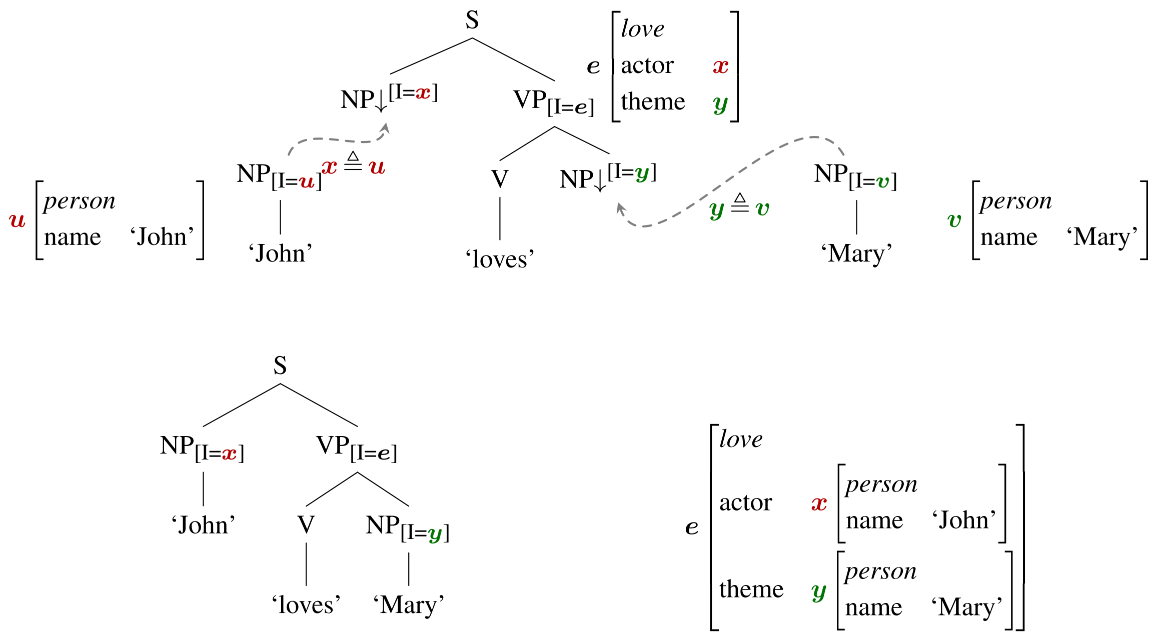


Figure 2: An example of pairing of LTAG trees and frames and the result of their combination.

and 'Mary' are both paired with frames of type *person*, where only the value of the attribute name differs, respectively labeled by the variables $\boldsymbol{u}$ and $\boldsymbol{v}$.

During parsing, as syntactic trees are combined (by adjunction or substitution), the semantic representations are also combined. The unification of variables in the feature structures associated to the nodes triggers the unification of variables in the frames. In our example, as the substitution of the subject NP takes place (combining the elementary trees of 'love' and 'John'), the respective values associated to the attribute I in the feature structures are unified. This results in the unification of the variables $\boldsymbol{x}$ and $\boldsymbol{u}$, which makes the frame for John become the actor of the event 'love'. The same happens when the tree for 'Mary' is substituted at the object NP node of the 'love' tree: $\boldsymbol{y}$ and $\boldsymbol{v}$ unify to let the frame for 'Mary' become the value of the theme attribute in the frame $\boldsymbol{e}$.

3. XMG and XMG2

For the electronic description of such resources, the approach proposed by (Kallmeyer and Osswald, 2013) is to use a metagrammar. The latter makes the development and maintenance of grammars easier by allowing abstractions. This is especially useful in LTAG grammars, as they show a lot of structural redundancy. XMG, for eXtensible MetaGrammar (Crabbé et al., 2013), permits the generation of grammars from fully declarative specifications (called metagrammars) which are based on logic programming and constraints. The name XMG stands both for the description language and the compiler for this language, that is to say the tool that will create a grammar from a metagrammatical description. Its newer evolution XMG2 (Petitjean et al., 2016) allows for more flexibility regarding the type of linguistic data to describe, for instance frames which were not initially supported. XMG2 is not a more complex descrip-

