Deep Semantics for Dependency Structures

Paul Bedaride, Claire Gardent

Deep Semantics for Dependency Structures

Paul Bédaride[1] and Claire Gardent[2]

[1] Universität Suttgart paul.bedaride@loria.fr
[2] CNRS/LORIA claire.gardent@loria.fr

Abstract. Although dependency parsers have become increasingly popular, little work has been done on how to associate dependency structures with deep semantic representations. In this paper, we propose a semantic calculus for dependency structures which can be used to construct deep semantic representations from joint syntactic and semantic dependency structures similar to those used in the ConLL 2008 Shared Task.

Key words: Dependency graphs, Deep Semantics, Graph Rewriting

1 Introduction

Deep semantics have been developed for stochastic categorial parsers (1) and for parsers based on phrase structure grammars (2; 3; 4). Much less work has been done, however, on combining dependency parsers with a a deep semantics calculus. Although (5) sketches a syntax-semantics interface for dependency grammar, the proposed approach requires a constraint-based, tightly interleaved construction of dependency, predicate/argument and scoping structure which is not easily adaptable to the output of contemporary dependency parsers. Similarly, (6) presents a formalism for semantic construction from dependency structures. However, the approach incorrectly assumes that semantic dependencies match syntactic dependencies and so fails to generalise (cf. Section 2).

In this paper, we present an approach for rewriting dependency graphs into deep semantic representations that can be applied to joint syntactic and semantic dependency structures similar to those used in the ConLL 2008 Shared Task. We start by discussing a number of issues raised by dependency structures in relation to semantic construction and by motivating the choices underlying our approach (Section 2). We then present our proposal (Section 3).

2 Motivations

In essence, a dependency structure consists of nodes labelled with lexical items (and optionally, parts-or-speech) and linked by binary asymetric relations called dependencies. Figure 5 illustrates this with the plain (non bold) nodes and edges forming a possible dependency graph for the sentence "John seems to love Mary".

Importantly, dependency structures differ from phrase structure trees or categorial derivations in that they describe relations between words and eschew the notions of syntactic constituents and non terminal syntactic categories. Starting with (7) however, a key assumption is that the computation of deep semantics strongly relies on syntax. Thus in phrase structure grammars, each syntactic

rule is coupled with a semantic rule specifying how the semantics of its daughters combines to yield the semantics of the constituent being derived. Similarly, in categorial grammars, each word is simultaneously assigned a syntactic and a semantic category describing both how it syntactically combines with other word/category pairs and how its semantics combine with the semantics of the items it combines with. In essence, syntax guides semantic construction in that it constrains the semantic type[3] of word occurrences and specifies how the semantics of constituents combine to yield the semantics of derived constituents. Given this, dependency graphs raise two main issues with respect to semantic construction.

First, the impoverished syntactic categories they include make it difficult to determine the semantic type of a given word occurrence. For instance, given the two sentences in (1), there is no obvious way to determine from their dependency graphs (shown in Figures 5 and 6) that the semantic functor licensed by "seems" combines with a VP semantics in (1a) but with a sentence semantics in (1b).

(1) a. John seems to love Mary
    b. It seems that John loves Mary

The problem is that, in both cases, the syntactic category associated with "seems" is a simple part-of-speech category which fails to indicate the syntactic and hence the semantic type of the verb arguments. To put it another way, there is no indication in a dependency graph of which syntactic type of "seem" is used to build each of the two sentences. To determine that "seems" combines with an infinitival VP in (a) but with a sentential argument in (b), dependencies that are non local to "seems" would need to be checked e.g., Does "love" in the dependency graph dominate a "to" or a "that" node ?

A second issue regarding semantic construction from dependency graphs is that syntactic and semantic dependencies do not necessarily match (3). In particular, there is sometimes a mismatch between predicate/argument and scope relations. For instance, in questions such as (2), "which man" scopes over the rest of the sentence to yield the meaning *Which is the x such that x is a man and John thinks that Mary likes x ?* Standard dependency structures fail to capture the scope of the wh-element because "man" is related by an object relation to "likes" but not to the main verb "thinks".

(2) Which man does John think that Mary likes?

