Deep-Syntactic Parsing

Miguel Ballesteros[1], **Bernd Bohnet**[2], **Simon Mille**[1], **Leo Wanner**[1,3]
[1]Natural Language Processing Group, Pompeu Fabra University, Barcelona, Spain
[2]School of Computer Science, University of Birmingham, United Kingdom
[3]Catalan Institute for Research and Advanced Studies (ICREA)
[1,3]{name.lastname}@upf.edu [2]bohnetb@cs.bham.ac.uk

Abstract

"Deep-syntactic" dependency structures that capture the argumentative, attributive and coordinative relations between full words of a sentence have a great potential for a number of NLP-applications. The abstraction degree of these structures is in-between the output of a syntactic dependency parser (connected trees defined over all words of a sentence and language-specific grammatical functions) and the output of a semantic parser (forests of trees defined over individual lexemes or phrasal chunks and abstract semantic role labels which capture the argument structure of predicative elements, dropping all attributive and coordinative dependencies). We propose a parser that delivers deep syntactic structures as output.

1 Introduction

Surface-syntactic structures (SSyntSs) as produced by data-driven syntactic dependency parsers are *per force* idiosyncratic in that they contain governed prepositions, determiners, support verb constructions and language-specific *grammatical functions* such as, e.g., SBJ, OBJ, PRD, PMOD, etc. (Johansson and Nugues, 2007). For many NLP-applications, including machine translation, paraphrasing, text simplification, etc., such a high idiosyncrasy is obstructive because of the recurrent divergence between the source and the target structures. Therefore, the use of more abstract "syntactico-semantic" structures seems more appropriate. Following Mel'čuk (1988), we call these structures *deep-syntactic structures* (DSyntSs). DSyntSs are situated between SSyntSs and PropBank- (Palmer et al., 2005) or Semantic Frame-like structures (Fillmore et al., 2002). Compared to SSyntSs, they have the advantage to abstract from language-specific grammatical idiosyncrasies. Compared to PropBank and Semantic Frame stuctures, they have the advantage to be connected and complete, i.e., capture all argumentative, attributive and coordinative dependencies between the meaningful lexical items of a sentence, while PropBank and Semantic Frame structures are not always connected, may contain either individual lexical items or phrasal chunks as nodes, and discard attributive and coordinative relations (be they within the chunks or sentential). In other words, they constitute incomplete structures that drop not only idiosyncratic, functional but also meaningful elements of a given sentence and often contain dependencies between chunks rather than individual tokens. Therefore, we propose to put on the research agenda the task of deep-syntactic parsing and show how a DSyntS is obtained from a SSynt dependency parse using data-driven tree transduction in a pipeline with a syntactic parser.[1] In Section 2, we introduce SSyntSs and DSyntSs and discuss the fundamentals of SSyntS–DSyntS transduction. Section 3 describes the experiments that we carried out on Spanish material, and Section 4 discusses their outcome. Section 5 summarizes the related work, before in Section 6 some conclusions and plans for future work are presented.

2 Fundamentals of SSyntS–DSyntS transduction

Before we set out to discuss the principles of the SSyntS–DSynt transduction, we must specify the DSyntSs and SSyntSs as used in our experiments.

[1]The term 'tree transduction' is used in this paper in the sense of Rounds (1970) and Thatcher (1970) to denote an extension of *finite state transduction* (Aho, 1972) to trees.

Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 1402–1413, Dublin, Ireland, August 23-29 2014.

2.1 Defining SSyntS and DSyntS

SSyntSs and DSyntSs are directed, node- and edge-labeled dependency trees with standard feature-value structures (Kasper and Rounds, 1986) as node labels and dependency relations as edge labels.

The features of the node labels in **SSyntSs** are lex_{ssynt}, and "syntactic grammemes" of the value of lex_{ssynt}, i.e., *number*, *gender*, *case*, *definiteness*, *person* for nouns and *tense*, *aspect*, *mood* and *voice* for verbs. The value of lex_{ssynt} can be any (either full or functional) lexical item; in graphical representations of SSyntSs, usually only the value of lex_{ssynt} is shown. The edge labels of a SSyntS are grammatical functions 'subj', 'dobj', 'det', 'modif', etc. In other words, SSyntSs are syntactic structures of the kind as encountered in the standard dependency treebanks; cf., e.g., dependency version of the Penn TreeBank (Johansson and Nugues, 2007) for English, Prague Dependency Treebank for Czech (Hajič et al., 2006), Ancora for Spanish (Taulé et al., 2008), Copenhagen Dependency Treebank for Danish (Buch-Kromann, 2003), etc. In formal terms that we need for the outline of the transduction below, a SSyntS is defined as follows:

Definition 1 (SSyntS) *An SSyntS of a language $\mathcal{L}$ is a quintuple $T_{SS} = \langle N, A, \lambda_{l_s \to n}, \rho_{r_s \to a}, \gamma_{n \to g} \rangle$ defined over all lexical items L of $\mathcal{L}$, the set of syntactic grammemes G_{synt}, and the set of grammatical functions R_{gr}, where*

- *the set N of nodes and the set A of directed arcs form a connected tree,*
- *$\lambda_{l_s \to n}$ assigns to each $n \in N$ an $l_s \in L$,*
- *$\rho_{r_s \to a}$ assigns to each $a \in A$ an $r \in R_{gr}$, and*
- *$\gamma_{n \to g}$ assigns to each $\lambda_{l_s \to n}(n)$ a set of grammemes $G_t \in G_{synt}$.*