tion language and compiler, but a tool allowing to create new compilers of the kind of XMG. It offers a collection of different languages and compilers (automatically generated) dedicated to different description tasks. This set of description tools allows us to generate all the resources that we need in this paper using a single framework.
XMG comes with a system of dimensions which allows to separate the levels of linguistic description (here syntax and semantics, but also lexicon). The ⟨**syn**⟩ dimension allows to describe trees by using dominance and precedence constraints between syntactic nodes, while the ⟨**frame**⟩ dimension allows the description of typed feature structures, as well as type hierarchies (Lichte and Petitjean, 2015). The ⟨**lemma**⟩ and the ⟨**morph**⟩ dimensions can be used to generate a lexicon. The architecture of the lexicon that we use for our parsing task will be described in the Section 6., together with examples of XMG2 code using the four previsously mentioned dimensions.
An XMG2 compiler takes as input a metagrammar, and produces the lexicon of all the structures described in it. In other words, it converts a compact representation of a resource into the resource itself. In our case, every entry of the generated grammar is a pair of an unanchored tree and a typed feature structure. Grammars generated with XMG2 can naturally be used for tasks such as generation or parsing, provided that the adapted tool exists. The next section introduces TuLiPA, which is one of these tools.

4. TuLiPA

Mildly context-sensitive grammar formalisms like TAG have been shown to capture complex natural language phenomena, such as cross-serial dependencies, while being parsable in polynomial time. TuLiPA ("Tübingen Linguistic Parsing Architecture" (Kallmeyer et al., 2008)) is a parser for several mildly context-sensitive formalisms, including LTAG. The LTAG grammars used by the original TuLiPA version can feature semantic information using predicate semantics as in the syntax-semantics interfaces of Gardent and Kallmeyer (2003) and Kallmeyer and Romero (2008).
The lexical information processed by TuLiPA is 2-layered. The morphological lexicon maps inflected tokens to their lemma, storing morphological information in a feature structure. The lemmas are stored with semantic information and the tree families to which the lemmas can be anchored. Before parsing, trees are anchored, i.e. for every word of the input sentence, possible elementary trees are selected where the semantic information on the lemma of the word matches with the information on the anchor node of the tree.
Therefore, the grammar used by TuLiPA, similarly to the XTAG grammar (XTAG Research Group, 2001), is composed of three elements: a lexicon of unanchored elementary trees (tree templates), a lexicon of lemmas and a lexicon of fully inflected forms. This is another level of factorization (in addition to the metagrammatical one presented in Section 3.) which helps reducing the size of the resource. Currently, there are two parsing modes for parsing LTAG with semantic frames available in TuLiPA. As of (Kallmeyer et al., 2008), the input grammar is converted

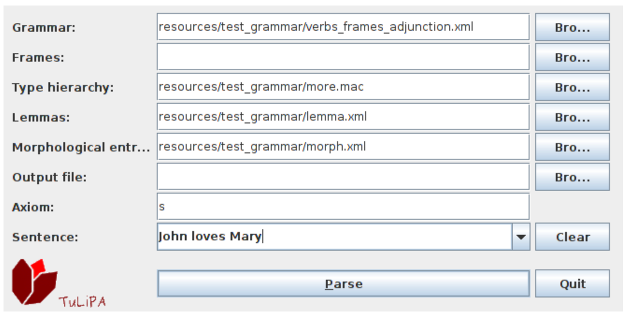

Grammar:	resources/test_grammar/verbs_frames_adjunction.xml
Frames:	
Type hierarchy:	resources/test_grammar/more.mac
Lemmas:	resources/test_grammar/lemma.xml
Morphological entr...	resources/test_grammar/morph.xml
Output file:	
Axiom:	s
Sentence:	John loves Mary

Figure 3: The user interface of TuLiPA

to a simple Range Concatenation Grammar (RCG, Boullier (1999)), which is used for parsing the input sentence, The TAG derivation structures are extracted from the RCG parsing results. Because this algorithm was designed to handle multi-component TAG (which is an extension of TAG), the conversion to RCG performed poorly on large-scale LTAG grammars. We included the implementation of a CYK parser by Thomas Schoenemann, based on the deduction rules given in (Kallmeyer, 2010). The parsing results, namely derivation trees, derived trees, derivation steps and possibly semantic representations, can be viewed in a graphical user interface or exported as XML files.
In the next section, we will describe how we extended TuLiPA to be able to process LTAG grammars including frame semantics.

