Statistical Language Generation from Semantic Structures

Bernd Bohnet[1], **Simon Mille**[2], **Leo Wanner**[2,3]
[1] Institut für maschinelle Sprachverarbeitung (IMS)
Universität Stuttgart, {first-name.last-name}@ims.uni-stuttgart.de
[2] Departament de Tecnologies de la Informació i les Comunicacions
Universitat Pompeu Fabra
[3]Institució Catalana de Recerca i Estudis Avançats (ICREA)
{first-name.last-name}@upf.edu

Abstract

Semantic stochastic sentence realization is still in its fledgling stage. Most of the available stochastic realizers start from syntactic structures or shallow semantic input structures which still contain numerous syntactic features. This is unsatisfactory since sentence generation traditionally starts from abstract semantic or conceptual structures. However, a change of this state of affairs requires first a change of the annotation of available corpora: even multilevel annotated corpora of the CoNLL competitions contain syntax-influenced semantic structures. We address both tasks—the amendment of an existing annotation with the purpose to make it more adequate for generation and the development of a semantic stochastic realizer. We work with the English CoNLL 2009 corpus, which we map onto an abstract semantic (predicate-argument) annotation and into which we introduce a novel "deep-syntactic" annotation, which serves as intermediate structure between semantics and (surface-)syntax. Our realizer consists of a chain of decoders for mappings between adjacent levels of annotation: semantic → deep-syntactic → syntactic → linearized → morphological.

1 Introduction

Deep, or semantic, stochastic sentence generation is still in its fledgling stage. Only a few stochastic generators start from real semantic input structures; see, for instance, (Wong and Mooney, 2007; Mairesse et al., 2010), who use higher order predicate logic structures as input. Most are either restricted to syntactic generation (Bangalore and Rambow, 2000; Langkilde-Geary, 2002; Filippova and Strube, 2008) or imply a symbolic submodule that operates on semantic structures to derive syntactic structures that are then used by the stochastic submodule (Knight and Hatzivassiloglou, 1995; Langkilde and Knight, 1998).

Walker et al. (2002) and Stent et al. (2004) start from *deep-syntactic structures* (DSyntSs) as introduced in the Meaning-Text Theory (MTT) (Mel'čuk, 1988), which they consider to be semantic. However, as argued by numerous authors, DSyntSs are, in fact, genuine syntactic structures, although they reflect the valency of the lexemes.

Bohnet et al. (2010) use CoNLL 2009 shared task corpora (Hajič, 2009) annotated in accordance with the PropBank/NomBank annotation guidelines (Palmer et al., 2005; Meyers et al., 2004), which they preprocess to adapt for dependency-based generation: non-connected adjectival modifiers are annotated as predicates with their syntactic heads as arguments, detached verbal arguments are connected with their head, etc. However, the result of this preprocessing stage is still not a genuine semantic structure: it contains all nodes of a (surface-) syntactic structure (auxiliaries, governed prepositions, determiners, etc.), including the nodes of functional words, and the part of speech tags of the individual nodes. Furthermore, it maintains the syntactic traces of the PropBank annotation such as the orientation of modifier relations and annotation of control and relative constructions.

All these types of information cannot be counted upon in most applications of natural language generation (NLG), which start from numeric time series or conceptual or semantic structures. In order to ensure a high quality linguistic generation, sentence realizers must be able to take as input abstract semantic structures derived from numeric time series or conceptual structures. In this paper, we present a deep sentence realizer that achieves this goal. Similar to (Bohnet et al., 2010), we start from

a CoNLL 2009 shared task corpus. However, unlike (Bohnet et al., 2010), we extend the CoNLL 2009 annotation in two respects: (i) we map the original CoNLL 2009 annotation onto a more abstract semantic annotation, and (ii) we introduce a deep-syntactic annotation in the sense of MTT (and as has already been used by Walker et al. (2002) and Stent et al. (2004)), which provides intermediate linguistic structures that do not contain any superficial functional nodes, but rather only the grammatical function structures. The introduction of the semantic annotation allows us to get close to the predicate-argument structures in general considered in generation as input structures of acceptable abstraction (Mellish et al., 2006); the introduction of the deep-syntactic annotation helps ensure high quality output in that it bridges the gap between the abstract semantic structures and concrete linguistic structures as the "surface-syntactic" structures are. So far, we carried out experiments only on the generation of English, but, in principle, our proposal is language-independent, as Bohnet et al. (2010)'s is.[1]

In the next section, we introduce the two new levels of annotation of the CoNLL 2009 corpus: the semantic and deep-syntactic annotations, and describe how we obtain them. In Section 3, we present the setup of the realizer. Section 4 outlines the individual stages of sentence realization: semantics → deep-syntax → (surface-)syntax → linearized structure → chain of inflected wordforms. Section 5 describes the setup of the experiments for the evaluation of the realizer and discusses the results of the evaluation. Section 7, finally, summarizes the most important features of the realizer and compares it to other recent approaches in the field.

