<StuMaBa>: From Deep Representation to Surface

Bernd Bohnet [1], **Simon Mille** [2], **Benoît Favre** [3], **Leo Wanner**[2,4]
[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, {first-name.last-name}@upf.edu
[3]Laboratoire d'Informatique de l'Universite du Maine
{first-name.last-name}@lium.univ-lemans.fr
[4]Institució Catalana de Recerca i Estudis Avançats (ICREA)

1 Setup of the System

We realize the full generation pipeline, from the deep (= semantic) representation (SemR), over the shallow (= surface-syntactic) representation (SSyntR) to the surface. To account systematically for the non-isomorphic projection between SemR and SSyntR, we introduce an intermediate representation: the so-called *deep-syntactic* representation (DSyntR), which does not contain yet (all) function words (as SemR), but which already contains grammatical function relation labels (as SSyntR).[1]

The system thus realizes the following steps:

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

In addition, two auxiliary steps are carried out. The first one is part-of-speech tagging; it is carried out after step 3. The second one is introduction of commata; it is done after step 4.

Each step is implemented as a decoder that uses a classifier to select the appropriate operations. For the realization of the classifiers, we use Bohnet et al. (2010)'s implementation of MIRA (Margin Infused Relaxed Algorithm) (Crammer et al., 2006).

2 Sentence Realization

Sentence generation consists in the application of the previously trained decoders in sequence 1.–4., plus the two auxiliary steps.

[1]The DSyntR is inspired by the DSynt structures in (Mel'čuk, 1988), only that the latter are still "deeper".

Semantic Generation Our derivation of the DSynt-tree from an input Sem-graph is analogous to graph-based parsing algorithms (Eisner, 1996). It 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 Sem-graph x).

As in (Bohnet et al., 2011), the search is a beam search which creates a maximum spanning tree using "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 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 1 displays the algorithm for the generation of the DSyntR from the SemR. The algorithm performs a greedy search for the highest scoring tree. *extend-tree* is the central function of the algorithm. It expands a tree by one edge, selecting each time the highest scoring edge. The attachment point for an outgoing edge is any node; for an incoming edge, it can only be the top node of the built tree.

For score calculation, we use structured features composed of the following elements: (i) the **lemma**ta, (ii) the **dist**ance between the starting node

Proceedings of the 13th European Workshop on Natural Language Generation (ENLG), pages 232–235,
Nancy, France, September 2011.

Algorithm 1: Semantic generation

```
//(x_i, y_i) semantic graph and
// gold deep syntactic tree for training case only
// both trees contain an artifical root node
tree ← {} // empty tree
// search start edge
best ← −2^31
for all n_1 ∈ x_i do
  for all n_2 ∈ x_i & n_1 ≠ n_2 do
    for all l ∈ edge-labels do
      s ← score({(synt(n_1),synt(n_2),l)})
      if s > best then
        tree ← {(synt(n_1),synt(n_2),l)}
        best ← s ; root ← n_1
// computed remaining nodes to be added
rest ← nodes(x_i) − nodes(tree)
while rest ≠ ∅ do
  // extend tree: extend tree by one edge
  best ← −2^31
  for all n_r ∈ rest do
    for all n_t ∈ tree do
      for all l ∈ edge-labels do
        s ← score(tree, {(synt(n_t),synt(n_r),l)})
        if s > best then
          tree ← tree ∪ {(synt(n_t),synt(n_r),l)}
          best ← s ; rest ← rest - n_r
          continue with while
    // check for new root
    s ← score(tree, {(synt(n_r),synt(root),l)})
    if s > best then
      tree ← tree ∪ {(synt(n_r),synt(root),l)}
      root ← n_r ; best ← s ; rest ← rest - n_r
      continue with while
return tree
TRAINING: if predicted tree ≠ gold tree
then update weight vector in accordance with the trees
```

s and the target node t, (iii) the **dir**ection of the path (if the path has a direction), (iv) the sorted **bag** of in-going 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/has not a specific feature.

Deep-Syntactic Generation: Since the DSyntR contains by definition only content words, function words must be introduced during the DSyntR–SSyntR generation passage in order to obtain a fully spelled out syntactic tree.