5. A TuLiPA Extension for Frame Semantics

There are several motivations for the choice of TuLiPA as a starting point for our parser: with TuLiPA, we already have an open source LTAG parser, with graphical user interface, and multiplatform (as it is written in Java). As explained in the previous section, TuLiPA can already process semantic descriptions expressed as predicates.
Our extension, released like TuLiPA as an executable jar[1], accumulates typed feature structures instead of predicates during the parsing. The main changes are the following:

1. The grammars used are now composed of tree-frame pairs as presented in Section 2.. They are generated by XMG2, using the ⟨**syn**⟩ and the ⟨**frame**⟩ dimensions.

2. A type hierarchy (also produced by XMG2) must also be given to the parser to process the unification of typed feature structures.

3. The lexicon of trees and frames can be given separately. If this option is chosen, the tree-frame pairs are built during parsing (according to information provided by the lexicon).

Figure 3 shows the user interface of the parser where all the input data is given.
In the next section, we will go through all the steps which are necessary to create toy resources and recreate a parse example.

[1]`https://github.com/spetitjean/TuLiPA-frames`

```
 1 class commonnoun
 2 declare ?NP ?N ?X0 ?X1
 3 {
 4   <syn>{
 5     node ?NP [cat=np, i=?X0];
 6     node ?N (mark=anchor) [cat=n, i=?X1];
 7     ?NP -> ?N
 8   };
 9   <frame>{
10     ?X0[pizza]
11   }
12 }
```

Figure 4: XMG2 description for a proper noun elementary tree and its frame.

```
 1 class commonnoun
 2 declare ?NP ?N ?X0 ?X1
 3 {
 4   <syn>{
 5     node ?NP [cat=np, i=?X0];
 6     node ?N (mark=anchor) [cat=n, i=?X1];
 7     ?NP -> ?N
 8   };
 9     <iface>{[i=?X0]}
10 }
```

Figure 5: XMG2 description for a proper noun elementary tree.

6. Example of Parsing

To recreate the analysis of 'John eats pizza' similar to the one given in Figure 2, we will first see how to create the inputs needed by TuLiPA. The source code is available online, see 1. All the input files (tree templates, frames, lemma lexicon and morphological lexicon) are XML files produced by XMG2. The use of a single framework is a new feature: While the grammar and the frames were always produced by XMG, the morphological and lemma files used to be generated by a tool called lexConverter. The extensibility of XMG2 made it possible to create new compilers to generate these lexicons, using a consistent syntax and the same modular approach. In this work, we will use three different XMG2 compilers called `synframe`, `lex` and `mph`, which are accessible by the same command (`xmg compile`).

6.1. The grammar

The first resource to build is a LTAG grammar composed of at least 3 trees (transitive verb for 'eat', proper noun for 'John' and common noun for 'pizza') paired with their corresponding frames. These structures are described in the metagrammar and compiled using XMG2.

The XMG2 code describing a common noun tree and its frame can be as in Figure 4. Our architecture uses the notion of family, which are sets of trees which allow the same lexical anchors. In our example, the family `commonnoun` is composed of a unique tree, and the only lexical item compatible with this family is `pizza`.

```
 1 class FramePizza
 2 declare ?X0
 3 {
 4   <frame>{
 5     ?X0[pizza]
 6   };
 7   <iface>{
 8     [i=?X0]
 9   }
10 }
```

Figure 6: XMG2 description for a single frame, to be paired with a syntactic tree during parsing.

On the first line, **`class`** `commonnoun` means that we define an XMG2 abstraction, which is in fact the tree-frame pair as it will be in the grammar. `?NP ?N ?X0 ?X1` are unification variables used in the class. The tags **`syn`** and **`frame`** separate the syntactic and the semantic descriptions. In the syntactic one, two nodes (`?NP` and `?N`) are created with feature structures (`?NP` has category `np`, etc.). The node `?N` is marked as an anchor node. Finally, the constraint on line 7 means that XMG2 will only generate trees where the NP node has the N node as daughter.

On the semantic side, we create a typed feature structure of type `pizza`, labeled by the variable `?X0`. However, pairing the frame for a pizza with the TAG tree for a common noun is not a very natural solution. We will now split this entry in two different lexicons: a purely syntactic lexicon of unanchored elementary trees and a purely semantic lexicon of frames. The syntactic tree `commonnoun` will be associated to the semantic frame `pizza` only when this token is read by the parser. The lemma lexicon will take care of this binding, as explained in the following subsection. This allows to associate different frames to one elementary tree, giving more flexibility.