The features of the node labels in **DSyntSs** as worked with in this paper are lex_{dsynt} and "semantic grammemes" of the value of lex_{dsynt}, i.e., *number* and *determination* for nouns and *tense*, *aspect*, *mood* and *voice* for verbs.[2] In contrast to lex_{ssynt} in SSyntS, DSyntS's lex_{dsynt} can be any *full*, but not a *functional* lexeme. In accordance with this restriction, in the case of *look after a person*, AFTER will not appear in the corresponding DSyntS; it is a functional (or governed) preposition (so are TO or BY, in Figure 1).[3] In contrast, AFTER in *leave after the meeting* is a full lexeme; it will remain in the DSyntS because there it has its own meaning of "succession in time". The edge labels of a DSyntS are language-independent "deep-syntactic" relations I,...,VI, ATTR, COORD, APPEND. 'I',...,'VI' are argument relations, analogous to $A0$, $A1$, etc. in the PropBank annotation. 'ATTR' subsumes all (circumstantial) ARGM-x PropBank relations as well as the modifier relations not captured by the PropBank and FrameNet annotations. 'COORD' is the coordinative relation as in: *John*-COORD→*and*-II→*Mary*, *publish*-COORD→*or*-II→*perish*, and so on. APPEND subsumes all parentheticals, interjections, direct addresses, etc., as, e.g., in *Listen, John!*: *listen*-APPEND→*John*. DSyntSs thus show a strong similarity with PropBank structures, with four important differences: (i) their lexical labels are not disambiguated; (ii) instead of circumstantial thematic roles of the kind ARGM-LOC, ARGM-DIR, etc. they use a unique ATTR relation; (iii) they capture all existing dependencies between meaningful lexical nodes; and (iv) they are connected.[4] A number of other annotations have resemblance with DSyntSs; cf. (Ivanova et al., 2012) for an overview of deep dependency structures. Formally, a DSyntS is defined as follows:

Definition 2 (DSyntS) *An DSyntS of a language $\mathcal{L}$ is a quintuple $T_{DS} = \langle N, A, \lambda_{l_s \to n}, \rho_{r_s \to a}, \gamma_{n \to g} \rangle$ defined over the full lexical items L_d of $\mathcal{L}$, the set of semantic grammemes G_{sem}, and the set of deep-syntactic relations R_{dsynt}, where*

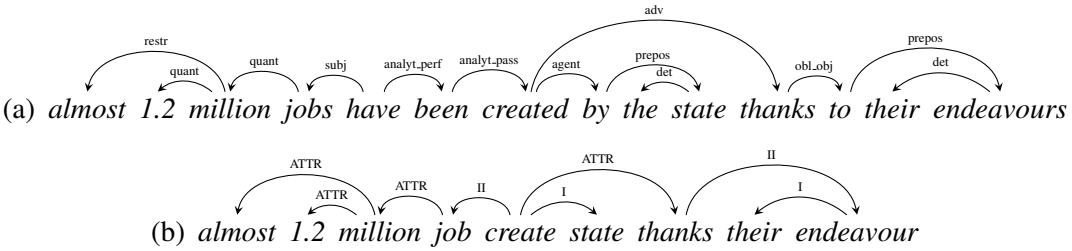
- *the set N of nodes and the set A of directed arcs form a connected tree,*
- *$\lambda_{l_s \to n}$ assigns to each $n \in N$ an $l_s \in L_d$,*
- *$\rho_{r_s \to a}$ assigns to each $a \in A$ an $r \in R_{dsynt}$, and*
- *$\gamma_{n \to g}$ assigns to each $\lambda_{l_s \to n}(n)$ a set of grammemes $G_t \in G_{sem}$.*

Consider in Figure 1 an example for an SSyntS and its corresponding DSyntS.

[2] Most of the grammemes have a semantic and a surface interpretation; see (Mel'čuk, 2013).

[3] Functional lexemes also include auxiliaries (e.g. HAVE, or BE when it is not a copula), and definite and indefinite determiners (THE, A); see Figure 1).

[4] Our DSyntSs are thus DSyntSs as used in the Meaning-Text Theory (Mel'čuk, 1988), only that our DSyntSs do not disambiguate lexical items and do not use *lexical functions* (Mel'čuk, 1996).



(a) *almost 1.2 million jobs have been created by the state thanks to their endeavours*

(b) *almost 1.2 million job create state thanks their endeavour*

Figure 1: An SSyntS (a) and its corresponding DSyntS (b)

2.2 Fleshing out the SSyntS–DSyntS transduction

It is clear that the SSyntS and DSyntS of the same sentence are not isomorphic. The following correspondences between the SSyntS S_{ss} and DSyntS S_{ds} of a sentence need to be taken into account during SSyntS–DSyntS transduction:

(i) a node in S_{ss} is a node in S_{ds};
(ii) a relation in S_{ss} corresponds to a relation in S_{ds};
(iii) a fragment of the S_{ss} tree corresponds to a single node in S_{ds};
(iv) a relation with a dependent node in S_{ss} is a grammeme in S_{ds};
(v) a grammeme in S_{ss} is a grammeme in S_{ds};
(vi) a node in S_{ss} is conflated with another node in S_{ds}; and
(vii) a node in S_{ds} has no correspondence in S_{ss}.

