Glue semantics for Universal Dependencies

Matthew Gotham and Dag Haug, University of Oslo

Introduction In recent years, the Universal Dependencies initiative [8] has established itself as a *de facto* annotation standard for cross-linguistic annotation of syntactic structure (treebank development) and subsequent statistical parsing with models trained on those treebanks. However, many downstream tasks require not dependency parses but rather more elaborate semantic structures that must be derived from the parses. The challenge, then, is to do this while relying as little as possible on language-specific, typically lexical resources that are not available for many of the 60 languages for which there are UD treebanks.

In this paper we outline an approach to this problem that builds on techniques developed for LFG + Glue. There are several motivations for this: First, LFG's f-structures track the same aspect of syntactic structure as UD dependency trees. Second, the particular version of dependency grammar that UD embodies has inherited much from LFG via the Stanford Dependencies and the PARC dependencies. Third, unlike many other approaches, LFG + Glue does not assume a one-to-one mapping from syntactic to semantic structures but instead develops a syntax-semantics interface that can map a single syntactic structure to several meaning representations, i.e. the syntax underspecifies the semantics.

The latter point becomes especially important because UD – for all its LFG inheritance – is a compromise between theoretical concerns such as language-specific and typological adequacy on the one hand, and computational concerns such as efficient annotation and reliable parsing on the other hand. A typical UD tree therefore contains much less information than the ideal syntactic representations assumed in theoretical formal semantics. In particular – disregarding the enhanced dependencies that are only available for a small subset of the UD treebanks – UD structures always form a rooted tree over the tokens of the sentence. This means that every node corresponds to an overt token of the sentence (the overt token constraint) and has exactly one mother (the single head constraint), unless it is the root, in which case it has no mother.

Both the single-head constraint and the overt token constraint limit the expressivity of the syntactic formalism in a way that impairs meaning construction. Figure 1 shows the UD annotation of a typical (object) control structure. The single head constraint makes it impossible to express that *the dog* is simultaneously the object of *persuaded* and the subject of *to bark*, and the overt token constraint makes it impossible insert any other subject for *bark*, say a PRO that could be coindexed with *the dog*. For similar reasons, the annotation of relative structures without a relativizer (relative pronoun or completizer) shown in Figure 2 does not indicate the position of the gap in the relative clause *they thought we admired*.

Approach We generate usable semantic representations in four steps.

1. non-deterministic tree rewriting in tandem with creation of meaning constructors for each generated tree
2. construction of linear logic proofs and corresponding semantic representations
3. enrichment of semantic representations using lexical resources
4. task-specific grounding of semantic representations e.g. to some knowledge base

This paper focuses on the first two steps, i.e. the language-independent, compositional semantic process which is the foundation for steps 3 and 4, but also has considerable interest in itself as extending the glue semantics framework to another syntactic backbone in addition to LFG [4], LTAG [5], HPSG [2] and Minimalism [6].

Linear logic fragment We use a propositional linear logic [1] with a single connective $\multimap$ and (undefined) propositional functions v, t and e whose domain is the nodes of the (rewritten) UD graph. These are intended as mnemomnic for the types on the lambda side (*ev*ent, *t*ruth value, *e*ntity) and every proposition we use will result from one of these functions. For example, *dog* in Figure 1 will have the type $e(4) \multimap t(4)$.

Generation of meaning constructors proceeds by a depth-first traversal of the UD tree. Each visited node is matched against a file with criteria and for each fulfilled criterion a set of pairs of meaning constructors and modified trees is created, one for each possible instantiation of the corresponding meaning constructor. Each of these pairs are passed to the next rule and once there are no more rules, we pass to the next node in each of the trees and repeat the process. When all nodes of all trees have been visited, we are left with a set of pairs $\langle T, M \rangle$ where T is a possibly modified tree and M is a multiset set of meaning constructors.

The rules construct meaning types in a two-step process: first we construct an 'end type' based on the node itself, e.g. $(v(\downarrow) \multimap t(\downarrow)) \multimap t(\downarrow)$ (the Montague lift of an event type meaning) for verbs. To this type we add implications based on the daughter nodes. For example, an *nsubj* daughter triggers a dependency on $e(\downarrow nsubj)$. In this way we generate e.g. the instantiated type $e(2) \multimap (v(7) \multimap t(7)) \multimap t(7)$ for *barks* in Figure 2.