In sum, the combined lack in a dependency graph, of a fully fledged syntactic categorial system and of a syntactic structure makes it difficult both to determine the semantic type of a word occurrence (Does "seems" combine with a VP or an S semantics?) and to appropriately describe how meanings should combine (How can both scope and predicate/argument relationships be appropriately captured?). To address these issues, we propose an approach to semantic construction which does not rely on a strict syntax/semantic parallelism but

[3] Here and in what follows, the term "semantic type" refers to the logical type of the denotation of natural language expressions e.g., in an extensional typed lambda calculus the type t of sentences or the type *((e,t),(e,t),e)* of quantifiers with e the type of individuals and t the type of truth values.

constructs semantic representations based on a small set of general principles describing the syntax-semantic interface. These principles are encoded in graph rewriting rules which determine the semantic type of each word occurrence based on the graph configuration in which it occurs. We show how this approach handles the cases above and a range of various other semantic phenomena.

3 Proposal

We start (Section 3.1) by briefly introducing graph rewriting and discussing termination, confluence and well-formedness. We then describe our approach to semantic construction (Section 3.2) and illustrate its working by describing the derivation of "Every man loves a woman" (Section 3.3). We then go on to sketch how to handle control, raising, modifiers and relative clauses (Section 3.4).

3.1 Graph Rewriting

Used in e.g., formal calculus, combinatoric algebra and operational semantics, rewriting is a technique for modelling reduction and simplification. For instance, the rewriting rule $r_1 : x*y+x*z \rightarrow x*(y+z)$ permits factorising $5*6+5*7+5*8$ to $5 * ((6 + 7) + 8)$. More generally, a rewriting system consists of a set of rewriting rules of the form $l \rightarrow r$ where l and r are filtering and rewriting patterns respectively. Given a graph g, such a rule will apply to g if g matches the filtering pattern l. The result of applying a rule to a graph g is g where the sub-part of g matched by l is rewritten according to the rewriting pattern r. Matching consists in looking for a homomorphism between the pattern graph l and the host graph g while the allowed rewriting operations include information duplication, deletion and addition[4].

GrGen, a standard graph rewriting system To define our rewrite rules, we use an existing rewriting system called GrGen (9). In GrGen, the objects handled by rewriting are directed graphs with typed nodes and edges. Each node and each edge has a type. Additionally, nodes can be associated with a set of attribute value pairs constrained by the node type. In the filtering pattern, attribute value pairs are interpreted as constraints while in the rewriting pattern, they are interpreted as assignments. Finally, nodes names can be used to constrain the mapping between filtering and rewriting pattern in that two nodes with the same names must be identical.

Expressive and efficient, GrGen[5] is well suited to specify our semantic construction rules. For instance, the rewrite rule graphically depicted in Figure 1b can be specified as shown in Figure 1a. In essence, the rewrite rule expands the

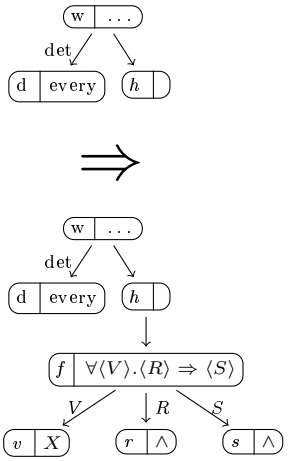
[4] For a precise definition of matching, we refer the reader to (8).

[5] There are other rewriting systems available such as in particular, the Tsurgen system used in the Stanford Parser to map parse trees into dependency graphs. We opted for GrGen instead because GrGen is efficient, notationally expressive (for specifying graphs but also rules and rule application strategies) and comes with a sophisticated debugging environment.

seed node h[6] licensed by an "Every N" dependency subgraph, with a semantic subgraph capturing the corresponding semantic type namely, the type of a universal quantifier.

```
rule forall {
 pattern{
  w:word; d:word; h:sem;
  w -det-> d;
  w --> h;
  if {d.word=="every"}
 }
 replace{
  w -det-> d;
  w --> h;
  f:sem; v:sem; r:sem; s:sem;
  eval{
    v.var = "X";
    f.formula = "";
    r.formula = "∧";
    s.formula = "∧";
  }
  h --> f;
  f -V-> v;
  f -R-> r;
  f -S-> s;
 }
}
```

(a) GrGen Rule



(b) Graphical representation

Fig. 1: A rewrite rule which expands the seed node licensed by an "Every N" subgraph with a semantic subgraph encoding the corresponding semantic type namely, the type of a universal quantifier. Here and in what follows, we use the following graphical conventions. Node names (e.g., w, d, h, f, q, r) appear to the left of the vertical bar splitting a node description while attribute values (every, $X, \wedge$) appear to its right. Attribute names are omitted. Node types are indicated using different fonts whereby italics indicate a node in the semantic representation structure, a plain font a node in the dependency structure and a bold font a node in the SRL structure. Edge type is not represented but is deducible from the types of the in and out vertices.