The grammeme correspondences (iv) and (v) and the "pseudo" correspondences in (vi) and (vii)[5] are few or idiosyncratic and are best handled in a rule-based post-processing stage. The main task of the SSyntS–DSyntS transducer is thus to cope with the correspondences (i)–(iii). For this purpose, we can view both SSyntS and DSyntS as vectors indexed in terms of two-dimensional matrices $I = N \times N$ (N being the set of nodes of a given tree $1, \ldots, m$), with $I(i,j) = \rho(n_i, n_j)$, if $n_i, n_j \in N$ and $(n_i, n_j) \in A$ and $I(i,j) = 0$ otherwise (where '$\rho(n_i, n_j)$' is the function that assigns to an edge a relation label and $i, j = 1, \ldots, m; i \neq j$ are nodes of the tree). That is, for a given SSyntS, the matrix $I(i,j)$ contains in the cells (i,j), $i,j = 1, \ldots, m$, the names of the SSynt-relations between the nodes n_i and n_j , and '0' otherwise, while for a given DSyntS, the cells of its matrix I_D contain DSyntS-relations.

Starting from the matrix I_S of a given SSyntS, the task is therefore to obtain the matrix I_D of the corresponding DSyntS, that is, to identify correspondences between i/j, (i,j) and groups of (i,j) of I_S with i'/j' and (i',j') of I_D; see (i)–(iii) above. In other words, the task consists in identifying and removing all functional lexemes, and attach correctly the remaining nodes between them.[6]

As a "token chain→surface-syntactic tree" projection, this task can be viewed as a classification task. However, while the former is isomorphic, we know that the SSyntS–DSyntS projection is not. In order to approach the task to an isomorphic projection (and thus simplify its modelling), it is convenient to interpret SSyntS and the targeted DSyntS as collections of *hypernodes*:

Definition 3 (Hypernode) *Given a SSyntS S_s with its index matrix I_S (a DSyntS S_d with its index matrix I_D), a node partition p (with $|p| \geq 1$) of I_S (I_D) is a hypernode h_{s_i} (h_{d_i}) iff p corresponds to a partition p' (with $|p'| \geq 1$) of S_d (S_s).*

In this way, the SSyntS–DSyntS correspondence boils down to a correspondence between individual hypernodes and between individual arcs, and the transduction embraces the following three (classification) subtasks: 1. Hypernode identification, 2. DSynt tree construction, and 3. DSynt arc labeling, which are completed by a post-processing stage.

[5](vi) covers, e.g., reflexive verb particles such as *se* in Spanish, which are conflated in the DSyntS with the verb: *se*←aux_refl_dir-*conocer* vs. CONOCERSE 'know each other'; (vii) covers, e.g., the zero subject in pro-drop languages (which is absent in the SSyntS and present in the DSyntS).

[6]What is particularly challenging is the identification of functional prepositions: based on the information found in the corpus only, our system must decide if a given preposition is a full or a functional lexeme. That is, we do not resort to any external lexical resources.

1. Hypernode identification. The hypernode identification consists of a binary classification of the nodes of a given SSyntS as nodes that form a hypernode of cardinality 1 (i.e., nodes that have a one-to-one correspondence to a node in the DSyntS) vs. nodes that form part of a hypernode of cardinality > 1. In practice, hypernodes of type one will be formed by: 1) noun nodes that do not govern determiner or functional preposition nodes, 2) full verb nodes that are not governed by any auxiliary verb nodes and that do not govern any functional preposition node, adjective nodes, adverbial nodes, and semantic preposition nodes. Hypernodes of type two will be formed by: 1) noun nodes + determiner / functional preposition nodes they govern, 2) verb nodes + auxiliary nodes they are governed by + functional preposition nodes they govern.

2. DSynt tree reconstruction. The outcome of the hypernode identification stage is thus the set $H_s = H_{s_{|p|=1}} \cup H_{s_{|p|>1}}$ of hypernodes of two types. With this set at hand, we can define an isomorphy function $\tau : H_s \rightarrow H_{d_{|p|=1}}$ (with $h_d \in H_{d_{|p|=1}}$ consisting of $n_d \in N_{ds}$, i.e., the set of nodes of the target DSyntS). τ is the identity function for $h_s \in H_{s_{|p|=1}}$. For $h_s \in H_{s_{|p|>1}}$, τ maps the functional nodes in h_s onto grammemes (attribute-value pairs) of the lexically meaningful node in h_d and identifies the lexically meaningful node as head. Some of the dependencies of the obtained nodes $n_d \in N_{ds}$ can be recovered from the dependencies of their sources. Due to the projection of functional nodes to grammemes (which can be also seen as node removal), some dependencies will be also missing and must be introduced. Algorithm 1 recalculates the dependencies for the target DSyntS S_d, starting from the index matrix I_S of SSyntS S_s to obtain a connected tree.

Algorithm 1: DSyntS tree reconstruction

for $\forall n_i \in N_d$ **do**
  if $\exists n_j : (n_j, n_i) \in S_s \wedge \tau(n_j) \in N_d$ **then**
  $(n_j, n_i) \rightarrow S_d$ // the equivalent of the head node of n_i is included in DSyntS
  else if $\exists n_j, n_a : (n_j, n_i) \in S_s \wedge \tau(n_j) \notin N_d \wedge$
      $\tau(n_a) \in N_d$ **then**
      //n_a is the first ancestor of n_j that has an equivalent in DSyntS
      //the equivalent of the head node of n_i is not included in DSyntS, but the ancestor n_a is
      $(n_a, n_i) \rightarrow S_d$
    else
      //the equivalent of the head node of n_i is not included in DSyntS, but several ancestors of it are
      $n_b := BestHead(n_i, S_s, S_d)$
      $(n_b, n_i) \rightarrow S_d$
endfor

$BestHead$ recursively ascends S_s from a given node n_i until it encounters one or several head nodes $n_d \in N_{ds}$. In case of several encountered head nodes, the one which governs the highest frequency dependency is returned.