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

Table 1: Selected feature templates for the SemR → DSyntR mapping ('s' = "source node", 't' = "target node")

Algorithm 2: DSynt Generation

```
//(x_i, y_i^g) the deep syntactic tree
// and gold surface syntactic tree for training case only
// R set of rules
// traverse the tree 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)
  // during training, we update the weight vector
  // if the rules are not equal to the gold rules,
  // select the rules which insert a node into the tree
  // or a new node label
  node-insert-rules ← select-node-rules(node,x_i,R)
  // during training, we update here the weight vector
  y_i ← apply (edge-insert-rules,y_i)
```

For this passage, we use a tree transducer for which we automatically derive 27 rules by comparing a gold standard set of DSynt structures and SSynt dependency trees. The rules are of the following three types: 1) Introduction of an edge and a node: X ⇒ X *label*$_s$ → Y; as 'X ⇒ X *P*→ ',' ; 2) Introduction of a new node and edges between two nodes: X *label*$_d$→ Y ⇒ X *label*^1_s → N *label*^2_s → Y, as 'X *OPRD*→ Y ⇒ X *OPRD*→ 'to' *IM*→ Y'; 3) Introduction of a new node label: X ⇒ N, as ' 'LOCATION' ⇒ 'on' ' .

Discriminative classifiers are trained for each of

the three rule types such that they either select a specific rule or NONE (with "NONE" meaning that no rule is to be applied). Algorithm 2 displays the algorithm for the generation of the SSyntR from the DSyntR.

Tree-based Part-of-Speech tagging: For linearization, i.e., word order determination, information on the part of speech (PoS) of the node labels of the SSynt tree is needed. For this purpose, we developed a tree-based PoS-tagger. The tagger works similarly to a standard PoS-tagger, except that we do not use (i) features derived from the context of the word (i.e., of the token to its left and of the token to its right) for which the PoS tag is being sought; and (ii) wordforms (since a semantic graph and thus also the trees derived from it in the course of generation are annotated only with lemmata and semantic grammemes). However, we use features derived from the SSynt structure that we obtained in the previous step and the grammemes provided in the semantic graph from which we start. As classifier, we use a linear support vector machine with averaging.

Linearization: For linearization and morphologization, we use a similar technique as Bohnet et al. (2010). Linearization is a beam search for optimal linearization according to a local and a global score functions.

In order to derive the word order, we use a bottom-up linearization method. We start by ordering the words of sub-trees in which the children do not have children themselves. We continue then with sub-trees in which all sub-trees are already ordered. This method allows us to use the order of the sub-trees to derive features. We order each sub-tree that includes a head and its children: (1) The algorithm creates sets of nodes for each sub-tree in the syntactic tree that contain the children of the node and the node itself. (2) The linearization algorithm orders the list of nodes in such that the node list of the children are ordered first. (3) Complete sentences are built by introducing the list of nodes in which only the head was included so far. The algorithm builds thus n-best lists of ordered sentences by adding ordered parts left-to-right.

Morphologization: Morphologization 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 the scripts to obtain the wordforms.

System 1	
Mapping	Value
Semantics→Deep-Syntax (ULA/LAS)	99.0/95.1
Deep-Syntax→Surface-Syntax (correct)	98.6
Tree-based PoS tagging	97.8
Syntax→ Topology (% sent. eq. to reference)	54.2
Topology→Morphology (accuracy)	98.2
All stages from **deep representation**	
BLEU	76.4
NIST	13.45
All stages from **shallow representation**	
BLEU	88.7
NIST	13.89
System 2	
Semantics→Deep-Syntax (ULA/LAS)	99.0/95.1
Deep-Syntax→Surface-Syntax (correct)	98.9
Tree-based PoS tagging	98.2
Syntax→ Topology (% sent. eq. to reference)	57.7
Topology→Morphology (accuracy)	98.2
All stages from **deep representation**	
BLEU	79.6
NIST	13.55
All stages from **shallow representation**	
BLEU	89.6
NIST	13.93

Table 2: Performance of our realizer on the development set.

3 Evaluation

We submitted two systems ("System 1" and "System 2"). System 2 was a late submission. System 1 can be considered as a baseline system. System 2 introduces commata more accurately because of an improved feature set. In addition, System 2 uses the word order of the children of a node as context to derive features for the linearization. Furthermore, it uses a language model to rerank output sentences. For the language model, we use a 5-gram model with *Kneser-Ney* smoothing derived from 11 million sentences, cf. (Kneser and Ney, 1995). Table 2 displays the figures obtained for both the realization stages in isolation and the entire pipeline.[2]

[2]After the first submission of our system, we corrected a bug. As a consequence, the results improved. The bug occurred during the mapping of the data from HFG-format into CoNLL

References

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, Beijing.

B. Bohnet, S. Mille, and L. Wanner. 2011. Statistical Language Generation from Semantic Structures. In *Proceedings of the International Conference on Dependency Linguistics*, Barcelona.

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.

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, Copenhagen.

R. Kneser and H. Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics,Speech and Signal Processing.*

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

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

2009 format, which was carried out to obtain training data in the format of our generator. It consisted in accessing a number of wrong columns and/or subcolumns of the annotation for a few features (which led to, e.g., the use of wordforms instead of lemmata).