2 Adjusting the CoNLL Annotation

As mentioned above, it is common in NLG to start from abstract input representations: conceptual or semantic structures derived from ontologies or even from numeric time series. Since it is not feasible to map such input structures to the linguistic surface in one shot without sacrifying the entire potential of linguistic variation, most generators draw on models that foresee a number of intermediate representations. Common are: 1) conceptual or semantic representation that is close to the abstraction of the knowledge in ontologies; 2) syntactic representation that captures the sentence structure; 3) a linearized morphological representation that spells out the inflection and orders the words in the sentence; see (Mellish et al., 2006) for an overview.

In order to get close to this ideal picture, we not only ensure, as Bohnet et al. (2010) do, that the starting semantic structure, i.e., the PropBank annotation, is a connected graph, but, furthermore, make it truly semantic. Furthermore, we introduce the MTT's DSyntS as an intermediate structure. DSyntS links to the semantic structure (SemS) in that it does not contain any function words, and, at the same time, to the CoNLL syntactic structure (SyntS) in that it contains the grammatical functions of the content words. DSyntS thus facilitates a two-step semantics-syntax projection, allowing for higher quality generation. For an evaluation of the quality of our annotations on a manually annotated gold standard, see (Wanner et al., submitted).

2.1 Deriving the Semantic Annotation

In order to turn a PropBank/NomBank-annotation, which, when visualized as a tree, looks as illustrated in Figure 1,[2] into a genuine semantic input annotation that can serve as departure for stochastic sentence generation, we 1) exclude the functional nodes from the annotation, 2) substitute syntactictically motivated arcs by semantic arcs, 3) introduce missing semantic nodes, minimal information structure, and 4) ensure connectivity of the semantic annotation.

1. Removal of functional nodes and syntactic edges: The following functional nodes and syntactic edges are removed from the PropBank annota-

[1] Obviously, the derivation of the semantic structure, which draws upon the available syntactic features remains language-specific.

[2] Ai (i = 1,2,3,...) denotes the i-th argument of a predicative word according to this word's frame (≈ valency) structure; A0 denotes "the external argument" of a predicative word; and AM-X denotes a modifier of type X (X = TMP (temporal), LOC(ation), DIR(ection), MNR (manner), etc.). In the course of this section, we also refer to R-Ai, C-Ai, NMOD, etc.: R-Ai (i = 1,2,3,...) stands for "i-th argument in a relative clause"; and C-Ai (i = 1,2,3,...) for "i-th argument in a control construction". For further details, see, e.g., (Palmer et al., 2005; Meyers et al., 2004) and references therein. NMOD, PMOD, VBD, etc. are Penn TreeBank tags.

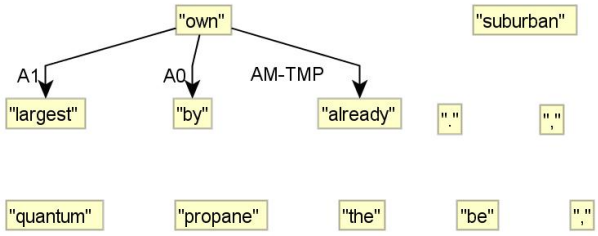

Figure 1: PropBank/NomBank annotation of the sentence *The largest, Suburban Propane, was already owned by Quantum.*

tion: (i) governed prepositions (i.e., prepositions annoted as predicate arguments A1, A2, ...); (ii) relative pronouns (i.e., nodes connected to the governing verb by an "R-Ax" edge); (iii) determiners and analytical auxiliaries (identified as such in the Penn TreeBank and PropBank annotations.);[3] (iv) control construction C-Ax edges since they stand for a syntactic dependency between a semantically controlled element and a verbal predicate.

2. Substitution of syntactically motivated edges: "Modifier" construction edges in PropBank AM-DIR, AM-LOC, AM-MNR, AM-TMP, AM-EXT, AM-PRD, AM-PNC, AM-CAU, AM-ADV, and AM-NEG are in their nature syntactic edges in that they go from the modified to the modifier. However, from the semantic view, the "modifiers" (or, better, "attributes" since we talk about semantic structure) are, in fact, predicative semantemes that take as argument the node that governs them in the syntactic structure. As a consequence, for these nodes we invert the original arc and label it with A1 in most cases. In the case of semantic prepositions and adverbs with two arguments, the second actant is linked to the preposition/adverb in question by an A2-edge.

3. Introduce missing semantic information: The PropBank annotation does not encode number and tense information, except for verbs with an analytical tense auxiliary. Since we remove auxiliaries, we add a tense feature to every predicate which has tense; similarly, we add a number feature to every noun:[4] TENSE: "past" for the PoS-tags VBD, VDD, VHD, VVD and "pres(ent)" for the PoS-tags VBP|VBZ, VDP|VDZ, VHP|VHZ, VVP|VVZ;[5] NUMBER: "singular" for the PoS-tags NN and NNP and "plural" for the PoS-tags NNS and NNPS.

