Semantic construction with graph grammars

Alexander Koller
University of Potsdam
koller@ling.uni-potsdam.de

Abstract

We introduce *s-graph grammars*, a new grammar formalism for computing graph-based semantic representations. Semantically annotated corpora which use graphs as semantic representations have recently become available, and there have been a number of data-driven systems for semantic parsing that can be trained on these corpora. However, it is hard to map the linguistic assumptions of these systems onto more classical insights on semantic construction. S-graph grammars use graphs as semantic representations, in a way that is consistent with more classical views on semantic construction. We illustrate this with a number of hand-written toy grammars, sketch the use of s-graph grammars for data-driven semantic parsing, and discuss formal aspects.

1 Introduction

Semantic construction is the problem of deriving a formal semantic representation from a natural-language expression. The classical approach, starting with Montague (1974), is to derive semantic representations from syntactic analyses using hand-written rules. More recently, semantic construction has enjoyed renewed attention in mainstream computational linguistics in the form of *semantic parsing* (see e.g. Wong and Mooney, 2007; Zettlemoyer and Collins, 2005; Chiang et al., 2013). The idea in semantic parsing is to make a system learn the mapping of the string into the semantic representation automatically, with or without the use of an explicit syntactic representation as an intermediate step. The focus is thus on automatically induced grammars, if the grammar is not left implicit altogether.

Training such data-driven models requires semantically annotated corpora. One recent development is the release of several corpora in which English sentences were annotated with *graphs* that represent the meanings of the sentences (Banarescu et al., 2013; Oepen et al., 2014). There has already been a body of research on semantic parsing for graphs based on these corpora, but so far, the ideas underlying these semantic parsers have not been connected to classical approaches to semantic construction. Flanigan et al. (2014) and Martins and Almeida (2014) show how to adapt data-driven dependency parsers from trees to graphs. These approaches do not use explicit grammars, in contrast to all work in the classical tradition. Chiang et al. (2013) do use explicit Hyperedge Replacement Grammars (HRGs, Drewes et al., 1997) for semantic parsing. However, HRGs capture semantic dependencies in a way that is not easily mapped to conventional intuitions about semantic construction, and indeed their use in linguistically motivated models of the syntax-semantics interface has not yet been demonstrated.

But just because we use graphs as semantic representations and learn grammars from graph-banks does not mean we need to give up linguistic insights from several decades of research in computational semantics. In this paper, we address this challenge by presenting *s-graph grammars*. S-graph grammars are a new synchronous grammar formalism that relates strings and graphs. At the center of a grammatical structure of an s-graph grammar sits a *derivation tree*, which is simultaneously interpreted into a string (in a way that is equivalent to context-free grammar) and into a graph (in a way that is equivalent to HRGs). Because of these equivalences, methods for statistical parsing and grammar induction transfer from these formalisms to s-graph grammars. At the same time, s-graph grammars make use of an explicit inventory of semantic argument positions. Thus they also lend themselves to writing linguistically interpretable grammars by hand.

Proceedings of the 11th International Conference on Computational Semantics, pages 228–238,
London, UK, April 15-17 2015.

The paper is structured as follows. In Section 2, we will briefly review the state of the art in semantic parsing, especially for graph-based semantic representations. We will also introduce Interpreted Regular Tree Grammars (IRTGs) (Koller and Kuhlmann, 2011), which will serve as the formal foundation of s-graph grammars. In Section 3, we will then define s-graph grammars as IRTGs with an interpretation into the *algebra of s-graphs* (Courcelle and Engelfriet, 2012). In Section 4, we will illustrate the linguistic use of s-graph grammars by giving toy grammars for a number of semantic phenomena. We conclude by discussing a number of formal aspects, such as parsing complexity, training, and the equivalence to HRG in Section 5.

2 Previous Work

We start by reviewing some related work.

2.1 Semantic construction and semantic parsing

In this paper, we take "semantic construction" to mean the process of deriving a semantic representation from a natural-language string. Semantic construction has a long tradition in computational semantics, starting with Montague (1974), who used higher-order logic as a semantic representation formalism.

Under the traditional view, the input representation of the semantic construction process is a syntactic parse tree; see e.g. Blackburn and Bos (2005) for a detailed presentation. One key challenge is to make sure that a semantic predicate combines with its semantic arguments correctly. In Montague Grammar, this is achieved by constructing through functional application of a suitably constructed lambda-term for the functor. Many unification-based approaches to semantic construction, e.g. in the context of HPSG (Copestake et al., 2001), TAG (Gardent and Kallmeyer, 2003), and CCG (Baldridge and Kruijff, 2002) use explicit argument slots that are identified either by a globally valid name (such as "subject") or by a syntactic argument position. Copestake et al. (2001), in particular, conclude from their experiences from designing large-scale HPSG grammars that explicitly named arguments simplify the specification of the syntax-semantics interface and allow rules that generalize better over different syntactic structures. Their *semantic algebra* assigns to each syntactic constituent a semantic representation consisting of an MRS, together with *hooks* that make specific variable names of the MRS available under globally available names such as "subject", "object", and so on. The algebra then provides operations for combining such representations; for instance, the "combine with subject" operation will unify the root variable of the subject MRS with the variable with the name "subject" of the verb MRS. Like this standard approach, and unlike the HRG approach described below, this paper identifies semantic argument positions by name as well, and extends this idea to graph-based semantic representations.

Recently, semantic construction has resurfaced in mainstream computational linguistics as "semantic parsing". Here the focus is on directly learning the mapping from strings to semantic representations from corpora, with or without the use of an explicit syntactic grammar. The mapping can be represented in terms of synchronous grammars (e.g. Wong and Mooney, 2007) or CCG (e.g. Zettlemoyer and Collins, 2005). Both of these lines of research have so far focused on deriving lambda terms, and are not obviously applicable to deriving graphs.