In our case, we can split syntax and semantics as shown in Figures 5 and 6. The linking between the syntactic node and the semantic frame is this time done through the interface (**`iface`**). This dimension contains only a feature structure, which allows to share information between other dimensions. During parsing, when the elementary tree and the frame are paired, the two interfaces are unified, resulting in the unification of the two variables named `?X1` (which were independent until now, as every structure has its own namespace). The two files (syntactic and semantic) are compiled using the `synframe` compiler to produce the two lexicons.

6.2. The lemmas

The second step is to create a lexicon of lemmas containing at least three entries (one transitive verb and two nouns). The entry for pizza is as shown in Figure 7.

Here, we specify the name of the lemma (`entry`), its syntactic category `cat` and the family of trees which can be anchored by it (`fam`), which is the one we created in the previous XMG2 syntactic class, namely `commonnoun`. The value of the `sem` feature indicates which frame this lemma should be associated to. The class `LemmaPizza` is part of

```
class LemmaPizza
{
  <lemma> {
    entry <- "pizza";
    sem   <- FramePizza;
    cat   <- n;
    fam   <- commonnoun
   }
}
```

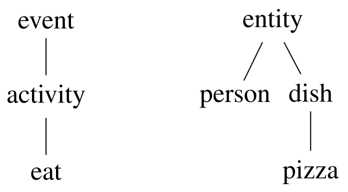
Figure 7: XMG2 description for the lemma ’pizza’.

```
class MorphPizza
{
  <morpho> {
    morph <- "pizza";
    lemma <- "pizza";
    cat   <- n;
    num   <- sg
   }
}
```

Figure 8: XMG2 description for the morphological entry for ’pizza’.

```
1  frame-types = {event, activity, eat,
2          entity, person, dish, pizza}
3  frame-constraints = {
4          activity -> event,
5          entity event -> -,
6          eat -> activity,
7          person -> entity,
8          dish -> entity,
9          dish person -> -,
10         pizza -> dish }
```

(a) Implementation in XMG2



(b) Graphical representation

Figure 9: The implementation and graphical representation of the type hierarchy used in ’John eats pizza’

a new XMG2 file, which must be compiled with the `lex` compiler.

6.3. The inflected forms

Finally, we must write a lexicon of at least three inflected forms (’John’ for the lemma ’john’, etc.). The entry for ’pizza’ (one of the inflected forms of the previously defined lemma, along with ’pizzas’ for example) is shown in the code of Figure 8. This entry associates the inflected form ’pizza’ to the lemma `pizza`, with additional features, here only a syntactic category and a number. The XMG2 file containing this class amongst others can be compiled with the `mph` compiler to produce the lexicon of inflected forms.

6.4. The type hierarchy

The implementation of the type hierarchy in Figure 9a follows Lichte and Petitjean (2015). In the code, line 1 and 2 define the elementary types that are used in the hierarchy. From line 3 on, constraints are declared on these elementary types. Constraints in the form of line 4 express type subsumption: every `activity` is also an `event`. Constraints as in line 5 express that the unification of elementary types fails: When a minimal model of the type hierarchy is computed, `entity` and `event` do not have a common subtype. An example is provided in the next section. It is also possible to express constraints on the attributes of frames with a certain type. For a more detailled description, see (Lichte and Petitjean, 2015). The graphical representation in Figure 9b has to be read top-down, with the most general types on top.

6.5. The parse result

The result of the parse of the sentence ’John eats pizza’, as shown by the graphical interface of the parser, is given in Figure 10.