4. Introduce minimal information structure:

In order to be able to map the semantic structure onto a syntactic tree, a minimal information (or *communicative* in terms of Mel'čuk (2001)) structure that captures theme/rheme and given/new is needed. We add the THEMATICITY and GIVENESS features: "THEMATICITY = theme" is assigned to the element which acts as subject in the syntactic structure and "THEMATICITY = rheme" to the main verb, the objects and close verb modifiers; "DEFINITENESS = 1" is assigned to elements with an indefinite determiner in the syntactic structure, and "DEFINITENESS = 2|3" to elements with a definite|demonstrative determiner.

5. Ensure connectivity of the semantic structure

As Bohnet et al. (2010), we ensure that the resulting semantic structure is a connected graph in that we traverse the syntactic dependency tree (i.e., the Penn Treebank annotation) d_{s_i} of each sentence x_i in the corpus breadth first and examine for each of d_{s_i}'s nodes n whether (i) it has a correspondence node n' in d_{s_i}'s semantic structure s_i obtained from the original shallow semantic graph in stages 1–4 sketched above, and (ii) n' is connected to the node that is n's semantic correspondence node. If not, we introduce a new arc between them. However, unlike Bohnet et al. (2010), who use a look-up table to read out the direction and labels of the introduced arcs, we implemented a rule-based procedure. This procedure makes use of PoS tags, syntactic arc labels, and the linearization information contained in the syntactic tree. Figure 2 shows a sample SemS as obtained applying Algorithm 2.[6]

[3] Interrogative pronouns are annotated the same way as relative pronouns in PB, but they are not removed since their removal would imply a loss of meaning; instead, we invert the R-Ax edge and relabel it with an arc "A1": an interrogative pronoun is also a semantic predicate having as argument what is being questioned.

[4] By doing so, we follow the newly announced Surface Generation Challenge `http://www.nltg.brighton.ac.uk/research/genchal11`.

[5] In case of analytical constructions (e.g., *has built*), the tense-feature is not directly on the verb, but derived from the syntactic construction.

[6] The passive of *own* is captured in the semantic annotation by the communicative feature "THEMATICITY = theme" as-

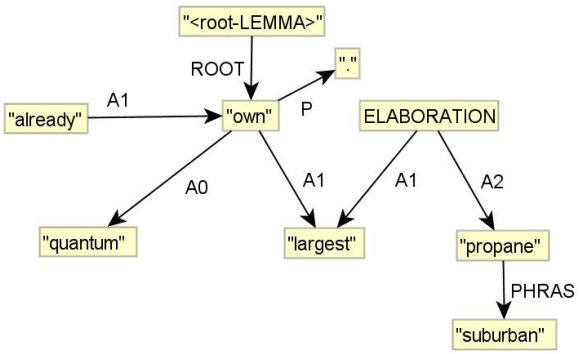

Figure 2: Semantic annotation of the sentence *The largest, Suburban Propane, was already owned by Quantum.* (the features assigned to each node are not shown)

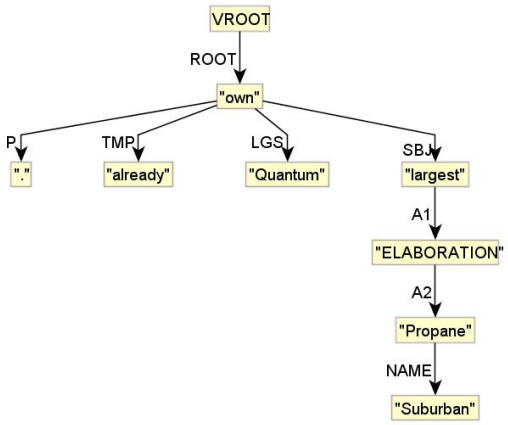

Figure 3: Deep-syntactic annotation of the sentence *The largest, Suburban Propane, was already owned by Quantum.* (the features asssigned to each node are not shown)

2.2 Deriving the Deep-Syntactic Annotation

As pointed out above, DsyntS is meant to facilitate the mapping between the abstract semantic structure obtained as described above and the CoNLL syntactic structure. It contains only content nodes, i.e., nodes of the semantic structure (function words are removed, and some nodes such as ”QUANTITY” or ”ELABORATION” are inserted into the semantic and deep-syntactic structures), and, at the same time, syntactic relations since the deep syntactic structure shows explicitely the structure of the sentence. That is, the governors and dependents are not organized based on predicate/argument relations, but rather on the notion of syntactic governor. The syntactic governor of a lexeme is the one that imposes syntactic constraints on its dependents: linearization and agreement constraints, case or governed preposition assignments, etc. Hence, like the syntactic structure, the deep-syntactic structure representation is a tree, not a graph. Every node at this level contains part-of-speech tags. Figure 3 shows a sample dsynts.