2.2 Graphs as semantic representations

The recent data-driven turn of semantic parsing motivates the development of a number of semantically annotated corpora ("sembanks"). Traditional semantic representations, such as higher-order logic, have shown to be challenging to annotate. Even the Groningen Meaning Bank (Bos et al., 2014), the best-known attempt at annotating a corpus with traditional semantic representations (DRT), uses a semi-automatic approach to semantic annotation, in which outputs of the Boxer system (Curran et al., 2007) are hand-corrected. A larger portion of recent sembanks use *graphs* as semantic representations. This strikes a middle ground, which is mostly restricted to predicate-argument structure, but can cover phenomena such as control, coordination, and coreference. In this paper, we will take the hand-annotated graphs of

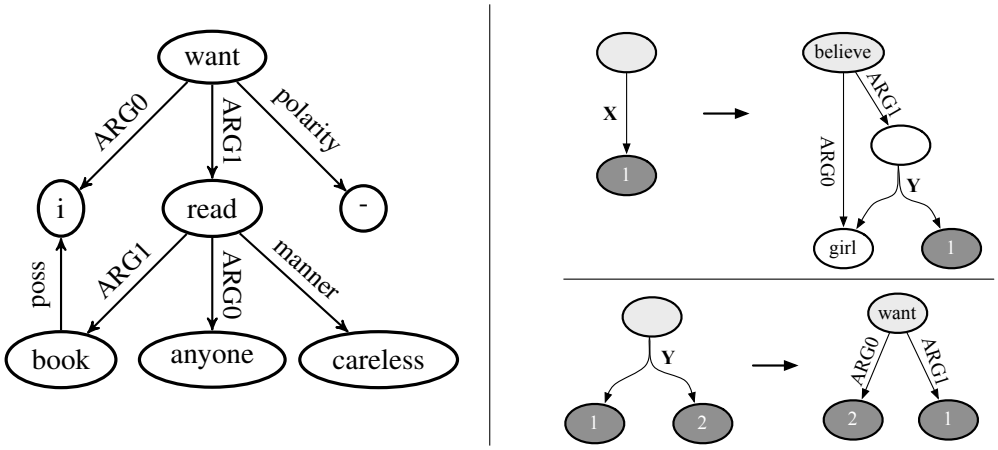

Figure 1: Left: an AMR for the sentence "I do not want anyone to read my book carelessly"; right: two example HRG rules.

the AMR-Bank (Banarescu et al., 2013) as our starting point. These *abstract meaning representations* (AMRs; see the example in Fig. 1) use edge labels to indicate semantic roles, including ones for semantic arguments (such as "ARG0", "ARG1", etc.) and ones for modification (such as "time", "manner", etc.). Graphs are also used as semantic representations in the *semantic dependency graphs* from the SemEval-2014 Shared Task (Oepen et al., 2014). These are either manually constructed or converted from deep semantic analyses using large-scale HPSG grammars.

The recent availability of graph-based sembanks has triggered some research on semantic parsing into graphs. Flanigan et al. (2014) and Martins and Almeida (2014) do this by adapting dependency parsers to compute dependency graphs rather than dependency trees. They predict graph edges from corpus statistics, and do not use an explicit grammar; they are thus very different from traditional approaches to semantic construction.

Chiang et al. (2013) present a statistical parser for synchronous string/graph grammars based on *hyperedge replacement grammars* (HRGs, Drewes et al., 1997). HRGs manipulate hypergraphs, which may contain hyperedges with an arbitrary number k of endpoints, labeled with nonterminal symbols. Each rule application replaces one such hyperedge with the graph on the right-hand side, identifying the endpoints of the nonterminal hyperedge with the "external nodes" of the graph. Jones et al. (2012) and Jones et al. (2013) describe a number of ways to infer HRGs from corpora. However, previous work has not demonstrated the suitability of HRG for linguistically motivated semantic construction. Typical published examples, such as the HRG rules from Chiang et al. (2013) shown in Fig. 1 on the right, are designed for succinctness of explanation, not for linguistic adequacy (in the figure, the external nodes are drawn shaded). Part of the problem is that HRG rules take a primarily top-down perspective on graph construction (in contrast to most work on compositional semantic construction) and combine arbitrarily complex substructures in single steps: the Y hyperedge in the first example rule is like a higher-order lambda variable that will be applied to three nodes introduced in that rule. The grammar formalism we introduce here builds graphs bottom-up, using a small inventory of simple graph-combining operations, and uses names for semantic argument positions that are much longer-lived than the "external nodes" of HRG.

2.3 Interpreted regular tree grammars

This paper introduces synchronous string/graph grammars based on *interpreted regular tree grammars* (IRTGs; Koller and Kuhlmann, 2011). We give an informal review of IRTGs here. For a more precise definition, see Koller and Kuhlmann (2011).

Informally speaking, an IRTG $\mathbb{G} = (\mathcal{G}, (h_1, \mathcal{A}_1), \ldots, (h_k, \mathcal{A}_k))$ derives a language k-tuples of objects, such as strings, trees, or graphs (see the example in Fig. 2). It does this in two conceptual steps. First, we build a *derivation tree* using a *regular tree grammar* $\mathcal{G}$. Regular tree grammars (RTGs; see

RTG rule	homomorphisms
$S \to r_1(NP, VP)$	1: $x_1 \bullet x_2$
	2: $x_1 \bullet x_2$
$VP \to r_2(V, NP)$	1: $x_1 \bullet x_2$
	2: $x_2 \bullet x_1$
$NP \to r_3$	1: John
	2: Hans
$NP \to r_4$	1: the box
	2: die Kiste
$NP \to r_5$	1: opens
	2: öffnet