Confluence, termination and well-formedness The standard approach to semantic construction typically relies on the typed lambda calculus to define and combine meaning representations. This ensures termination (through the typing system), confluence (all ways of combining the same sequence of lambda terms yield the same result) and well-formedness of the resulting formulae (beta-reduction will fail in case of a type clash).

[6] See Section 3.2 for an explanation of how seed nodes are introduced.

These properties are not garanteed by rewriting. Indeed it is easy to define a non confluent rewriting system that yields ill formed semantic representations. We avoid those pitfalls as follows. We ensure termination and confluence by imposing a total order on rule application. As we shall see in the following section, each rule captures a general semantic construction principle. The order imposed on their application captures the way in which these principles interact (e.g., scope can only be defined after quantifiers and their semantic arguments have been introduced). Further, well formedness is ensured by the translation from semantic graphs to FOL formulae which will fail in case a FOL formula cannot be reconstructed from a given constructed graph.

3.2 Basic semantic construction procedure

Our semantic calculus is semantic rather than syntax driven. Drawing on the global dependency analysis of a sentence, it incrementally constructs a semantic representation by building, linking and labelling the various substructures composing this representation.

To simplify semantic construction, we additionally assume that the dependency graphs we take as input are enriched with semantic role labelling (SRL) information. This permits abstracting over syntactic idiosyncrasies such as active-passive alternations or dative shifts, and making certain semantic dependencies e. g. in control constructions explicit. The dependency graphs are produced by the Stanford parser (10) and augmented with Propbank style semantic role labelling information as described in (11). Given such joint structures, we use rewrite rules to further extend them with a semantic representation. Here, we illustrate the approach by showing how to build first order logical formulae but nothing hinges on this and other types of semantic representations could be built such as e.g., Discourse Representation Structures (DRSs) or Minimal Recursion Semantics structures (MRSs).

Semantic construction is modelled by a rewriting system consisting of six general syntax-semantic principles implemented as a set of cascaded rules applying in a fixed order, each rule taking as input the output of the previous step. The underlying intuition is as follows. First, "seed nodes" are created by adding as many nodes to the semantic representation as there are words pointed to by semantic role labelling edges. That is, a node is created for each predicate and each argument in the SRL structure. Second, each seed node is expanded with the subformula skeleton representing its meaning. Nominal arguments are expanded with a generalised quantifier tree shape while predicates (verbs or deverbal nominals) are expanded with an existential quantification over eventuality variables. Third, scope is determined by linking the resulting trees together. Fourth, nodes are labelled with the appropriate predications whereby variables are bound by the appropriate operator.

In sum, semantic construction initialises a structure (seed nodes creation), expands it with structures representing the semantic type of each node (node expansion), determines scope by linking these substructures together (scoping) and finalises the resulting structure by labelling nodes with the appropriate literals and bound variable (node labelling, variable binding). Additional principles are implemented for modelling connectives such as "when" or "if-then", which are not discussed here because of space restrictions.

3.3 Run through example

We start by giving a bird eye view of the semantic construction process for the sentence "Every man loves a woman". In the next section, we will show in more detail how the rewrite rules permit appropriately mapping syntax to semantics and more particularly, how they ensure that variables are appropriately bound.

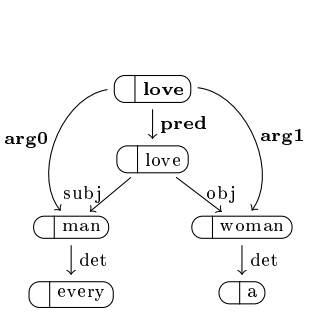
The joint dependency+SRL graph input to semantic construction is shown in Figure 2a. The first step (2b) creates three seed nodes each of which is licensed by an SRL node: the n_0 node is licensed by the predicate node associated with the verb "loves" and the n_1, n_2 nodes are licensed by the two verb argument heads "man" and "woman" respectively.