3 Setup of the Realizer

Our sentence realizer performs the following mappings to generate a sentence for a given semantic input graph:

1. *Semantic graph → Deep-syntactic tree*
2. *Deep-syntactic tree → Syntactic tree*
3. *Syntactic tree → Linearized structure*
4. *Linearized structure → Surface*

signed to *largest*.

Each of the steps is carried out by a decoder that uses a classifier to select the appropriate operations.

As already Bohnet et al. (2010), we use MIRA (Margin Infused Relaxed Algorithm) (Crammer et al., 2006) for the realization of the classifiers. MIRA has been successfully applied to structured learning tasks such as dependency parsing and semantic role labeling.[7]

We have to perform similar tasks for generation. The goal is to obtain a function that separates correct realizations (or items) by a decoder from the incorrect realizations. The items are characterised by features provided by feature extractors. The features are used to obtain a weight vector that separates the correct and incorrect items. The features are represented as a vector $\phi(x_i)$, which can be multiplied with the weight vector w in order to obtain a score.

The weight vector w can be obtained by an online learning algorithm. Online training considers a training example in each iteration of the training procedure. This has the advantage that we can process one example at a time, keeping only this example in the memory.

Algorithm 1 shows the outline of the training algorithm. The algorithm iterates I times over all training examples $\tau(x_i, y_i)_{i=1}^{n}$. A passive-

[7]The difference between MIRA and the perceptron algorithm is the use of a loss function by MIRA during the training procedure that measures the regret or cost for a wrong classification y' compared to the correct one y.

aggressive weight vector update strategy updates at the beginning of the training procedure the weights more aggressively. To what extent is determined by the factor β.

The weight vector v accumulates all weights, which are *averaged* at the end of the algorithm to avoid overfitting (Collins, 2002).

Algorithm 1: Online learning

Input: $\tau = \{(x_i, y_i)\}_{i=1}^{n}$
$w^{(0)} = 0; v = 0; i = 0;$
$\beta = I * N$
for n = 1 **to** I // Training iterations
  for n = 1 **to** N // Training instances
    $w^{(i+1)}$ = update $w^{(i)}$ according to (x_i, y_i)
    v = v + β w^{i+1}
    i = i + 1
    β = β - 1
$w = v/(I * N)$

4 Sentence Generation

Sentence generation consists in the application of the previously trained decoders in the sequence outlined in the previous section.

4.1 Semantic Generation

Our approach to semantic generation, which consists of the derivation of the deep-syntactic tree from an input semantic graph, is analogous to graph-based parsing (Eisner, 1996; McDonald and Pereira, 2006).

The derivation is defined as search for the highest scoring tree y from all possible trees given an input graph x:

$$F(x) = argmax\ Score(y), where\ y \in MAP(x)$$

(with $MAP(x)$ as the set of all trees spanning over the nodes of the semantic graph x).

As already proposed by Bohnet et al. (2010), the search is a beam search which creates a maximum spanning tree.[8] Unlike Bohnet et al. (2010), however, we use "early update" as introduced for parsing by Collins and Roark (2004): when the correct beam element drops out of the beam, we stop and update the model using the best partial solution. The idea behind this is that when all items in the current beam are incorrect, further processing is obsolete since the correct solution cannot be reached extending any elements of the beam. When we reach a final state, i.e. a tree spanning over all words and the correct solution is in the beam, but not ranked first, we perform an update as well since the correct element should have ranked first in the beam.

Algorithm 2 displays the algorithm for the generation of the deep-syntactic structure from the semantic structure. *extend-trees* is the central function of the algorithm. It expands a tree or a set of trees by one edge, selecting each time the highest scoring edge. Attachment point for an outgoing edge is any node; for an incoming edge only the top node of the built tree.

Algorithm 2: Semantic generation

//(x_i, y_i) semantic graph and the deep syntactic tree
//beam-size ← 80
// build an initial tree
for all $n_1 \in x_i$ **do**
  trees ← {} // empty list of partial trees
  for all $n_2 \in x_i$ **do**
    if $n_1 \neq n_2$ **then**
      for all $l \in$ edge-labels **do**
        trees = trees ∪ {(*synt*(n_1),*synt*(n_2),l)}
trees ← *sort-trees-descending-to-score*(trees)
trees ← *subset*(0,beam-size,trees)
// extend the initial trees consisting of one edge
while rest $\neq \emptyset$ **do**
  trees ← *extend-trees*(trees)
  trees ← *sort-trees-descending-to-score*(trees)
  trees ← *subset*(0,beam-size,trees)
  // training: **if** gold tree is not in the beam
  // **then** update weight vector and continue with next
return first element of trees

For score calculation, we use structured features composed of the following elements: (i) the lemmata, (ii) the **dist**ance between the starting node s and the target node t, (iii) the **dir**ection of the path (if the path has a direction), (iv) the sorted **bag** of ingoing edges labels without repetition, (v) the **path** of edge labels between source and target node. The templates of the composed structured features are listed in Table 1. We obtain about 2.6 Million features in total. The features have binary values, meaning that a structure has a specific feature or it does