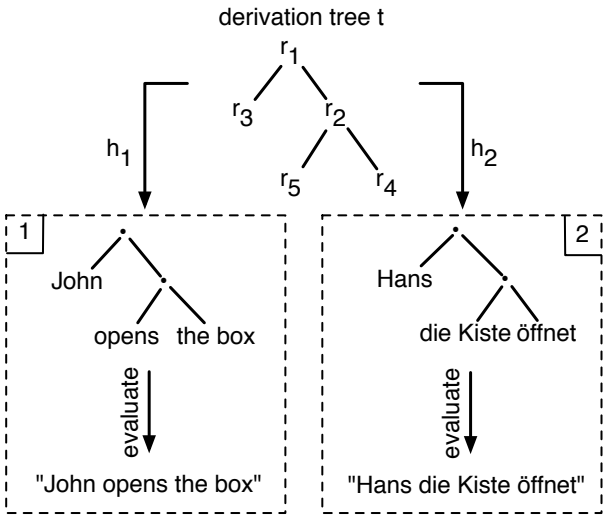

Figure 2: An IRTG (left) with an example derivation (right).

e.g. Comon et al., 2008, for an introduction) are devices for generating trees by successively replacing nonterminals using production rules. For instance, the RTG in the left column of Fig. 2 can derive the derivation tree t shown at the top right from the start symbol S.

In a second step, we can then *interpret* t into a tuple $(a_1, \ldots, a_k) \in \mathcal{A}_1 \times \ldots \times \mathcal{A}_k$ of elements from the *algebras* $\mathcal{A}_1, \ldots, \mathcal{A}_k$. An algebra $\mathcal{A}_i$ is defined over some set A_i, and can evaluate *terms* built from some signature Δ_i into elements of A_i. For instance, the *string algebra* T^* contains all strings over some finite alphabet T. Its signature contains two types of operations. Each element a of T is a zero-place operation, which evaluates to itself, i.e. $[\![\text{John}]\!] = \text{John}$. Furthermore, there is one binary operation $\bullet$, which evaluates to the binary concatenation function $[\![\bullet(t_1, t_2)]\!] = w_1 w_2$ where $w_i = [\![t_i]\!]$. Each term built from these operation symbols evaluates to a string in T^*; for instance, at the lower right of Fig. 2, the term $(\text{John} \bullet (\text{opens} \bullet \text{the box}))$ evaluates to the string $[\![\text{John} \bullet (\text{opens} \bullet \text{the box})]\!] =$ "John opens the box".

An IRTG derives elements of the algebras by mapping, for each interpretation $1 \leq i \leq k$, the derivation tree t to a term $h_i(t)$ over the algebra using a *tree homomorphism* h_i. The tree homomorphisms are defined in the right-hand column of the table in Fig. 2; for instance, we have $h_1(r_2) = x_1 \bullet x_2$ (which concatenates the V-string with the NP-string, in this order) and $h_2(r_2) = x_2 \bullet x_1$ (which puts the NP-string before the V-string). It then evaluates $h_i(t)$ over the algebra $\mathcal{A}_i$, obtaining an element of $\mathcal{A}_i$. Altogether, the language of an IRTG is defined as

$$L(\mathbb{G}) = \{\langle [\![h_1(t)]\!]_{\mathcal{A}_1}, \ldots, [\![h_k(t)]\!]_{\mathcal{A}_k} \rangle \mid t \in L(\mathcal{G})\}.$$

In the example, the pair $\langle \text{John opens the box}, \text{Hans öffnet die Kiste} \rangle$ is one element of $L(\mathbb{G})$. Generally, IRTGs with two string-algebra interpretations are strongly equivalent to synchronous context-free grammars, just as IRTGs with a single string-algebra interpretation represent context-free grammars. By choosing different algebras, IRTGs can also represent (synchronous) tree-adjoining grammars (Koller and Kuhlmann, 2012), tree-to-string transducers (Büchse et al., 2013), etc.

3 An algebra of s-graphs

We can use IRTGs to describe mappings between strings and graph-based semantic representations. Let an *s-graph grammar* be an IRTG with two interpretations: one interpretation (h_s, T^*) into a string algebra T^* as described above, and one interpretation $(h_g, \mathcal{A}_g)$ into an algebra $\mathcal{A}_g$ of graphs. In this section, we define a graph algebra for this purpose.

The graph algebra we use here is the *HR algebra* of Courcelle (1993), see also Courcelle and Engelfriet (2012). The values of the HR algebra are *s-graphs*. An s-graph G is a directed graph with node

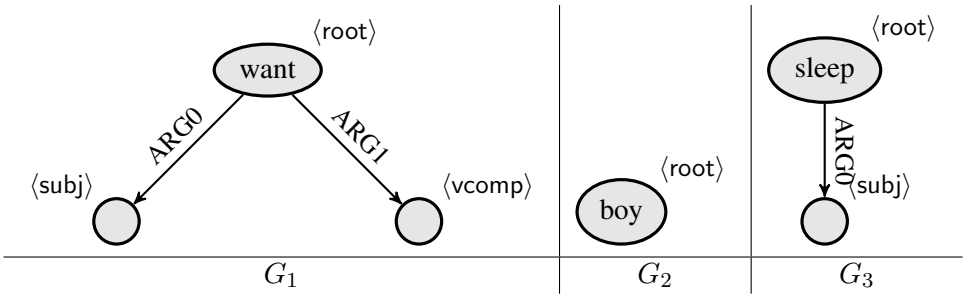

Figure 3: Example s-graphs.

labels and edge labels in which each node may be marked with a set of *source names* $s \in S$ from some fixed finite set S.[1] At most one node in an s-graph may be labeled with each source name s; we call it the *s-source of G*. We call those nodes of G that carry at least one source name the *sources* of G.

When we use s-graphs as semantic representations, the source names represent the different possible semantic argument positions of our grammar. Consider, for example, the s-graphs shown in Fig. 3. G_3 comes from the lexicon entry of the verb "sleep": It has a node with label "sleep", with an ARG0-edge into an unlabeled node. The "sleep"-node carries the source name root, indicating that this node is the "root", or starting point, of this semantic representation. (We draw source nodes shaded and annotate them with their source names in angle brackets.) We will ensure that every s-graph we use has a root-source; note that this node may still have incoming edges. The other node is the subj-source of the s-graph; this is where we will later insert the root-source of the grammatical subject. Similarly, G_1 has three sources: next to the labeled root, it has has two further unlabeled sources for the subj and vcomp argument, respectively.