The second step expands these seed nodes to build substructures describing their semantic type. Seed nodes that are licensed by a dependency node with nominal category dominating a determiner node licence the construction of a subtree representing a generalised quantifier i.e., a tripartite structure consisting of a quantifier, a restriction and a scope where the quantifier will be determined by the specific determiner dominated by the noun (e.g., a universal for "every" or "all" and an existential for "a"[7]). In contrast, seed nodes licenced by a predicate (e.g., a verb or a deverbal nominal) trigger the construction of a structure representing an existentially bound event variable. The node expansions licensed by "Every man", "loves" and "A woman" respectively are shown in Figure 2c.

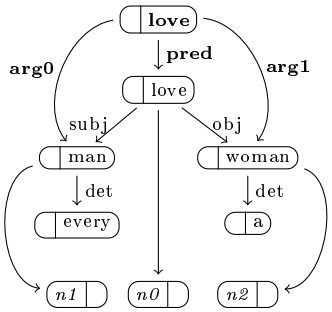
The third step (Figure 2d) connects the substructures built so far thereby determining scope. Scope is specified by adding an edge between the scoping node of each scope bearing operator and the head of the semantic substructure licensed by the next syntactic argument in the sentence (e.g., by adding a link from the scoping node of "every man" to the root node of the head of the subformula licenced by "a woman"). The verb structure is linked to the restriction of its right most argument (the tree for "loves" is linked to the scoping node of "a woman"). Here we make the simplifying assumption that scope is unambiguously determined by the linear order of words in the sentence. Scope ambiguity could be accounted for either by having a rule strategy that supports alternative rules and rule application order thereby inducing several possible solutions or by mapping the dependency graphs to underspecified semantic representations such as MRSs. In this case, the scoping links would need to underspecify, rather than specify scope and all argument structures should be linked directly to the verb structure.

Fourth (Figure 2e), predications are handled and existing substructures are expanded with the appropriate literals. For quantifiers, the restriction node is labelled with a literal whose predicate is the lemma of the nominal heading the quantifier and whose variable is the quantifier variable. Similarly the semantic structure licensed by verb and noun predicates are expanded as shown in Figure 2e so as to contain as predicate, the lemma of the licensing verb or deverbal noun and as variable, the variable bound by the existential quantifier licensed by this verb/deverbal. Additionally, literals are added for each of the verb arguments where each literal relates the verb event variable to the argument variable via the thematic role relation given in the SRL structure.

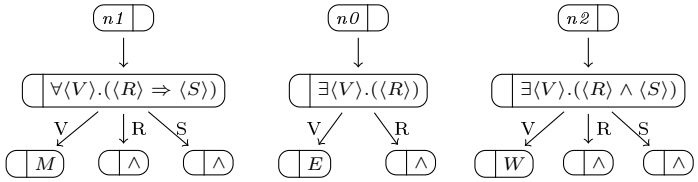
[7] For donkey sentences such as "If a farmer owns a donkey, he beats it" where the indefinite "a farmer" licenses a universal quantifier, the approach should be modified so as to build Discourse Representation Structures rather than FOL formulae.