[8] The maximum spanning tree algorithm can be applied here thanks to the introduction of the deep-syntactic structure.

not.

feature templates
label+dist(s, t)+dir
label+dist(s, t)+lemma$_s$+dir
label+dist(s, t)+lemma$_t$+dir
label+dist(s, t)+lemma$_s$+lemma$_t$+dir
label+dist(s, t)+bag$_s$+dir
label+dist(s, t)+bag$_t$+dir
label+path(s, t)+dir

Table 1: Feature templates for the semantic → deep-syntactic mapping ('s' means "source node" and 't' "target node" of an edge)

4.2 Deep-Syntactic Generation

Since the DSyntStr contains by definition only content words, function words such as governed prepositions, auxiliaries, and determiners must be introduced during the deep-syntactic–surface-syntactic generation passage in order to obtain a fully spelled out syntactic tree.

Tree transducers are best suited for this task because of their capability to rewrite trees. Top down tree transducers have been independently introduced by Rounds (1970) and Thatcher (1970) as extensions of finite state transducers. Tree Transducers have been already successfully applied in NLP—for instance, in machine translation (Knight and Graehl, 2005). Tree transducers traverse the input trees from the root to the leaves and rewrite the tree using rewriting rules.

For DSynt-generation, we use around 280 rules derived automatically by comparing a gold standard set of deep-syntactic structures and surface-syntactic depedency trees. The rules are of the following three types:

1. Rules introducing an edge and a node:
   X ⇒ X *label*$_s$ → Y ,
   Example: X ⇒ X *NMOD*→ *'the'*
2. Rules introducing a new node and edges between two nodes:
   X *label*$_d$→ Y ⇒ X *label*^1_s → N *label*^2_s → Y
   Example: X *OPRD*→ Y ⇒ X *OPRD*→ 'to' *IM*→ Y
3. Rules introducing a new node label:
   X ⇒ N
   Example: 'LOCATION' ⇒ 'on'

The restricted number of rules and rule types suggests the use of classifiers to select applicable rules in each stage of the DSynt-generation and thus consider more contextual information for the decision.

We train discriminative classifiers for each of three rule types that either selects a specific rule or NONE (i.e., that no rule is to be applied). Some parts do not need any changes. Therefore, on this parts there is no need to apply and the classifer has to select NONE. The Algorithm 3 displays the algorithm for the generation of the surface-syntactic structure from the deep-syntactic structure. The algorithm uses for score calculation features listed in Table 2.

Algorithm 3: Deep Syntactic Generation

```
//(x_i, y_i^g) the deep syntactic tree
// and gold surface syntactic tree for training case only
// R set of rules
// travers the tree top down depth first
y_i ←clone(x_i)
node-queue ← root(x_i)
while node-queue ≠ ∅ do
  //depth first traversal
  node ← remove-first-element(node-queue)
  node-queue ← children(node, x_i)∪ node-queue
  // select the rules, which insert a leaf node
  leaf-insert-rules ← select-leaf-rules(next-node,x_i,R)
  y_i ← apply (leaf-insert-rules,y_i)
  // in the training, we update here the weight vector
  // if the rules are not equal to the gold rules
  //
  // select the rules, which insert a node in the tree
  // or a new node label
  node-insert-rules ← select-node-rules(node,x_i,R)
  // in the training, we update here the weight vector
  y_i ← apply (edge-insert-rules,y_i)
```

Table 3 shows the confusion matrix of the DSynt→SSynt transducer rules. The first column contains the number of the gold rule that should have been applied; the second the gold rule itself and the third the actually applied rule. 'ie:' is the prefix of "insert-edge" rules, and 'in:' the prefix of "insert-node" rules.[9]

As we see, confusions occur, first of all, in the selection of the correct preposition in <nominal modifier>–<prepositional modifier> sequences in

[9]We hope that the Penn TreeBank tags 'NMOD', 'PMOD', 'DIR', 'OBJ', etc. are intuitive enough to allow for the understanding of the semantics of the rules.

feature template
pos(node)
pos(head(node))
pos(head(head(node)))
pos(node)+pos(head((node))
pos(node) + pos(head(node))+ edge-label(node)
feature-1(node)
feature-2(node)
feature-3(node)
feature-1(node)+feature-2(node)
lemma(node)
lemma(head(node))
lemma(node)+lemma(head(node))
bag-of-children-pos(node)
sorted-bag-of-children-pos(node)
sorted-bag-of-children-labels(node)

Table 2: *pos* are coarse-grained Part-of-Speech tags, *feature* are the features attached to the nodes, *lemma* are node labels, *edge-label* labels of edges; *feature-1* stands for "definite=yes", *feature-2* for "num=sg", and *feature-3* for "tense=past"