The HR algebra defines a number of operations for combining s-graphs which are useful in semantic construction.

- The *rename* operation, $G[\mathsf{a}_1 \to \mathsf{b}_1, \ldots, \mathsf{a}_n \to \mathsf{b}_n]$, evaluates to a graph G' that is like G – except that for all i, the a_i-source of G is now no longer an a_i-source, but a b_i-source. All the n renames are performed in parallel. The operation is only defined if for all i, G has a a_i-source, and either there is no b_i-source or b_i is renamed away (i.e., $\mathsf{b}_i = \mathsf{a}_k$ for some k). Because we use the source name root so frequently, we write $G[\mathsf{b}]$ as shorthand for $G[\mathsf{root} \to \mathsf{b}]$.

- The *forget* operation, $\mathsf{f}_{\mathsf{a}_1,\ldots,\mathsf{a}_n}(G)$, evaluates to a graph G' that is like G, except that for all i, the a_i-source of G is no longer an a_i-source in G'. (If it was a b-source for some other source name b, it still remains a b-source.) The operation is only defined if G has a a_i-source for each i. [2]

- The *merge* operation, $G_1 \mid\mid G_2$, evaluates to a graph G' that consists of all the nodes and edges in G_1 and G_2. As a special case, if G_1 has an a-source u and G_2 has an a-source v, then u and v are mapped to the same node in G. This node then has all the adjacent edges of u and v in the original graphs.

By way of example, Fig. 4 shows the results of applying some of these operations to the s-graphs from Fig. 3. The first s-graph in the figure is $G_3' = G_3[\mathsf{vcomp}]$, which is shorthand for $G_3[\mathsf{root} \to \mathsf{vcomp}]$. Observe that this s-graph is exactly like G_3, except that the "sleep" node is now a vcomp-source and not a root-source. We then merge G_1 with G_3', obtaining the s-graph $G_1 \mid\mid G_3'$. In this s-graph, the vcomp-sources of G_1 and G_3' have been fused with each other; therefore, the ARG1-edge out of the "want" node (from G_1) points to a node with label "sleep" (which comes from G_3'). At the same time, the subj-sources of the two graphs were also fused. As a consequence, the ARG0 edge of the "want" node (from G_1) and the ARG0 edge of the "sleep" node (from G_3') point to the same node.

[1] Courcelle and Engelfriet's definition is agnostic as to whether the graph is directed and labeled.

[2] Note that Courcelle sees *rename* and *forget* as special cases of the same operation. We distinguish them for clarity.

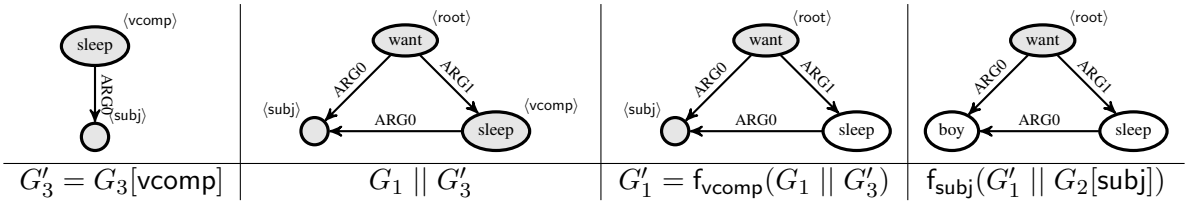

Figure 4: Combining the s-graphs of Fig. 3.

RTG rule	homomorphisms
S $\rightarrow \mathsf{comb\_subj}(\text{NP}, \text{VP})$	s: $x_1 \bullet x_2$ g: $\mathsf{f}_{\mathsf{subj}}(x_2 \mid\mid x_1[\mathsf{subj}])$
VP $\rightarrow$ sleep	s: sleep g: G_3
NP $\rightarrow$ boy	s: the boy g: G_2
VP $\rightarrow \mathsf{want}_1(\text{VP})$	s: wants to $\bullet\, x_1$ g: $\mathsf{f}_{\mathsf{vcomp}}(G_1 \mid\mid x_1[\mathsf{vcomp}])$
VP $\rightarrow \mathsf{want}_2(\text{NP}, \text{VP})$	s: wants $\bullet\, x_1 \bullet$ to $\bullet\, x_2$ g: $\mathsf{f}_{\mathsf{vcomp}}(G_1 \mid\mid \mathsf{f}_{\mathsf{obj}}(x_1[\mathsf{obj}] \mid\mid x_2[\mathsf{subj} \rightarrow \mathsf{obj}, \mathsf{root} \rightarrow \mathsf{vcomp}]))$

Figure 5: An s-graph grammar that illustrates complements.

Next, we decide that we do not want to add further edges to the "sleep" node. We can therefore forget that it is a source. The result is the s-graph $G'_1 = \mathsf{f}_{\mathsf{vcomp}}(G_1 \mid\mid G'_3)$. Finally, we can merge G'_1 with $G_2[\mathsf{subj}]$, and forget the subj-source, to obtain the final graph in Fig. 4. This s-graph could serve, for instance, as the semantic representation of the sentence "the boy wants to sleep".

4 Semantic construction with s-graph grammars

We will now demonstrate the use of s-graph grammars as a grammar formalism for semantic construction. We will first present a grammar that illustrates how functors, including control and raising verbs, combine with their complements. We will then discuss a second grammar which illustrates modification and coordination.

4.1 Complements

Consider first the grammar in Fig. 5. This is an s-graph grammar that consists of an RTG with start symbol S whose rules are shown in the left column, along with a string and a graph interpretation. The right column indicates the values of the homomorphisms h_s and h_g on the terminal symbols of the RTG rules; for instance, $h_g(\mathsf{comb\_subj}) = \mathsf{f}_{\mathsf{subj}}(x_2 \mid\mid x_1[\mathsf{subj}])$.