3. Label Classification. The tree reconstruction stage produces a "hybrid" connected dependency tree $S_{s \rightarrow d}$ with DSynt nodes N_{ds}, and arcs A_s labelled by SSynt relation labels, i.e., an index matrix we can denote as I^-, whose cells (i, j) contain SSynt labels for all $n_i, n_j \in N_{ds} : (n_i, n_j) \in A_s$ and '0' otherwise. The next and last stage of SSynt-to-DSyntS transduction is thus the projection of SSynt relation labels of $S_{s \rightarrow d}$ to their corresponding DSynt labels, or, in other words, the mapping of I^- to I_D of the target DSyntS.

4. Postprocessing. As mentioned in Section 2, there is a limited number of idiosyncratic correspondences between elements of SSyntS and DSyntS (the correspondences (iv–vii) which can be straightforwardly handled by a rule-based postprocessor because (a) they are non-ambiguous, i.e., $a \leftrightarrow b, c \leftrightarrow d \Rightarrow a = b \wedge c = d$, and (b) they are few. Thus, only determiners and auxiliaries in SSyntS map onto a grammeme in DSyntS, both SSyntS and DSyntS count with less than a dozen grammemes, etc.

3 Experiments

In order to validate the outlined SSyntS–DSyntS transduction and to assess its performance in combination with a surface dependency parser, i.e., starting from plain sentences, we carried out a number of

experiments in which we implemented the transducer and integrated it into a pipeline shown in Figure 2.

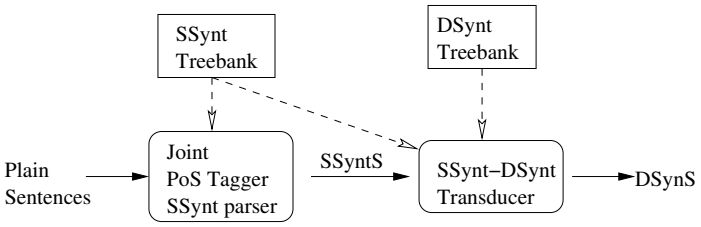


Figure 2: Setup of a deep-syntactic parser

For our experiments, we use the AnCora-UPF SSyntS and DSyntS treebanks of Spanish (Mille et al., 2013) in CoNLL format, adjusted for our needs. In particular, we removed from the 79-tag SSyntS treebank the semantically and information structure influenced relation tags to obtain an annotation granularity closer to the ones used for previous parsing experiments (55 relation tags, see (Mille et al., 2012)).

Our development set consisted of 219 sentences (3271 tokens in the DSyntS treebank and 4953 tokens in the SSyntS treebank), the training set of 3036 sentences (57665 tokens in the DSyntS treebank and 86984 tokens in the SSyntS treebank), and the *test set* held-out for evaluation of 258 sentences (5641 tokens in the DSyntS treebank and 8955 tokens in the SSyntS treebank).

To obtain the SSyntS, we use Bohnet and Nivre (2012)'s transition-based parser, which combines lemmatization, PoS tagging, and syntactic dependency parsing—tuned and trained on the respective sets of the SSyntS treebank. Cf. Table 1 for the performance of the parser on the development set.

POS	LEMMA	LAS	UAS
96.14	91.10	78.64	86.49

Table 1: Results of Bohnet and Nivre's surface-syntactic parser on the development set

In what follows, we first present the realization of the SSyntS–DSyntS transducer and then the realization of the baseline.

3.1 SSyntS–DSyntS transducer

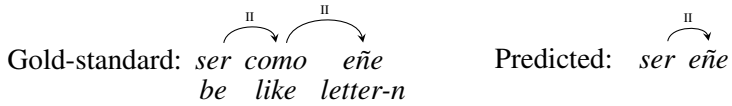
As outlined in Section 2.2, the SSyntS–DSyntS transducer is composed of three submodules and a post-processing stage:

1. Hypernode identification. For the hypernode identification, we trained a binary polynomial (degree 2) SVM from LIBSVM (Chang and Lin, 2001). The SVM allows both features related to the processed node and higher-order features, which can be related to the head node of the processed node or to its sibling nodes. After several feature selection trials, we chose the following features for each node n:

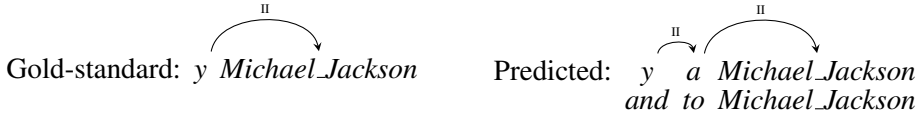
- lemma or stem of the label of n,
- label of the relation between n and its head,
- surface PoS of n's label (the SSynt and DSyntS treebanks distinguish between surface and deep PoS),
- label of the relation between n's head to its own head,
- surface PoS of the label of n's head node.

After an optimization round of the parameters available in the SVM implementation, the hypernode identification achieved over the gold development set 99.78% precision and 99.02% recall (and thus 99.4% F1). That is, only very few hypernodes are not identified correctly. The main error source are *governed prepositions*: the classifier has to learn when to assign a preposition an own hypernode (i.e., when it is lexically meaningful) and when it should be included into the hypernode of the governor (i.e., when it is functional). Our interpretation is that the features we use for this task are appropriate, but that the training data set is too small. As a result, some prepositions are erroneously left out from or introduced into the DSyntS.