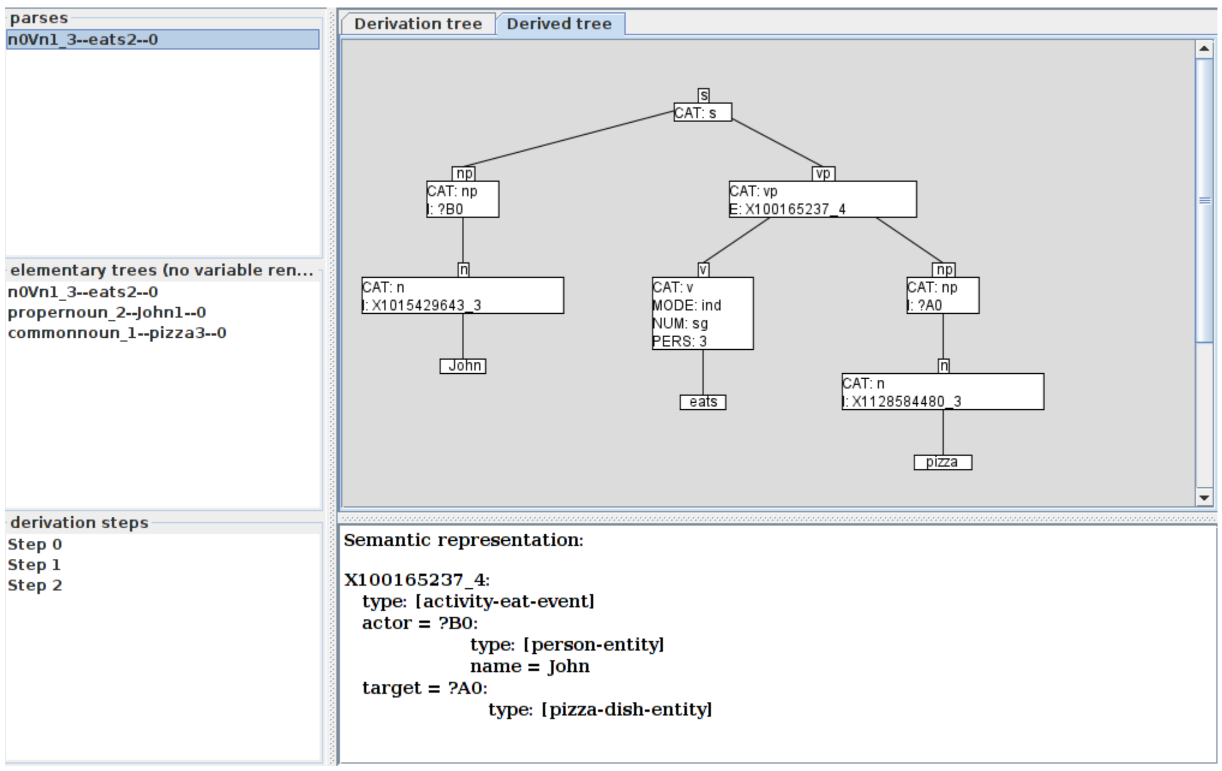
On the left, we see the list of successful parses (only one), the set of elementary trees which were used to derive the selected parse, and the derivation steps. The derived tree is shown at the top right, with the semantic representation at the bottom.

The parse result is similar to the one which we gave in Figure 2. The semantic representation consists of one frame: It has type `activity-eat-event` and two attributes, the `actor` and `target` of the verb. Their values are shared with the syntax: The actor is of type `person-entity` and is unified with the interface feature in the subject-`np`-node through the variable `?B0`. The target is of type `pizza-dish-entity` and corresponds to the object-`np`-node, unified by the variable `?A0`. These unifications are triggered by substitution, and the insertion of ’john’ into the frame is triggered by lexical anchoring. Note that the type `dish`, which is not present in the `eat` frame nor in the `pizza` frame, is inferred from the type hierarchy (as every frame of type `pizza` must also have type `dish`). The interface feature of the subject-`n`-node give an idea of this. Unification via adjunction works in the same way.

The types of the feature structures are here conjunctive types and get modified by the constraints expressed in the metagrammar. For instance, one constraint in our metagrammar specifies that all structures of type `person` also have type `entity`, hence the conjunctive types `[person-entity]`.

7. Conclusion

In this paper, we presented our parser for LTAG and frame semantics. The parser is an extension of TuLiPA and is consequently based on the same architecture, meaning that the grammar is separated into several levels: a lexicon of unanchored elementary trees paired with frames (provided



parses
n0Vn1_3--eats2--0

elementary trees (no variable ren...
n0Vn1_3--eats2--0
propernoun_2--John1--0
commonnoun_1--pizza3--0

derivation steps
Step 0
Step 1
Step 2

```
Semantic representation:

X100165237_4:
  type: [activity-eat-event]
  actor = ?B0:
              type: [person-entity]
              name = John
  target = ?A0:
                type: [pizza-dish-entity]
```

Figure 10: Result of the parse of 'John eats pizza'

together or separately), one of lemmas and one of inflected forms. The composition of frames, implemented following Kallmeyer and Osswald (2013), happens as elementary trees are combined, triggered by the unification of linked variables. In Section 6., we showed how to create the different resources to parse a simple example, and also that the analysis done by our tool for this example was the expected one. The availability of such a tool will make the use of the existing LTAG grammars including frame based semantics possible, and ease the creation of new ones.