As a first example, let us use this grammar to derive a semantic representation for "the boy sleeps", as shown in Fig. 6. We first use the RTG to generate a derivation tree t, shown at the center of the figure. Using the homomorphism h_s, this derivation tree projects to the term $h_s(t)$ (shown on the left), which evaluates to the string "the boy sleeps" in the string algebra. At the same time, the homomorphism h_g projects the derivation tree to the term $h_g(t)$, which evaluates to the s-graph shown on the far right. Observe that the "boy" node becomes the ARG0-child of the "sleep" node by renaming the root-source of G_2 to subj; when the two graphs are merged, this node is then identified with the subj-source of G_3. These operations are packaged together in the homomorphism term $h_g(\mathsf{comb\_subj})$.

The grammar in Fig. 5 can also analyze sentences involving control verbs. Fig. 7 shows a derivation of the sentence "the boy wants to sleep". We can use the same rule for "sleep" as before, but now we use it as the VP argument of want_1. The grammar takes care to ensure that all s-graphs that can be

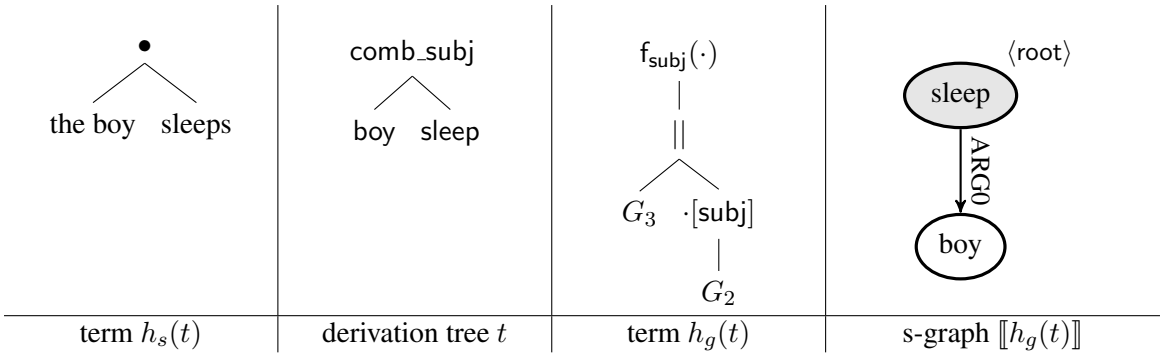

Figure 6: A derivation for "the boy sleeps", using the grammar in Fig. 5.

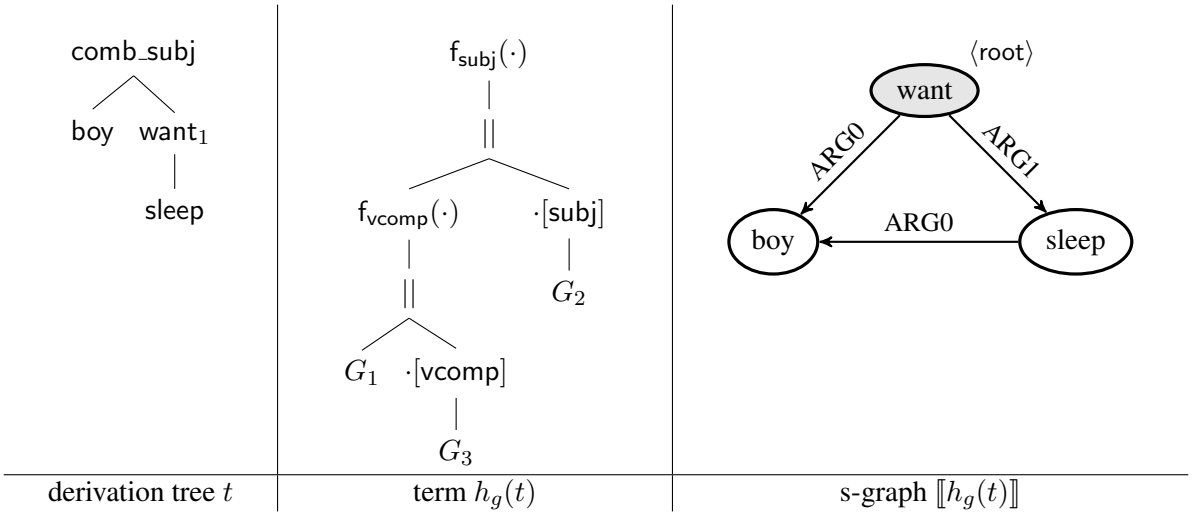

Figure 7: A derivation of "the boy wants to sleep" using the grammar in Fig. 5.

derived from VP have a subj-source along with the root-source that all derivable s-graphs have. Before we merge G_1 and G_3, we rename the root-source to vcomp to fill that argument position. By leaving the subj-source of G_3 as is, the merge operation fuses the subj-arguments of "sleep" and "wants", yielding the s-graph G_1' from Fig. 4. Notice that we did not explicitly specify the control behavior in the grammar rule; we just "let it happen" through judicious management of argument names.

The grammar also has a second rule for "wants", representing its use as a raising verb. The rule passes its grammatical object to its verb complement, where it is made to fill the subject role by renaming subj to obj. We omit the detailed derivation for lack of space, but only note that this rule allows us to derive AMRs like the one on the left of Fig. 1.

4.2 Modifiers

We now turn to an analysis of modification. The AMR-bank models modification with modification edges pointing from the modifier to the modifiee. This can be easily represented in an s-graph grammar by merging multiple root-sources without renaming them first.