# rule	gold rule	wrongly applied rule
65	ie:NMOD:for:PMOD	ie:NMOD:of:PMOD
40	ie:LOC:in:PMOD	ie:NMOD:of:PMOD
34	ie:NMOD:to:PMOD	ie:NMOD:of:PMOD
23	ie:NMOD:on:PMOD	ie:NMOD:of:PMOD
26	ie:NMOD:with:PMOD	ie:NMOD:of:PMOD
18	ie:NMOD:from:PMOD	ie:NMOD:of:PMOD
16	ie:DIR:to:PMOD	ie:ADV:to:PMOD
12	ie:DIR:from:PMOD	ie:DIR:to:PMOD
11	in:NMOD:to	
11	ie:NMOD:of:PMOD	
10	ie:NMOD:of:PMOD	ie:LOC:in:PMOD
9	ie:ADV:at:PMOD	ie:ADV:for:PMOD
9	ie:DIR:from:PMOD	ie:ADV:from:PMOD
6	ie:PMOD:to:PMOD	
8	ie:OBJ:that:SUB	
8	ie:OPRD:to:IM	
8	ie:LOC:at:PMOD	ie:NMOD:with:PMOD

Table 3: Confusion matrix of the dsynt → synt rules

edge inserting rules. A possible solution to this problem that needs to be further explored is the inclusion of a larger context or/and consideration of semantic features.

4.3 Linearization and Morphologization

There is already a body of work available in statistical text generation on linearization and morphological realization. Therefore, these subtasks did not form the focus of our work. In the current version of the realizer, we use Bohnet et al. (2010)'s implementations. The linearization is a beam search for an optimal linearization according to a local and global score functions.

The morphological realization algorithm selects the edit script based on the minimal string edit distance (Levenshtein, 1966) in accordance with the highest score for each lemma of a sentence obtained during training and applies then the scripts to obtain the wordforms.

5 Experiments

To evaluate the proposed realizer, we carried out a number of experiments, whose setup and results are presented in what follows.

5.1 Setup of the Experiments

In our experiments, we use the PropBank/NomBank corpus of the CoNLL shared task 2009, which we preprocess as described in Section 2 to obtain the semantic structure from which we start. We follow the usual training, development and test data split (Langkilde-Geary, 2002; Ringger et al., 2004; Bohnet et al., 2010). Table 4 provides an overview of the used data.[10]

set	section	# sentences
training	2 - 21	39218
development	24	1334
test	23	2400

Table 4: Data split of the used data in the WSJ Corpus

In order to measure the accuracy of the isolated components and of the realizer as a whole and to be able to compare their performance with previous works, we use measures already used before, for instance, in (Ringger et al., 2004; Bohnet et al., 2010). Thus, for the semantics → deep-syntax mapping, we use the unlabeled and labeled attachment score, as it is also commonly used in dependency parsing. The unlabeled attachment score (ULA) is the percentage of correctly identified heads. The labeled attachment score (LAS) is the percentage of correctly identified heads that are in addition correctly labeled by syntactic functions. For the assessment of the deep-syntax → syntax mapping, we use the F-score of correctly/wrongly introduced nodes. For the evaluation of the sentence realizer as a whole, we use

[10]The raw PropBank/NomBank corpus of the CoNLL shared task 2009 is the WSJ corpus, such that the section numbers refer to sections in the WSJ corpus.

the BLEU metric on a gold standard compiled from our corpus.

Since we use Bohnet et al. (2010)'s implementations of the linearization and morphological realization, we use their metrics as well. To assess linearization, three metrics are used: (i) per-phrase/per-clause accuracy (*acc snt.*):

$$acc = \frac{correct\ constituents}{all\ constituents};$$

(ii) edit distance metrics:

$$di = 1 - \frac{m}{total\ number\ of\ words}$$

with m as the minimum number of deletions combined with insertions to obtain the correct order (Ringger et al., 2004); and (iii) the BLEU-score.

For the asessment of the morphological realization, the accuracy score (the ratio between correctly generated word forms and the entire set of generated word forms) is used.

5.2 Results of the Experiments

Table 5 displays the figures obtained for both the isolated stages of the semantic sentence realization and the generation as a whole—with reference to some of the recent works on statistical generation, and, in particular to (Bohnet et al., 2010), which is most similar to our proposal.[11] We include the performance of (Bohnet et al., 2010) in two stages that differ from our semantics→syntax, and syntax→topology (or linearized structure), and its overall performance. (Filippova and Strube, 2009) and (Ringger et al., 2004) are, in fact, not fully comparable with our proposal since the data are different. Furthermore, Filippova and Strube (2009) linearize only English sentences that do not contain phrases that exceed 20,000 linearization options—which means that they filter out about 1% of the phrases. We include them because these are reference works with which any new work on statistical generation has to compete.