As a next step, in order to improve parsing efficiency when using large grammar resources, we consider implementing grammar compression using subtree sharing and compression of these subtrees into minimal Finite State Automata, as described in (Waszczuk et al., 2016)

We are also working on a parser for Role and Reference Grammar (RRG, Van Valin (2005)), following the formalization proposed in (Kallmeyer and Osswald, 2017). RRG is a grammar theory based on flat constituent structures and focusing on semantic and pragmatic aspects. We can of course imagine that grammars based on RRG and frame semantics will be created, with a framework similar to the one presented in this paper, and tested with our tool.

8. Acknowledgements

The research presented here has been supported by the TreeGraSP Project funded by an ERC Consolidator Grant. We would like to thank three anonymous reviewers for their valuable comments on earlier drafts of this article.

9. Bibliographical References

Boullier, P. (1999). On TAG and Multicomponent TAG Parsing. Research Report RR-3668, INRIA.

Crabbé, B., Duchier, D., Gardent, C., Le Roux, J., and Parmentier, Y. (2013). XMG : eXtensible MetaGrammar. *Computational Linguistics*, 39(3):1–66.

Gardent, C. and Kallmeyer, L. (2003). Semantic construction in feature-based tree adjoining grammar. In *10th conference of the European Chapter of the Association for Computational Linguistics*.

Joshi, A. K. and Schabes, Y. (1997). Tree-Adjoining Grammars. In G. Rozenberg et al., editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin, New York.

Kallmeyer, L. and Osswald, R. (2013). Syntax-driven semantic frame composition in lexicalized tree adjoining grammars. *Journal of Language Modelling*, 1(2):267–330.

Kallmeyer, L. and Osswald, R. (2017). Combining predicate-argument structure and operator projection: Clause structure in role and reference grammar. In *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms*, pages 61–70. Association for Computational Linguistics.

Kallmeyer, L. and Romero, M. (2008). Scope and Situation Binding in LTAG using Semantic Unification. *Research on Language and Computation*, 6(1):3–52.

Kallmeyer, L., Lichte, T., Maier, W., Parmentier, Y., Dellert, J., and Evang, K. (2008). Tulipa: Towards a multi-formalism parsing environment for grammar engineering. In *Coling 2008: Proceedings of the workshop* *on Grammar Engineering Across Frameworks*, pages 1–8, Manchester, England, August.

Kallmeyer, L. (2010). *Parsing Beyond Context-Free Grammars*. Springer Publishing Company, Incorporated, 1st edition.

Lichte, T. and Petitjean, S. (2015). Implementing semantic frames as typed feature structures with XMG. *Journal of Language Modelling*, 3(1):185–228.

Petitjean, S., Duchier, D., and Parmentier, Y. (2016). XMG 2: Describing Description Languages. In *Logical Aspects of Computational Linguistics. Celebrating 20 Years of LACL (1996–2016) 9th International Conference, LACL 2016, Nancy, France, December 5-7, 2016, Proceedings 9*, pages 255–272. Springer Berlin Heidelberg.

Van Valin, Jr., R. D. (2005). *Exploring the Syntax-Semantics Interface*. Cambridge University Press.

Vijay-Shanker, K. and Joshi, A. K. (1988). Feature structures based tree adjoining grammars. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 2*, COLING '88, pages 714–719, Stroudsburg, PA, USA. Association for Computational Linguistics.

Waszczuk, J., Savary, A., and Parmentier, Y. (2016). Enhancing practical TAG parsing efficiency by capturing redundancy. In *21st International Conference on Implementation and Application of Automata (CIAA 2016)*, Séoul, South Korea, July.

XTAG Research Group. (2001). A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

Zinova, J. (2017). *Russian Verbal Prefixation: A Frame Semantic Analysis*. Heinrich-Heine-Universität Düsseldorf dissertation.