We illustrate this using the grammar in Fig. 8. In the grammar we use shorthand notation to specify elementary s-graphs. For instance, the rule for coord merges its (renamed) arguments with an s-graph with three nodes, one of which is a root-source with label and and the other two are unlabeled sources for the source names 1 and 2, respectively; and with edges labeled op1 and op2 connecting these sources. In the examples below, we abbreviate this graph as G_{coord}; similarly, G_{snore} and $G_{\mathsf{sometimes}}$ are the elementary s-graphs in the "snores" and "sometimes" rules.

Consider the derivation in Fig. 9, which is for the sentence "the boy who sleeps snores" using the s-graph grammar from Fig. 8. The subtree of the derivation starting at "rc" represents the meaning of the

RTG rule	homomorphisms
NP → mod_rc(NP, RC)	s: $x_1 \bullet x_2$ g: $x_1 \mid\mid x_2$
RC → rc(RP, VP)	s: $x_1 \bullet x_2$ g: $(\mathsf{f}_{\mathsf{root}}(x_2 \mid\mid x_1[\mathsf{subj}]))[\mathsf{subj} \to \mathsf{root}]$
RP → who	s: who g: $\langle \mathsf{root} \rangle$
VP → coord(VP, VP)	s: $x_1 \bullet \mathsf{and} \bullet x_2$ g: $\mathsf{f}_{1,2}((\langle 1 \rangle \xleftarrow{\mathrm{op1}} \mathsf{and} \langle \mathsf{root} \rangle \xrightarrow{\mathrm{op2}} \langle 2 \rangle) \mid\mid x_1[1] \mid\mid x_2[2])$
VP → sometimes(VP)	s: sometimes $\bullet\, x_1$ g: $(\langle \mathsf{root} \rangle \xrightarrow{\mathrm{time}} \mathsf{sometimes}) \mid\mid x_1$
VP → snore	s: snores g: $\mathsf{snore} \langle \mathsf{root} \rangle \xrightarrow{\mathrm{ARG0}} \langle \mathsf{subj} \rangle$

Figure 8: An s-graph grammar featuring modification.

relative clause. It combines the s-graph for "sleep" from the grammar in Fig. 5 with the s-graph for the relative pronoun, which is just a single unlabeled node, using the ordinary renaming of the root-node of the relative pronoun to subj. However, the rule for the relative clause then differs from the ordinary rule for combining subjects with VPs by forgetting the root-source of the verb and renaming subj to root. This yields the s-graph "sleep $\xrightarrow{\mathrm{ARG0}} \langle \mathsf{root} \rangle$". The "mod_rc" rule combines this with the s-graph for "the boy" by simply merging them together, yielding "sleep $\xrightarrow{\mathrm{ARG0}}$ boy$\langle \mathsf{root} \rangle$". Finally this s-graph is combined as a subject with "snores" using the ordinary subject-VP rule from earlier.

This derivation illustrates the crucial difference between complements and adjuncts in the AMR-Bank. To combine a head with its *complements*, an s-graph grammar will rename the roots of the complements to their respective argument positions, and then forget these argument names because the complements have been filled. By contrast, the grammar assumes that each s-graph for an *adjunct* has a root-source that represents the place where the adjunct expects the modifiee to be inserted. Adjuncts can then be combined with the heads they modify by a simple merge, without any renaming or forgetting.

One challenge in modeling modification is in the way it interacts with coordination. In a sentence like "the boy sleeps and sometimes snores", one wants to derive a semantic representation in which the boy is the agent of both "sleeps" and "snores", but "sometimes" only modifies "snores" and not "sleeps". Fig. 10 shows how to do this with the grammar in Fig. 8. The subtree below "sometimes" (which projects to the string "sometimes snores" on the string interpretation) is interpreted as the s-graph "snore$\langle \mathsf{root} \rangle \xrightarrow{\mathrm{time}}$ sometimes". This is because the grammar rule for "sometimes" simply adds an outgoing "time" edge to the root-source of its argument, and leaves all sources in place. The coordination rule then combines this s-graph with G_3 from Fig. 3. This rule renames the root-sources of these two s-graphs to 1 and 2 respectively, but does not change their subj-sources. Thus these are merged together, so an ordinary application of the subject rule yields the s-graph shown at the right of Fig. 10.

5 Formal aspects

We conclude the paper by discussing a number of formal aspects of s-graph grammars.

First of all, this paper has focused on illustrating s-graph grammars using hand-written toy grammars for a number of semantic phenomena. We believe that s-graph grammars are indeed a linguistically natural grammar formalism for doing this, but ultimately one obviously wants to exploit the recent availability of meaning-banks to automatically induce and train statistical grammars. It is straightforward to make IRTGs a statistical grammar formalism, and to adapt algorithms for maximum likelihood and EM estimation, Viterbi parsing, etc. to statistical IRTGs (see Koller and Kuhlmann (2011) for a sketch of a