2. Tree reconstruction. The implementation of the tree reconstruction module shows an unlabelled dependency attachment precision of 98.18% and an unlabelled dependency attachment recall of 97.43% over the gold development set. Most of the errors produced by this module have their origin in the previous module, i.e., hypernode identification. When a node has been incorrectly removed, the module errs in the attachment because it cannot use the node in question as the destination or the origin of a dependency, as it is the case in the gold-standard annotation:



When a node has erroneously not been removed, no dependencies between its governor and its dependent can be established since DSyntS must remain a tree (which gives the same LAS and UAS errors as when a node has been erroneously removed):



3. Relation label classification. For relation label classification, we use a multiclass linear SVM. The label classification depends on the concrete annotation schemata of the SSyntS and DSyntS treebanks on which the parser is trained. Depending on the schemata, some DSynt relation labels may be easier to derive from the original SSyntS relation labels than others. Table 2 lists all SSynt relation labels that have a straightforward mapping to DSyntS relation labels in the used treebanks, i.e., neither their dependent nor their governor are removed, and the SSyntS label always maps to the same DSynt label.

SSynt	DSynt	SSynt	DSynt	SSynt	DSynt	SSynt	DSynt
abbrev	ATTR	aux_refl_indir	III	dobj_clitic	II	prepos_quot	II
abs_pred	ATTR	bin_junct	ATTR	dobj_quot	II	prolep	APPEND
adv	ATTR	compl1	II	elect	ATTR	quant	ATTR
adv_mod	ATTR	compl2	III	juxtapos	APPEND	quasi_coord	COORD
agent	I	compl_adnom	ATTR	modal	II	quasi_subj	I
appos	ATTR	coord	COORD	modif	ATTR	relat	ATTR
attr	ATTR	copul	II	num_junct	COORD	restr	ATTR
aux_phras	—	copul_clitic	II	obj_copred	ATTR	sequent	ATTR
aux_refl_dir	II	copul_quot	II	prepos	II	subj	I
						subj_copred	ATTR

Table 2: Straightforward SSynt to DSyntS mappings

Table 3 shows SSyntS relation–DSyntS relation label correspondences that are not straightforward.

SSynt DepRel$_A$	Mapping to DSynt
analyt_fut	remove Gov and Dep; add tense=FUT
analyt_pass	remove Gov; invert I and II; add voice=PASS
analyt_perf	remove Gov; add tense=PAST
analyt_progr	remove Gov; add tem_constituency=PROGR
aux_refl_lex	remove Dep; add *se* at the end of Gov's lemma
aux_refl_pass	remove Dep; invert I and II; add voice=PASS
compar	remove Dep if conjunction
compar_/coord_/sub_conj	remove Dep if governed preposition
det	**IF** Dep=el—un **THEN** remove Dep; add definiteness=DEF/INDEF
	IF Dep=possessive **THEN** DepRel ATTR‖I‖II‖III
	IF Dep=other **THEN** DepRel ATTR
dobj	remove Dep if governed preposition
iobj	remove Dep if governed preposition; DepRel II‖III‖IV‖V‖VI
iobj_clitic	DepRel II‖III‖IV‖V‖VI
obl_compl	remove Dep if governed preposition; DepRel I‖II‖III‖IV‖V‖VI
obl_obj	remove Dep if governed preposition; DepRel II‖III‖IV‖V‖VI
punc	—
punc_init	—

Table 3: Complex SSynt to DSynt mappings

The final set of features selected for label classification includes: (i) lemma of the dependent node, (ii) dependency relation to the head of the dependent node, (iii) dependency relation label of the head node to its own head, (iv) dependency relation to the head of the sibling nodes of the dependent node, if any.

After an optimization round of the parameter set of the SVM-model, relation labelling achieved 94.00% label precision and 93.28% label recall on the development set. The recall is calculated considering all the nodes that are included in the gold standard. The error sources for relation labelling were mostly the dependencies that involved possessives and the various types of objects (see Table 3) due to their differing valency. For instance, the relation *det* in *su*$\leftarrow det-$*coche* 'his/her car' and *su*$\leftarrow det-$*llamada* 'his/her phone call' have different correspondences in DSyntS: *su*$\leftarrow ATTR-$*coche* vs. *su*$\leftarrow I-$*llamada*. That is, the DSyntS relation depends on the lexical properties of the governor.[7] Once again, more training data is needed in order to classify better those cases.

4. Postprocessing In the postprocessing stage for Spanish, the following rules capture non-ambiguous correspondences between elements of the SSynt-index matrix $I_S = N_s \times N_s$ and DSyntS index matrix $I_D = N_d \times N_d$, with $n_s \in N_s$ and $n_d \in N_d$, and n_s and n_d corresponding to each other (we do not list here identity correspondences such as between the number grammemes of n_s and n_d):

- if n_s is dependent of *analyt_pass* or *analyt_refl_pass* relation, then the voice grammeme in n_d is *PASS*;
- if n_s is dependent of *analyt_progr*, then the voice grammeme in n_d is *PROGR*;
- if n_s is dependent of *analyt_refl_lex*, then add the particle -SE as suffix of node label (word) of d_d;
- if any of the children of n_s is labelled by one of the tokens UN 'a$_{masc}$', UNA 'a$_{fem}$', UNOS 'some$_{masc}$' or UNAS 'some$_{fem}$', then the definiteness grammeme in n_d *INDEF*, otherwise it is *DEF*;
- if the n_s label is a finite verb and n_s does not govern a *subject* relation, then add to I' the relation $n_d - I \rightarrow n'_d$, with n'_d being a newly introduced node.