Additional complexity arises when the paths can be instantiated in more than one way. (1) shows our rule for the indefinite determiner (with $\uparrow$ pointing to the mother node). As in [1] we use a functional uncertainty rather than linear logic quantification to express the possible scoping sites for the quantifier.

(1) $\quad (e(\uparrow) \multimap t(\uparrow)) \multimap (e(\uparrow) \multimap t(\%h)) \multimap t(\%h) : \%h = \uparrow\uparrow^*$

Finally, when there is *no* way to instantiate a functional uncertainty, the rule will add the necessary nodes to the tree (with the restriction that, as in LFG, nodes under a Kleene star are never created). (2-a) gives the

rule for the *acl:relcl* relation which marks the verb of a relative clause, which gets instantiated as (2-b) for the example in Figure 2, where *thought* bears the *acl:relcl relation* and so instantiates $\downarrow$.

(2) a. $(e(\downarrow dep^* dep\{PronType = Rel\}) \multimap (v(\downarrow) \multimap t(\downarrow)) \multimap t(\downarrow)) \multimap (e(\uparrow) \multimap t(\uparrow)) \multimap e(\uparrow) \multimap t(\uparrow)$
b. $(e(9) \multimap (v(4) \multimap t(4)) \multimap t(4)) \multimap (e(2) \multimap t(2)) \multimap e(2) \multimap t(2)$

(2-b) will show up paired with trees where the node 9 has been added at various possible extraction sites.

Meaning representation We use an updated version of PCDRT [7] as our meaning representation language, because (i) it is defined in typed lambda calculus, and hence is straightforwardly compatible with Glue, (ii) it has a treatment of unresolved anaphora, which is essential for an adequate meaning representation for many naturally-occurring examples such as are collected in treebanks, and (iii) it is representationally similar to standard DRT, allowing for comparison with computational linguistic resources prepared on the basis of DRT. We deviate from standard PCDRT in having propositional discourse referents (marked p_n) for sentential embedding. Presupposed discourse referents are marked with the predicate *ant*, and presupposed conditions with the propositional function ∂, which maps TRUE to TRUE and is otherwise undefined.

Meanings for nodes and edges are constructed in parallel with the types. For example, giving a verb the end type $(v(\downarrow) \multimap t(\downarrow)) \multimap t(\downarrow)$ (as described above) corresponds to constructing the meaning representation $\lambda V.[e \mid head(e)]; V(e)$ (where *head* is the lemma of the verb), and when a dependent d triggers a dependency on $e(\downarrow d)$, that transforms the meaning representation to $\lambda x.\lambda V.[e \mid head(e), d(e,x)]; V(e)$. For example, *barks* in Figure 2 has the representation shown in (3).

(3) $\lambda x.\lambda V.[e \mid bark(e), nsubj(e,x)]; V(e) : e(2) \multimap (v(7) \multimap t(7)) \multimap t(7).$

(3) shows as much information as can be extracted from (this portion of) the UD tree alone. The thematic relation *nsubj* can be understood as a very general relation, to be further specified by means of lexical knowledge, which can be encoded in meaning postulates such as the one shown in (4).

(4) $\forall e \forall x((bark(e) \wedge nsubj(e,x)) \rightarrow agent(e,x))$

Lexical knowledge will also play a role in the specification of general thematic relations. For example, what has happened in the composition of the meaning shown in Figure 1 is that the meaning constructor contributed by the xcomp arc has (i) introduced an *xcomp* relation between the persuading event e_1 and the proposition p_6 that there is a barking event e_4, and (ii) introduced an individual x_5 as both the *nsubj* of e_4 and the *control_dep* of e_1. Further refinement of *control_dep* in this case depends on lexical knowledge, namely that 'persuade' is an object control verb. This can be seen as amounting to knowing the meaning postulate shown in (5).

(5) $\forall e \forall x((persuade(e) \wedge control\_dep(e,x)) \rightarrow dobj(e,x))$

Further refinement of *dobj*—presumably to *theme* in this case—depends on lexical knowledge about 'persuade'.