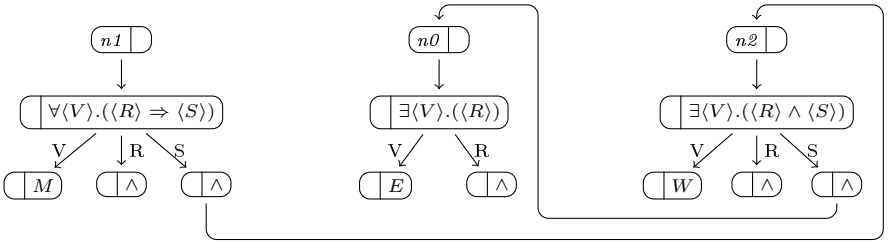
(a) Input dependency+SRL graph



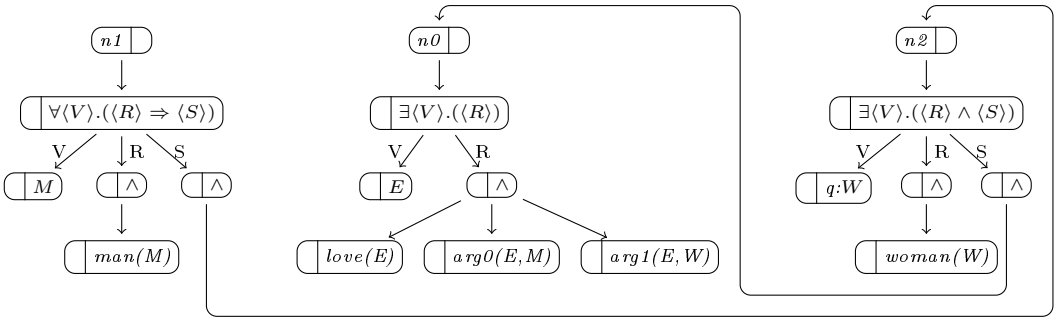
(b) Creation of seed nodes



(c) Specifying types



(d) Specifying scope



(e) Specifying predications

Fig. 2: Derivation example "Every man loves a woman". The notation $\forall\langle V\rangle.(\langle R\rangle \Rightarrow \langle S\rangle)$ is syntactic sugar indicating that the node is labelled with the attribute value pair QUANT:FORALL and that a FOL formula of that shape can be reconstructed from the subgraph rooted in that node. Indeed, from the final representation, the following FOL formula can be derived: $\forall M.(man(M) \wedge \exists W.(woman(W) \wedge \exists E.(love(E) \wedge arg0(E, M) \wedge arg1(E, W))))$

3.4 Rules, variable binding and semantic phenomena

We now show in more detail how variable binding occurs and sketch the treatment of relative clauses, raising, control and questions.

Variable binding (Quantifier restriction and verb semantics). Semantic construction must ensure that the variable bound by a quantifier correctly occurs in its restriction and in its scope. Here, this is ensured by equating the relevant variable in the restriction and in the scope with the quantifier variable. Figure 3 illustrates this graphically. The top rule shows how the quantifier restriction is labelled with a literal *lemma(V)* where *lemma* is the nominal head of the quantifier and V, the variable bound by the quantifier. Similarly (Figure 3b), each **argN** edge in the input graph licenses the introduction in the verb semantics of a literal *ArgN(E,A)* where E is the event variable licensed by the verb and A the variable licensed by the argN argument. Note that the binding of argument variables is mediated not by syntactic functions but by thematic roles thereby simplifying the syntax/semantic interface (because distinct syntactic realisations are abstracted over).

Relative clauses. Relativised arguments are processed in the same way as arguments of a main clause verb because thematic roles relate the verb of a relative clause, not to the relative pronoun, but to its antecedent (cf. Figure 4) and, as just mentioned, the binding of predicate argument variables is mediated by thematic roles. The scoping rules additionnally ensure that the semantics associated with a relative clause is included in the restriction of the relative antecedent.

Control As for relativised arguments, control verbs do not necessitate any additional rules because the semantic role labeller already provides the information required for appropriately binding the subject (or the object) of the control verb to the subject of its infinitival complement. Thus, in "John promised Mary to shave", "John" is labelled as ARG0 of both "promised" and "shave", thereby supporting the appropriate variable bindings.

Adjectival and Adverbial Modifiers. Adjectives and adverbs licence the introduction of a predication over an individual and an event variable respectively. This variable is equated with the variable predicated of by the denotation of the modifiee (i.e., the noun or the verb) using a rule which can be summarised as follows: if the dependency node A is in a modification relation to the dependency node W and W is related to a semantic structure with bound variable X, then the literal $A(X)$ should be included in the restriction of this semantic structure.

Raising. Raising verbs such as "seems" in "John seems to love Mary" and "It seems that John loves Mary" are handled as modifiers in that they modify the event variable introduced by the sentential or infinitival object. Semantic role labelling ensures that "John" is the ARG0 of "loves" in both cases (cf. Figure 5) and therefore that the appropriate semantics is constructed.