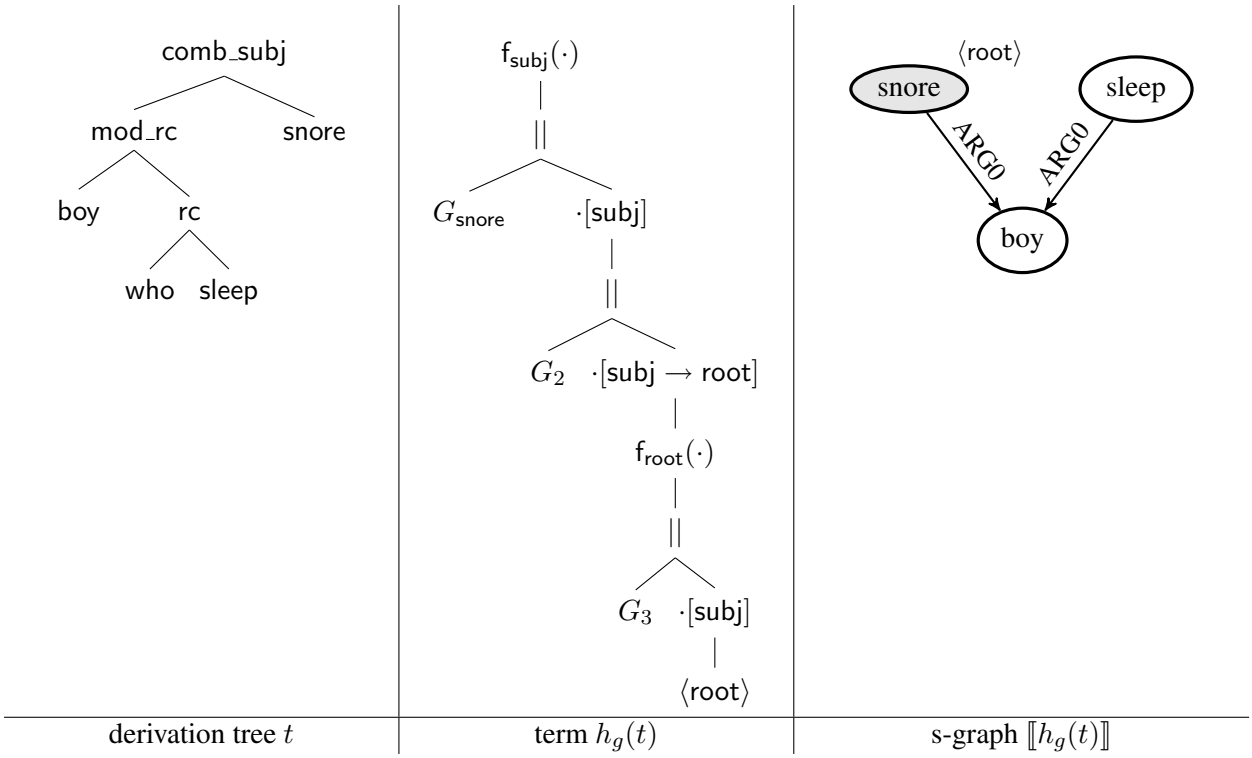

Figure 9: A derivation for "the boy who sleeps snores" using the grammar in Fig. 8.

PCFG-style probability model and Chiang (2003) for a discriminative probability model that transfers easily to IRTGs).

We have focused here on using s-graph grammars for semantic construction, i.e. as devices that map from strings to graphs. Algorithmically, this means we want to compute a parse chart from the string, extract the best derivation tree from it, and then compute its value in the graph interpretation. However, it is sometimes also useful to take a graph as the input, e.g. in an NLG or machine translation scenario, where one wants to compute a string from the graph.

We can adapt the IRTG parsing algorithm of Koller and Kuhlmann (2011) to graph parsing by showing how to compute *decomposition grammars* for the HR algebra. A decomposition grammar for an input object a is an RTG whose language is the set of all terms over the algebra that evaluate to a. We do not have the space to show this here, but the time complexity of computing decomposition grammars in the HR algebra is $O((n \cdot 3^d)^s \cdot ds)$, where n is the number of nodes in the graph, d is the maximum degree of the nodes in the graph, and s is the number of sources that are used in the grammar. This is also the overall parsing complexity of the graph parsing problem with s-graph grammars.

There is a close formal connection between s-graph grammars and Hyperedge Replacement Grammars (HRGs), which we reviewed in Section 2. Every HRG can be translated into an equivalent s-graph grammar; this follows from the known encoding of HRG languages as equational sets over the HR algebra, see e.g. Section 4.1 of Courcelle and Engelfriet (2012). More concretely, one can adapt the tree-decomposition construction of Chiang et al. (2013) to encode each HRG whose rules have maximum *treewidth* k into an equivalent s-graph grammar with $s = k + 1$ sources. Thus the parsing complexity of s-graph grammars coincides with Chiang et al.'s parsing complexity for HRGs of $O((n \cdot 3^d)^{k+1} \cdot d(k+1))$.

One key difference between HRGs and s-graph grammars is that the "sources" that arise in the translation of HRGs to IRTGs are meaningless, abstract names, which are forgotten after each rule application. By contrast, s-graph grammars can use linguistically meaningful source names which can persist if they are not explicitly forgotten in a rule. By managing the lifecycle of the subj-sources explicitly, we were able to write succinct rules for control verbs in Section 4.1. Similarly, the coordination rule in Section 4.2 fuses whatever sources the coordinated elements have, and thus it can be applied verbatim to other syntactic categories beyond VP. This reflects Copestake et al.'s (2001) observation that the use of