5.3 Discussion

The overall performance of our semantic realizer is comparable (although somewhat lower) to the performance of (Bohnet et al., 2010). This is remarkable given that we start from a considerably more abstract semantic structure that does not contain any function words and that encodes some of the information (for instance, information structure features) in terms of node attributes instead of nodes/arcs. The performance of the semantics→deep-syntax projection is slightly lower than the semantics→syntax projection of (Bohnet et al., 2010). However, the quality of our deep-syntax→syntax projection is rather high—despite the fact that during this projection new nodes are introduced into the target structure (i.e., the projection is by far not isomorphic). A more detailed analysis of this projection shows that the precision of correctly introduced nodes is 0.79 and the recall is 0.74. As a result, we obtain an F-score of 0.765. However, the introduction of nodes affects only a relatively small part of the syntactic structure. Before we apply the rules, the (gold) deep-syntactic tree has about 92% correct nodes and correctly attached edges of the (surface) syntactic tree. After the rule application this value improves to about 97.6%. Our performance during the syntax→topology stage is slightly lower than in (Bohnet et al., 2010). This is the effect of the (imperfect) introduction of function words (such as determiners and prepositions) into the syntactic structure at the preceding stage. But it is still higher than the performance of the reference realizers such as (Ringger et al., 2004) and (Filippova and Strube, 2009) for this task.

Mapping	Value
Semantics→Deep-Syntax (ULA/LAS)	93.8/87.3
Deep-Syntax→Syntax (correct)	97.5
Syntax→Topology (BLEU)	0.89
All stages (BLEU)	0.64
All stages (BLEU) (Bohnet et al., 2010)	0.659
Semantics→Syntax (ULA/LAS) (Bohnet et al., 2010)	94.77/89.76
Syntax→Topology (di/acc) (Bohnet et al., 2010)	0.91/74.96
(Filippova and Strube, 2009)	0.88/67
(Ringger et al., 2004) (BLEU)	0.836

Table 5: Performance of the individual stages of semantic sentence realization and of the realization as a whole

[11] We do not compare here to (Wong and Mooney, 2007) and (Mairesse et al., 2010) because the tasks of both are rather different from ours: both explore phrase-based generation.

6 Related Work

Most of the widely cited works on statistical generation which use intermediate syntactic representations, as, for instance, Knight and Hatzivassiloglou (1995), Langkilde and Knight (1998) or Ringger et al. (2004), do not handle statistically the first stage of generation. Rather, they use rule-based components to build syntactic trees—even though some of them actually tackle the issue of statistical lexicalization, which we do not. Many recent works focalize on surface realization only, i.e., linearization and morphologization of syntactic representations; see, for instance, (Bangalore and Rambow, 2000; Filippova and Strube, 2008).

Mairesse et al. (2010) describe a statistical language generator, which uses dynamic Bayesian networks to assign a semantic part directly to a phrase. The representation is based on stacks which contain the semantic information for a sentence decomposed into phrases. The Bayesian networks are used to order the phrases and to align semantic parts with phrases. The model generalizes to some degree since it contains lexicalized backoff features that reduce the needed semantic coverage. For instance, the probability `P(r = centre of town l s=reject(area(centre)))` is backed off by `P(r = centre of town l h = centre)`.

Wong and Mooney (2007) present a generator based on an inverted semantic parser. The input is a partially ordered meaning representation. The process is similar to the one described in (Mairesse et al., 2010) in that they do not use any intermediate structure. Their statistical system, trained on very few sentences (880) produces concurrent output sentences. To choose the best candidate, they use n-gram models, as Knight and Hatzivassiloglou (1995), Bangalore and Rambow (2000) and Langkilde-Geary (2002). Walker et al. (2002) and Stent et al. (2004) describe a trainable sentence planner for dialog systems. The system uses MTT's DSyntSs as intermediate representations. In this respect, their approach is similar to ours. However, unlike us, they consider the DSyntSs predicate-argument structures, mapping fragments of text plans onto them by a set of operations in a bottom-up, left-to-right fashion. Starting from DSyntSs, they then use the rule-based RealPro generator to generate the sentences (Lavoie and Rambow, 1997).

7 Conclusions

We presented a decoder-based statistical semantic sentence realizer, which goes significantly beyond the works in this area, while showing a similar or, in some aspects, even better performance. The main difference of our proposal, to the statistical realizers of Ringger et al. (2004; He et al. (2009) is that we start with the generation from a truly semantic (predicate-argument) graph. An important extension compared to (Langkilde and Knight, 1998; Bohnet et al., 2010) is the mapping from the semantic graph to the DSyntS that forms an intermediate structure between the semantic structure and the (surface-) syntactic structure. In analogy to the semantic structure, the DSyntS contains no function words, and in analogy to the syntactic structure, it contains grammatical functions of the words that are present. This is motivated by the fact that we can easily build first a syntactic structure and then, in the next step, introduce function words based on the syntactic properties. We see this approach as the most promising direction for the derivation of a highly accurate syntactic tree and also in accordance with a holistic linguistic theory, namely MTT.