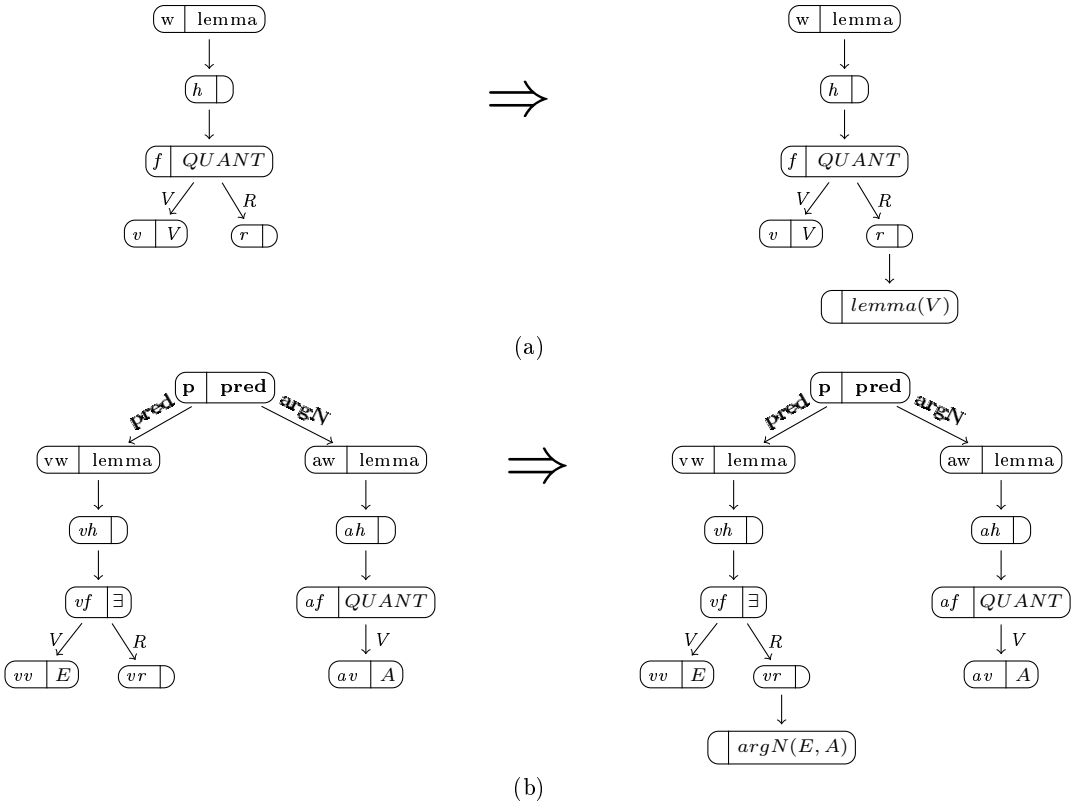


Fig. 3: Rules adding the literal licensed by the nominal head of the quantifier (*lemma(V)*) to its restriction (top) and the argument literals (*argN(E,A)*) to its scope (bottom). *QUANT* and ∃ abreviate the attribute value pairs QUANT:EXISTS OR FORALL and QUANT:EXISTS respectively while V, A and E are variables.The scope branch of the quantifier is not represented in the rule as the rule filter needs only specify the minimal pattern that should be present in the host graph for the rule to be applicable. Additional material is copied over. The top rule states that given a graph containing a dependency node w, labelled with the word "lemma" and linked to the skeleton quantifier subgraph rooted in h, the literal *lemma(V)* should be added to the quantifier restriction. Similarly, the bottom rule rewrites subgraphs relating a verb semantic skeleton (rooted in wh) and any of its argument semantic skeleton (rooted in ah) by adding the literal *argN(E,A)* to the verb semantics.

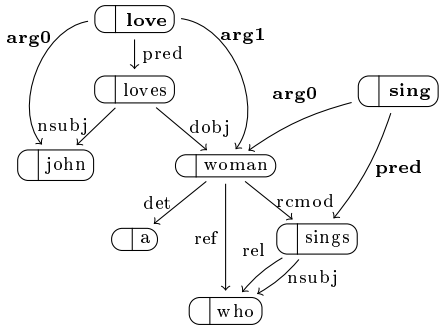


Fig. 4: "John loves a woman who sings". Semantic role labelling relate the predicate licensed by the verb of the relative clause not to the relative pronoun but to its antecedent thereby supporting a uniform semantic treatment of relativised and non relativised argumentsXS

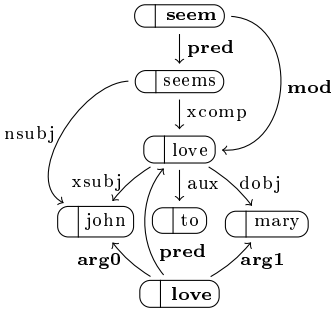


Fig. 5: "John seems to love Mary". The symbolic SRLer we use produces a modification relation between “seems” and its sentential argument thereby supporting a modifier treatment of “seem”.