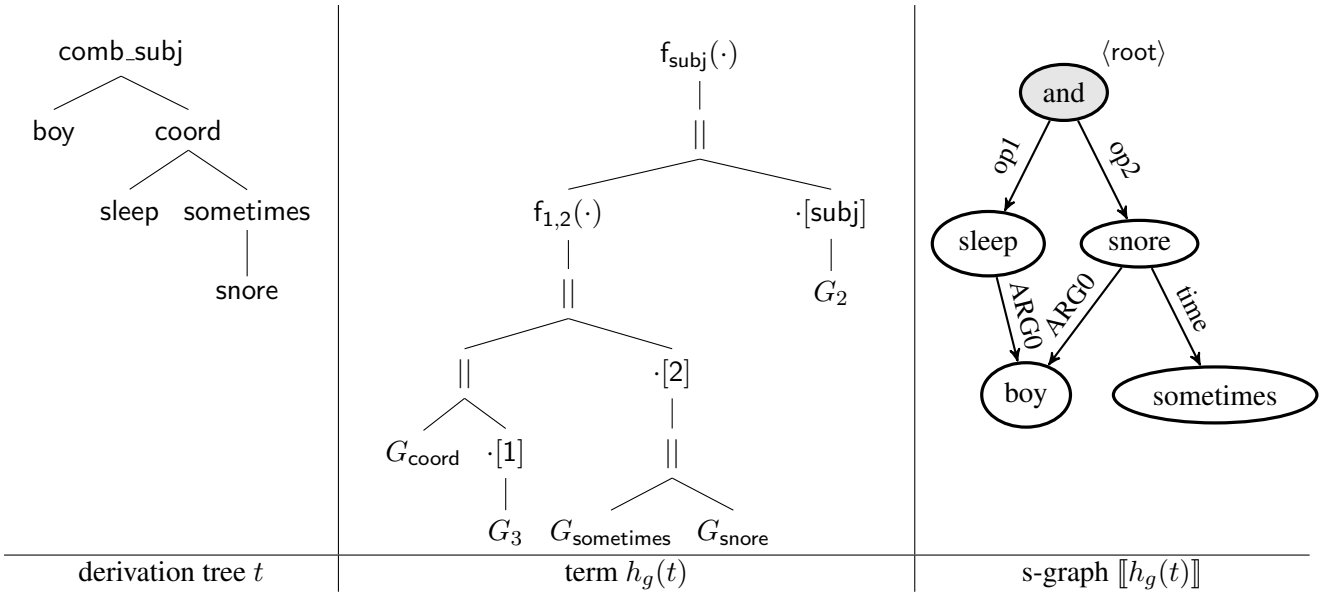

Figure 10: A derivation for "the boy sleeps and sometimes snores" using the grammar in Fig. 8.

explicit argument names can support more general semantic construction rules than are available when arguments are identified through their position in an argument list (as e.g. in HRG).

6 Conclusion

We have introduced s-graph grammars, a synchronous grammar formalism that describes relations between strings and graphs. We have also shown how this formalism can be used, in a linguistically principled way, for graph-based compositional semantic construction. In this way, we hope to help bridge the gap between linguistically motivated semantic construction and data-driven semantic parsing.

This paper has focused on developing grammars for semantic construction by hand. In future work, we will explore the use of s-graph grammars in statistical semantic parsing and grammar induction. A more detailed analysis of the consequences of the choice between argument names and argument positions in semantic construction is also an interesting question for future research.

Acknowledgments. The idea of using an IRTG over the HR algebra for semantic graphs was first developed in conversations with Marco Kuhlmann. The paper was greatly improved through the comments of a number of colleagues; most importantly, Owen Rambow, Christoph Teichmann, the participants of the 2014 Fred Jelinek Memorial JHU Workshop in Prague, and the three reviewers.

References

Baldridge, J. and G.-J. Kruijff (2002). Coupling CCG and hybrid logid dependency semantics. In *Proceedings of the 40th ACL*.

Banarescu, L., C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider (2013). Abstract meaning representation for sembanking. In *Proceedings of the Linguistic Annotation Workshop (LAW VII-ID)*.

Blackburn, P. and J. Bos (2005). *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI Publications.

Bos, J., V. Basile, K. Evang, N. Venhuizen, and J. Bjerva (2014). The Groningen Meaning Bank. In N. Ide and J. Pustejovsky (Eds.), *Handbook of Linguistic Annotation*. Berlin: Springer. Forthcoming.

Büchse, M., A. Koller, and H. Vogler (2013). General binarization for parsing and translation. In *Proceedings of the 51st ACL*.

Chiang, D. (2003). Mildly context-sensitive grammars for estimating maximum entropy parsing models. In *Proceedings of the 8th FG*.

Chiang, D., J. Andreas, D. Bauer, K. M. Hermann, B. Jones, and K. Knight (2013). Parsing graphs with hyperedge replacement grammars. In *Proceedings of the 51st ACL*.

Comon, H., M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi (2008). Tree automata techniques and applications. `http://tata.gforge.inria.fr/`.

Copestake, A., A. Lascarides, and D. Flickinger (2001). An algebra for semantic construction in constraint-based grammars. In *Proceedings of the 39th ACL*.

Courcelle, B. (1993). Graph grammars, monadic second-order logic and the theory of graph minors. In N. Robertson and P. Seymour (Eds.), *Graph Structure Theory*, pp. 565–590. AMS.

Courcelle, B. and J. Engelfriet (2012). *Graph Structure and Monadic Second-Order Logic*, Volume 138 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press.

Curran, J., S. Clark, and J. Bos (2007). Linguistically motivated large-scale NLP with C&C and Boxer. In *Proceedings of the 45th ACL: Demos and Posters*.

Drewes, F., H.-J. Kreowski, and A. Habel (1997). Hyperedge replacement graph grammars. In G. Rozenberg (Ed.), *Handbook of Graph Grammars and Computing by Graph Transformation*, pp. 95–162.

Flanigan, J., S. Thomson, J. Carbonell, C. Dyer, and N. A. Smith (2014). A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of ACL*, Baltimore, Maryland.

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

Jones, B. K., J. Andreas, D. Bauer, K. M. Hermann, and K. Knight (2012). Semantics-based machine translation with hyperedge replacement grammars. In *Proceedings of the 24th COLING*.

Jones, B. K., S. Goldwater, and M. Johnson (2013). Modeling graph languages with grammars extracted via tree decompositions. In *Proceedings of the 11th FSMNLP*.

Koller, A. and M. Kuhlmann (2011). A generalized view on parsing and translation. In *Proceedings of the 12th IWPT*.

Koller, A. and M. Kuhlmann (2012). Decomposing TAG algorithms using simple algebraizations. In *Proceedings of the 11th TAG+ Workshop*.

Martins, A. F. T. and M. S. C. Almeida (2014). Priberam: A turbo semantic parser with second order features. In *Proceedings of SemEval 2014*.

Montague, R. (1974). The proper treatment of quantification in ordinary english. In R. Thomason (Ed.), *Formal philosophy: Selected papers of Richard Montague*. New Haven: Yale University Press.

Oepen, S., M. Kuhlmann, Y. Miyao, D. Zeman, D. Flickinger, J. Hajic, A. Ivanova, and Y. Zhang (2014). SemEval 2014 Task 8: Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*.

Wong, Y. W. and R. J. Mooney (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th ACL*.

Zettlemoyer, L. S. and M. Collins (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st UAI*.