Unlike many of the previous works, we do not use at any stage components that are based on manually crafted rules. The abstract nature of the semantic structure and the availability of the DSyntS is an important add-on when compared to Bohnet et al. (2010)'s proposal, which starts from a semantic graph that already contains all words. The other works on statistical generation we know of that draw upon DSyntSs, namely (Walker et al., 2002; Stent et al., 2004), seem to overestimate the semantic nature of DSyntSs in that they consider them as (semantic) predicate-argument structures, which they are not: after all, DSyntSs are and remain syntactic structures, even if abstract ones.

Although we applied our approach so far only to English, the proposed realizer is language-independent—as the one proposed by Bohnet et al. (2010). In the months to come, we will apply it to other languages. This work will be accompanied by an effort to reach truly semantic corpus anno-

tations. The mapping of the PropBank/NomBank annotation to such an annotation demonstrated that CoNLL corpora are a good starting point for such an effort. As pointed out by one of the reviewers, LFG f-structures and MTT DSyntStrs also have a lot in common—which suggests experiments on deriving DSyntStr annotated corpora from LFG corpora.

Acknowledgements

The work described in this paper has been partially funded by the European Commission under the contract number FP7-ICT-248594. We would like to thank the four anonymous reviewers for their detailed and very useful comments.

References

S. Bangalore and O. Rambow. 2000. Exploiting a Probabilistic Hierarchical Model for Generation. In *Proceedings of COLING '00*, pages 42–48.

B. Bohnet, L. Wanner, S. Mille, and A. Burga. 2010. Broad coverage multilingual deep sentence generation with a stochastic multi-level realizer. In *Proceedings of COLING '10*, pages 98–106.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

M. Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the EMNLP Conference*.

K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. 2006. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, 7:551–585.

J. Eisner. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhaen.

K. Filippova and M. Strube. 2008. Sentence fusion via dependency graph compression. In *Proceedings of the EMNLP Conference*.

K. Filippova and M. Strube. 2009. Tree linearization in English: Improving language model based approaches. In *Proceedings of the NAACL '09 and HLT, Short Papers*, pages 225–228.

J. Hajič. 2009. The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of the CoNLL*.

W. He, H. Wang, Y. Guo, and T. Liu. 2009. Dependency based chinese sentence realization. In *Proceedings of the ACL and of the IJCNLP of the AFNLP*, pages 809–816.

K. Knight and J. Graehl. 2005. An overview of probabilistic tree transducers for natural language processing. In *Sixth International Conference on Intelligent Text Processing and Computational Linguistics*. Lecture Notes in Computer Science.

K. Knight and V. Hatzivassiloglou. 1995. Two-level, many paths generation. In *Proceedings of the ACL*.

I. Langkilde and K. Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the COLING/ACL*, pages 704–710.

I. Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the Second INLG Conference*, pages 17–28.

B. Lavoie and O. Rambow. 1997. A fast and portable realizer for text generation systems. In *Proceedings of the 5th Conference on ANLP*.

V.I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics*, 10:707–710.

F. Mairesse, M Gašić, F. Juričíč, S Keizer, B. Thomson, K. Yu, and S. Young. 2010. Phrase-based statistical language generation using graphical models and active learning. In *Proceedings of the 48th Annual Meeting of the ACL*.

R. McDonald and F. Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In *In Proc. of EACL*, pages 81–88.

C. Mellish, D. Scott, L. Cahill, D. Paiva, R. Evans, and M. Reape. 2006. A reference architecture for natural language generation systems. *Natural Language Engineering*, 12(1):1–34.

I.A. Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany.

I.A. Mel'čuk. 2001. *Communicative Organization in Natural Language : The Semantic-Communicative Structure of Sentences*. John Benjamins Publishing, Philadelphia.

A. Meyers, R. Reeves, C. Macleod, R. Szekely, V. Zielinska, B. Young, and R. Grishman. 2004. Annotating Noun Argument Structure for NomBank. In *Proceedings of LREC-2004*, Lisbon, Portugal.

M. Palmer, D. Gildea, and P. Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–105.

E. Ringger, M. Gamon, R.C. Moore, D. Rojas, M. Smets, and S. Corston-Oliver. 2004. Linguistically informed statistical models of constituent structure for ordering in sentence realization. In *Proceedings of COLING*, pages 673–679.

W. Rounds. 1970. Mappings and Grammars on Trees. *Mathematical Systems Theory*.

A. Stent, R. Prasad, and M. Walker. 2004. Trainable sentence planning for complex information presentation in spoken dialog systems. In *Proceedings of the 42th Annual Meeting of the ACL*.

J.W. Thatcher. 1970. Generalized Sequential Machine Maps. *Computer Systems*.

M.A. Walker, O.C. Rambow, and M. Rogati. 2002. Training a sentence planner for spoken dialogue using boosting. *Computer Speech and Language*, 16:409–433.

L. Wanner, S. Mille, and B. Bohnet. submitted. Do we need new semantic corpus annotation policies for deep statistical generation?

Y.W. Wong and R.J. Mooney. 2007. Generation by inverting a semantic parser that uses statistical machine translation. In *Proceedings of the HLT Conference of the NAACL*, pages 172–179.