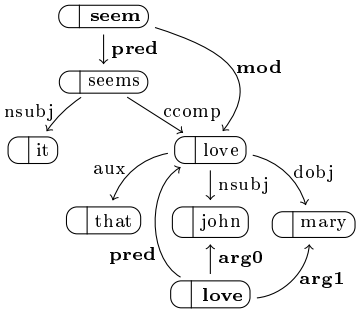


Fig. 6: "It seems that John loves Mary". The symbolic SRLer correctly labels “John” as the **arg0** of “loves”.

Questions As mentioned in Section 2, the dependency graph of questions such as "Which woman does Mary think John likes?" fails to support a strictly compositional semantics because local information is not sufficient to simultaneously determine that "Which woman" is the object of "likes" and takes scope over the whole sentence. In our approach, such sentences are unproblematic: "Which woman" licences the introduction of a quantifier which binds the object variable of "likes" (through the normal predicate/argument binding mechanism); further, its scope is determined to respect the linear order of the words in the input sentence.

Coverage and evaluation We tested (12) the coverage and the correction of our approach by applying it to a set of 1 000 sentence pairs annotated with an entailment value (true if the first sentence entails the other, false otherwise). For each sentence, the sentences were parsed using the Stanford parser and the semantic role labeller of (11), semantic construction was carried out and the resulting semantic representations translated to FOL. Automated reasoners were then used to check entailment. In all cases, a correct FOL formula was built. Moreover, entailment detection was correct in 71.3% of the cases. Since in many cases, parsing failed to produce a correct analysis, these first results are encouraging. They need to be further tested on real world data though as the testsuite used in this first experiment was artificially constructed and restricted to a limited set of linguistic variations (different verb subcategorisation type and control mainly).

4 Conclusion

By adopting a semantics rather than a syntax driven strategy, the semantic construction approach described in this paper permits bypassing the issues raised by the lack of syntactic information in dependency graphs. More generally, the approach can be seen as defining a set of very general principles governing the construction of semantic representations for predicate/argument structures, quantifiers and modifiers. Contrary to the lambda calculus approach, this allows for a very concise system where a small set of rewrite rules can be used to describe a large number of syntax-semantics interfaces. We are currently extending the approach to cover further semantic phenomena (e.g., comparatives and discourse connectives) and evaluate its coverage and correction using the entailment recognition test.

Bibliography

[1] Bos, J., Clark, S., Steedman, M., Curran, J.R., Hockenmaier, J.: Wide-coverage semantic representations from a ccg parser. In: Proceedings of the 20th International Conference on Computational Linguistics (COLING '04), Geneva, Switzerland (2004) 1240–1246

[2] Copestake, A., Flickinger, D., Sag, I., Pollard, C.: Minimal recursion semantics: An introduction. Journal of Research on Language and Computation **3** (2005) 281–332

[3] van Genabith, J., Frank, A., Crouch, D.: Glue, underspecification and translation (1999)

[4] Gardent, C., Kallmeyer, L.: Semantic construction in ftag. In: Proceedings of the 10th meeting of the European Chapter of the Association for Computational Linguistics, Budapest, Hungary (2003)

[5] Debusmann, R., Duchier, D., Koller, A., Kuhlmann, M., Smolka, G., Thater, S.: A relational syntax-semantics interface based on dependency grammar. In: COLING '04: Proceedings of the 20th international conference on Computational Linguistics, Morristown, NJ, USA, Association for Computational Linguistics (2004) 176

[6] Cimiano, P.: Flexible semantic composition with dude. In: Proceedings of the 8th International Conference on Computational Semantics (IWCS'09). (2009)

[7] Montague, R.: The proper treatment of quantification in ordinary English. In Thomason, R., ed.: Formal Philosophy. Selected Papers. Yale University Press, New Haven (1974)

[8] Ehrig, H., Heckel, R., Korff, M., M., L., Ribeiro, L., Wagner, A., Corradini, A.: Algebraic Approaches to Graph Transformation - Part II: Single Pushout A. and Comparison with Double Pushout A. In: Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1. World Scientific (1999) 247–312

[9] Kroll, M., Geiß, R.: Developing graph transformations with grgen.net. Technical report (2007) preliminary version, submitted to AGTIVE 2007.

[10] Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: ACL, Sapporo, Japan (2003) 423–430

[11] Bedaride, P., Gardent, C.: Noun/verb inference. In: 4th Language and Technology Conference, Poznan, Poland (2009)

[12] Bedaride, P., Gardent, C.: Benchmarking for syntax-based sentential inference. In: The 23rd International Conference on Computational Linguistics - COLING 2010, Beijing China (2010)