3.2 Baseline

As point of reference for the evaluation of the performance of our SSyntS–DSyntS transducer, we use a rule-based baseline that carries out the most direct transformations extracted from Tables 2 and 3. The baseline detects hypernodes by directly removing all the nodes that we are sure need to be removed, i.e. punctuation and auxiliaries. The nodes that are only *potentially* to be removed, i.e., all dependents of DepRels that have a possibly governed preposition or conjunction in Table 3, are left in the DSyntS. The new relation labels in the DSyntS are obtained by selecting the label that is most likely to substitute the SSyntS relation label according to classical grammar studies. The rules of the rule-based baseline look as follows:

```
1  if (deprel==abbrev) then deep_deprel=ATTR
2  if (deprel==obl_obj) then deep_deprel=II
...
n  if (deprel==punc) then remove(current_node)
```

4 Results and Discussion

Let us look in this section at the performance figures of the SSyntS parser, the SSyntS–DSyntS transducer, and the sentence–DSyntS pipeline obtained in the experiments.

4.1 SSyntS–DSyntS transducer results

In Table 4, the performance of the subtasks of the SSyntS–DSyntS transducer is contrasted to the performance of the baselines; the evaluation of the postprocessing subtask is not included because the one-to-one projection of SSyntS elements to DSyntS guarantees an accuracy of 100% of the operations performed. The transducer has been applied to the gold standard test set, which is the held-out test set, with gold standard PoS tags, lemmas and dependency trees. It outputs in total 5610 nodes; the rule-based baseline outputs 8653 nodes. As mentioned in Section 3, our gold standard includes 5641 nodes.

[7] Note that lexemes are not generalized: a verb and its corresponding noun (e.g., *construct*/*construction*) are considered distinct lexemes.

Hyper-Node Detection		
Measure	Rule-based Baseline	Tree Transducer
p	64.31 (5565/8653)	99.79 (5598/5610)
r	98.65 (5565/5641)	99.24 (5598/5641)
$F1$	77.86	99.51

Attachment and Labelling		
Measure	Rule-based Baseline	Tree Transducer
LAP	50.02 (4328/8653)	91.07 (5109/5610)
UAP	53.05 (4590/8653)	98.32 (5516/5610)
LA-P	57.66 (4989/8653)	92.37 (5182/5610)
LAR	76.72 (4328/5641)	90.57 (5109/5641)
UAR	81.37 (4590/5641)	97.78 (5516/5641)
LA-R	88.44 (4989/5641)	91.86 (5182/5641)

Table 4: Performance of the SSyntS–DSyntS transducer and of the rule-based baseline over the gold-standard held-out test set (LAP: labelled attachment precision, UAP: unlabelled attachment precision, LA-P: label assignment precision, LAR: labelled attachment recall, UAR: Unlabelled attachment recall and LA-R: Label assignment recall)

Our data-driven SSyntS–DSyntS transducer is much better than the baseline with respect to all evaluation measures.[8] The transducer relies on distributional patterns identified in the training data set, and makes thus use of information that is not available for the rule-based baseline, which studies one node at a time. However, the rule-based baseline results also show that transduction that would remove a few nodes would provide results close to a 100% recall for the hypernode detection because a DSynt tree is a subtree of the SSynt tree (if we ignore the nodes introduced by post-processing). This is also evidenced by the labeled and attachment recall scores. The results of the transducer on the test and development sets are quite comparable. The hypernode detection is even better on the test set. The label accuracy suffers most from using unseen data during the development of the system. The attachment figures are approximately equivalent on both sets.

4.2 Results of deep-syntactic parsing

Let us consider now the performance of the complete DSynt parsing pipeline (PoS-tagger+surface-dependency parser → SSyntS–DSyntS transducer) on the held-out test set. Table 5 displays the figures of the Bohnet and Nivre parser. The figures are in line with the performance of state-of-the-art parsers for Spanish (Mille et al., 2012).

POS	LEMMA	LAS	UAS
96.05	92.10	81.45	88.09

Table 5: Performance of Bohnet and Nivre's joint PoS-tagger+dependency parser trained on Ancora-UPF

Table 6 shows the performance of the pipeline when we feed the output of the syntactic parser to the rule-based baseline SSyntS–DSyntS module and the tree transducer. We observe a clear error propagation from the dependency parser (which provides 81.45% LAS) to the SSyntS–DSyntS transducer, which loses in tree quality more than 18%.

Hyper-Node Detection		
Measure	Baseline	Tree Transducer
p	63.87 (5528/8655)	97.07 (5391/5554)
r	98.00 (5528/5641)	95.57 (5391/5641)
$F1$	77.33	96.31

Labelling and Attachment		
Measure	Baseline	Tree Transducer
LAP	38.75 (3354/8655)	68.31 (3794/5554)
UAP	44.69 (3868/8655)	77.31 (4294/5554)
LA-P	49.66 (4298/8655)	80.47 (4469/5554)
LAR	59.46 (3354/5641)	67.26 (3794/5641)
UAR	68.57 (3868/5641)	76.12 (4294/5641)
LA-R	76.19 (4298/5641)	79.22 (4469/5641)

Table 6: Performance of the deep-syntactic parsing pipeline

5 Related Work

To the best of our knowledge, data-driven deep-syntactic parsing as proposed in this paper is novel. As *semantic role labeling* and *frame-semantic analysis*, it has the goal to obtain more semantically oriented structures than those delivered by state-of-the-art syntactic parsing. Semantic role labeling received considerable attention in the CoNLL shared tasks for syntactic dependency parsing in 2006 and 2007

[8] We also ran MaltParser by training it on the DSynt-treebank to parse the SSynt-test set; however, the outcome was too weak to be used as baseline.