Implementation Meaning representations are extracted from UD graphs according to the pipeline shown in Figure 3. As described above, for any input UD tree our software outputs a set of multisets of meaning constructors. Each of those multisets can then serve as the premises to be input into in a linear logic prover, and depending on the premises there may be one or more proofs. The meaning term associated with each proof is then input into a reduction engine for λ-DRT, and the result is a DRS for each proof. With lexical resources available, these DRSs can be rejected or enriched.

Discussion Unlike other approaches (e.g. [9, Ch. 5]), we avoid the need to binarize the dependency tree for semantic composition, but instead capitalize on the glue approach to composition and thereby derive different scopal readings. We also make more use of the typing system, which e.g. is used to restrict unwanted scopings. For example, lifting the end type of verbs to $(v \multimap t) \multimap t$ stops arguments from scoping below the event (yielding nonsense readings such as 'there was an event of no-one running'). At the same time, the typing system remains robust because it is deduced from UD tree rather than from an external lexicon. Finally, we put to use standard techniques from LFG such as functional uncertainties to derive different readings for bare relative clauses. In sum, we believe we have successfully constructed a theoretical framework for doing semantics off dependency structures. Because of its theoretical focus, our work so far has only been evaluated on a small development suite. In future work, we plan to empirically evaluate our approach on standard datasets, thereby testing not only the UD + Glue framework, but also the adequacy of UD representations for computational semantics.

References

[1] A. D. Andrews. "Propositional glue and the projection architecture of LFG". In: *L & P* 33.3 (2010). [2] A. Asudeh and R. Crouch. "Glue Semantics for HPSG". In: *HPSG 2002 Proceedings*. Ed. by F. van Eynde, L. Hellan, and D. Beermann. Stanford, CA: CSLI Publications, 2002. [3] P. Blackburn and J. Bos. "Working with Discourse Representation Theory". Unpublished book draft. 2006. [4] M. Dalrymple, J. Lamping, and V. Saraswat. "LFG Semantics via Constraints". In: *EACL 6*. Stroudsburg, PA, USA: ACL, 1993. [5] A. Frank and J. van Genabith. "GlueTag: Linear Logic-based Semantics for LTAG—and what it teaches us about LFG and LTAG". In: *LFG 2001 Proceedings*. Ed. by M. Butt and T. H. King. Stanford, CA: CSLI Publications, 2001.

[6] M. Gotham. “Making Logical Form Type-Logical. Glue Semantics for Minimalist syntax”. In: *L & P* (2018). In press.
[7] Dag Trygve Truslew Haug. “Partial Dynamic Semantics for Anaphora. Compositionality Without Syntactic Coindexation”. In: *JoS* 31 (2014), pp. 457–511. [8] M. Marneffe et al. “Universal Stanford dependencies: A cross-linguistic typology”. In: *LREC*. Reykjavik, Iceland: ELRA, 2014. [9] S. Reddy. “Syntax-Mediated Semantic Parsing”. PhD thesis. University of Edinburgh, 2017.

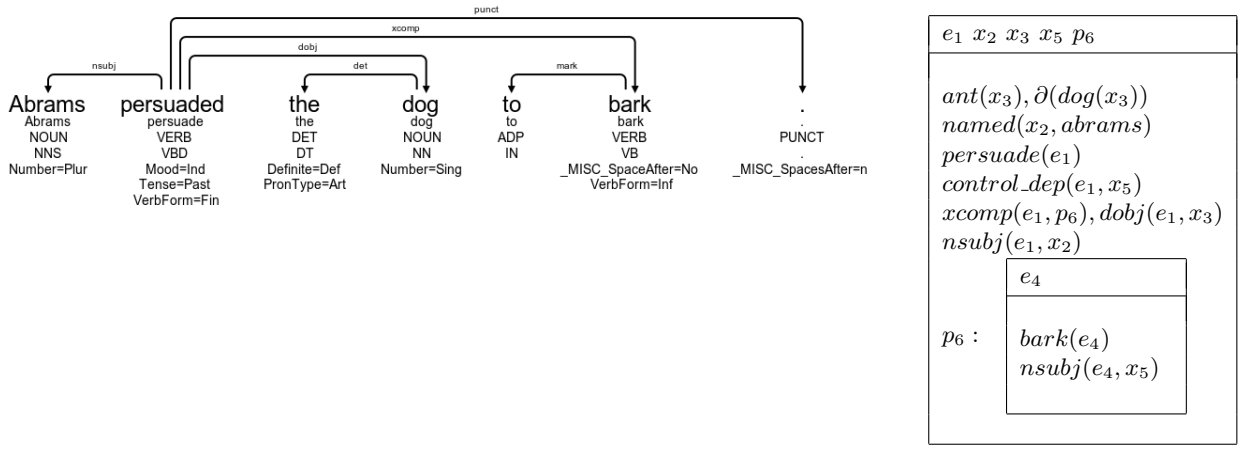


Figure 1: UD control structure and (unique) system output with unmodified tree

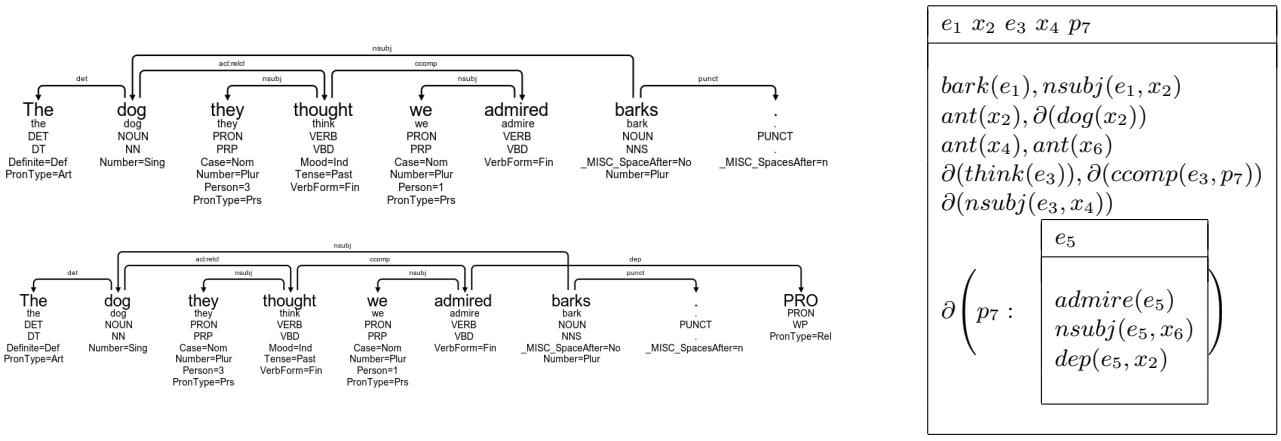
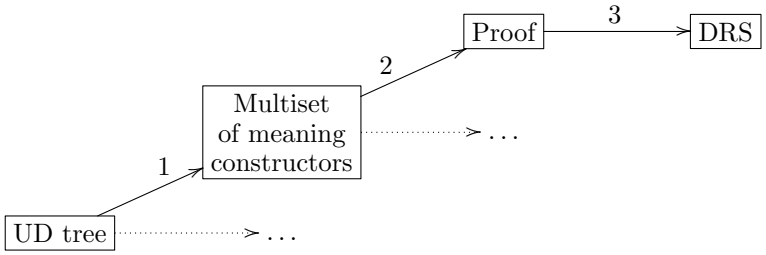


Figure 2: UD relative clause structure and (sample) system output with modified tree



1. Depth-first traversal of UD tree, described in the section 'Linear logic fragment'.
2. *Instant Glue* prover, written by Miltiadis Kokkonidis with the support of SSHRC SRG #410-2006-1650 to Ash Asudeh and available at `http://users.ox.ac.uk/~cpgl0036/prover/glue_prover.pl` (accessed 2018-02-13).
3. Beta reduction software for λ-DRT [3].

Figure 3: The pipeline. The dotted arrows indicate that depending on the input, steps 1 and 2 may output more than one result. Step 3, however, is functional.