(Buchholz and Marsi, 2006; Nivre et al., 2007), the CoNLL shared task for joint parsing of syntactic and semantic dependencies in 2008 (Surdeanu et al., 2008) and the shared task in 2009 (Hajič et al., 2009). The top ranked systems were pipelines that started with a syntactic analysis (as we do) and continued with predicate identification, argument identification, argument labeling, and word sense disambiguation; cf. (Johansson and Nugues, 2008; Che et al., 2009). At the end, a re-ranker that considers jointly all arguments to select the best combination was applied. Some of the systems were based on integrated syntactic and semantic dependency analysis; cf., e.g., (Gesmundo et al., 2009); see also (Lluís et al., 2013) for a more recent proposal along similar lines. However, all of them lack the ability to perform structural changes—as, e.g., introduction of nodes or removal of nodes necessary to obtain a DSyntS. Klimeš (2006)'s parser removes nodes (producing tectogrammatical structures as in the Prague Dependency Treebank), but is based on rules instead of classifiers, as in our case. The same applies to earlier works in the TAG-framework, as, e.g., in (Rambow and Joshi, 1997).

However, this is not to say that the idea of the surface→surface syntax→deep syntax pipeline is new. It goes back at least to Curry (1961) and is implemented in a number of more recent works; see, e.g., (de Groote, 2001; Klimeš, 2006; Bojar et al., 2008).

6 Conclusions and Future Work

We have presented a deep-syntactic parsing pipeline which consists of a state-of-the-art dependency parser and a novel SSyntS–DSyntS transducer. The obtained DSyntSs can be used in different applications since they abstract from language-specific grammatical idiosyncrasies of the SSynt structures as produced by state-of-the art dependency parsers, but still avoid the complexities of genuine semantic analysis.[9] DSyntS-treebanks needed for data-driven applications can be bootstrapped by the pipeline. If required, a SSyntS–DSyntS structure pair can be also mapped to a pure predicate-argument graph such as the DELPH-IN structure (Oepen, 2002) or to an approximation thereof (as the Enju conversion (Miyao, 2006), which keeps functional nodes), to an DRS (Kamp and Reyle, 1993), or to a PropBank structure. On the other hand, DSyntS-treebanks can be used for automatic extraction of deep grammars. As shown by Cahill et al. (2008), automatically obtained resources can be of an even better quality than manually-crafted resources. In this context, especially research in the context of CCGs (Hockenmeier, 2003; Clark and Curran, 2007) and TAGs (Xia, 1999) should be also mentioned.

To validate our approach with languages other than Spanish, we carried out an experiment on a Chinese SSyntS-DSyntS Treebank (training the DSynt-transducer on the outcome of the SSynt-parser). The results over predicted input showed an accuracy of about 75%, i.e., an accuracy comparable to the accuracy achieved for Spanish. We are also investigating multilingual approaches, such as the one proposed by McDonald et al. (2013).

In the future, we will carry out further in-depth feature engineering for the task of DSynt-parsing. It proved to be crucial in semantic role labelling and dependency parsing (Che et al., 2009; Ballesteros and Nivre, 2012); we expect it be essential for our task as well. Furthermore, we will join surface syntactic and deep-syntactic parsing we kept so far separate; see, e.g., (Zhang and Clark, 2008; Lluís et al., 2013; Bohnet and Nivre, 2012) for analogous proposals. Further research is required here since although joint models avoid error propagation from the first stage to the second, overall, pipelined models still proved to be competitive; cf. the outcome of CoNLL shared tasks.

The deep-syntactic parser described in this paper is available for downloading at `https://code.google.com/p/deepsyntacticparsing/`.

Acknowledgements

This work has been supported by the European Commission under the contract number FP7-ICT-610411. Many thanks to the three anonymous COLING reviewers for their very helpful comments and suggestions.

[9] The motivation to work with DSyntS instead of SSyntS is thus similiar to the motivation of the authors of the *Abstract Meaning Representation* (AMR) for Machine Translation (Banarescu et al., 2013), only that AMRs are considerably more semantic than DSyntSs.

References

Alfred V. Aho. 1972. *The theory of parsing, translation and, compiling*. Prentice Hall, Upper Saddle River, NJ.

Miguel Ballesteros and Joakim Nivre. 2012. MaltOptimizer: A System for MaltParser Optimization. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 12)*.

L. Banarescu, 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 7th Linguistic Annotation Workshop & Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *EMNLP-CoNLL*.

O. Bojar, S. Cinková, and J. Ptáček. 2008. Towards English-to-Czech MT via Tectogrammatical Layer. *The Prague Bulletin of Mathematical Linguistics*, 90:57–68.

Mathias Buch-Kromann. 2003. The Danish dependency treebank and the dtag treebank tool. In *2nd Workshop on Treebanks and Linguistic Theories (TLT), Sweden*, pages 217–220.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 149–164.

Aoife Cahill, Michael Burke, Ruth O'Donovan, Stefan Riezler, Josef van Genabith, and Andy Way. 2008. Wide-coverage deep statistical parsing using automatic dependency structure annotation. *Computational Linguistics*, 34(1):81–124.

Chih-Chung Chang and Chih-Jen Lin, 2001. *LIBSVM: A Library for Support Vector Machines*. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Wanxiang Che, Zhenghua Li, Yongqiang Li, Yuhang Guo, Bing Qin, and Ting Liu. 2009. Multilingual dependency-based syntactic and semantic parsing. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 49–54, Boulder, Colorado, June. Association for Computational Linguistics.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33:493–552.

R. Curry. 1961. Some logical aspects of grammatical structure. In R. Jakobson, editor, *Structure of Language and Its Mathematical Aspects*, pages 56–68. American Mathematical Society, Providence, RI.

Ph. de Groote. 2001. Towards abstract categorial grammar. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Charles J. Fillmore, Collin F. Baker, and Hiroaki Sato. 2002. The FrameNet database and software tools. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, volume IV, Las Palmas. LREC, LREC.

A. Gesmundo, J. Henderson, P. Merlo, and I.Titov. 2009. Latent variable model of synchronous syntactic-semantic parsing for multiple languages. In *CoNLL 2009 Shared Task., Conf. on Computational Natural Language Learning*, pages 37–42, Boulder, Colorado, USA.

Jan Hajič, Jarmila Panevová, Eva Hajičová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, Marie Mikulová, and Zdeňk Žabokrtský. 2006. Prague Dependency Treebank 2.0. Linguistic Data Consortium, Philadelphia.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL): Shared Task*, pages 1–18.

J. Hockenmeier. 2003. Parsing with generative models of predicate-argument structure. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 359–366, Sapporo, Japan.

Angelina Ivanova, Stephan Oepen, Lilja Øvrelid, and Dan Flickinger. 2012. Who did what to whom? a contrastive study of syntacto-semantic dependencies. In *Proceedings of the Sixth Linguistic Annotation Workshop*, pages 2–11, Jeju, Republic of Korea, July. Association for Computational Linguistics.

R. Johansson and P. Nugues. 2007. Extended constituent-to-dependency conversion for english. In J. Nivre, H.-J. Kaalep, K. Muischnek, and M. Koit, editors, *Proceedings of NODALIDA 2007*, pages 105–112, Tartu, Estonia.

Richard Johansson and Pierre Nugues. 2008. Dependency-based syntactic–semantic analysis with PropBank and NomBank. In *CoNLL 2008: Proceedings of the Twelfth Conference on Natural Language Learning*, pages 183–187, Manchester, United Kingdom.

H. Kamp and U. Reyle. 1993. *From Discourse to Logic*. Kluwer Academic Publishers, Dordrecht, NL.

R.T. Kasper and W.C. Rounds. 1986. A logical semantics for feature structures. In *Proceedings of the 24th annual meeting on Association for Computational Linguistics*, pages 257–266.

Václav Klimeš. 2006. *Analytical and Tectogrammatical Analysis of a Natural Language*. Ph.D. thesis, UFAL, MFF UK, Prague, Czech Republic.

Xavier Lluís, Xavier Carreras, and Lluís Màrquez. 2013. Joint arc-factored parsing of syntactic and semantic dependencies. *Transactions of the Association for Computational Linguistics*, pages 219–230.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 2013. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97.

Igor Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

Igor Mel'čuk. 1996. Lexical functions: A tool for the description of lexical relations in the lexicon. In L. Wanner, editor, *Lexical functions in lexicography and natural language processing*, pages 37–102. Benjamins Academic Publishers, Amsterdam.

Igor Mel'čuk. 2013. *Semantics: From meaning to text, Volume 2*. Benjamins Academic Publishers, Amsterdam.

Simon Mille, Alicia Burga, Gabriela Ferraro, and Leo Wanner. 2012. How does the granularity of an annotation scheme influence dependency parsing performance? In *Conference on Computational Linguistics, COLING 2012*.

Simon Mille, Alicia Burga, and Leo Wanner. 2013. AnCora-UPF: A Multi-Level Annotation of Spanish . In *Proceedings of the Second International Conference on Dependency Linguistics (DEPLING 2013).*

Yusuke Miyao. 2006. *From Linguistic Theory to Syntactic Analysis: Corpus-Oriented Grammar Development and Feature Forest Model*. Ph.D. thesis, University of Tokyo.

J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. 2007. The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 915–932.

Stephan Oepen. 2002. *Collaborative Language Engineering: A Case Study in Efficient Grammar-based Processing*. Stanford Univ Center for the Study.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank. *Computational Linguistics*, 31:71–106.

Owen Rambow and Aravind Joshi. 1997. A formal look at dependency grammar and phrase structure grammars, with special consideration of word-order phenomena. In L. Wanner, editor, *Recent Trends in Meaning-Text Theory*, pages 167–190. Benjamins Academic Publishers, Amsterdam.

W.C. Rounds. 1970. Mappings and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The conll 2008 shared task on joint parsing of syntactic and semantic dependencies. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177.

M. Taulé, M. Antònia Martí, and Marta Recasens. 2008. Ancora: Multilevel annotated corpora for Catalan and Spanish. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may. European Language Resources Association (ELRA).

J.W. Thatcher. 1970. Generalized sequential machine maps. *Journal of Computer and System Sciences*, 4(4):339–367.

F. Xia. 1999. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium*, pages 398–403, Beijing, China.

Yue Zhang and Stephen Clark. 2008. Joint word segmentation and POS tagging using a single perceptron. In *Proceedings of ACL-08: HLT*, pages 888–896, Columbus, Ohio, June. Association for Computational Linguistics.