Final Report:
Natural Language Generation in the Context of Machine Translation

Jan Hajič[1]
Martin Čmejrek[2]
Bonnie Dorr[3]
Yuan Ding[4]
Jason Eisner[5]
Dan Gildea[6]
Terry Koo[7]
Kristen Parton[8]
Gerald Penn[9]
Dragomir Radev[10]
Owen Rambow[11]
with contributions by[12]
Jan Cuřín
Jiří Havelka
Vladislav Kuboň
Ivona Kučerová

March 27, 2004

[1]Team leader, Charles University, Prague, Czech Republic
[2]Charles University, Prague, Czech Republic
[3]Affiliate team member, University of Maryland, USA
[4]University of Pennsylvania, USA
[5]Johns Hopkins University, USA
[6]Affiliate team member, University of Pennsylvania, USA
[7]MIT, USA
[8]Stanford Univerity, USA
[9]University of Toronto, Canada
[10]University of Michigan, USA
[11]University of Pennsylvania, USA
[12]All from Charles University, Prague, Czech Republic

Contents

List of Figures

Chapter 1

Introduction

Jan Hajič

Let's imagine a system for translating a sentence from a foreign language (say Czech, but substitute any of your favorite languages here) into your native language (say English). Such a system works as follows. It analyzes the foreign-language sentence to obtain a structural representation that captures its essence, i.e. "who did what to whom where." It then translates (or "transfers") the actors, actions, etc. into words in your language while "copying over" the deeper relationships between them. Finally it synthesizes a syntactically well-formed sentence that conveys the essential meaning of the original sentence.

Each step in this process is a hard technical problem, to which the best known solutions are either not adequate, or good enough only in narrow application domains, failing when applied to other domains. This summer, we have concentrated on improving one of these three steps, namely the synthesis (or generation), while having in mind that some of the core technologies can be applied to other parts of an MT system if the basic principle (i.e., using graph tree representations with only local transformations) is kept.

The target language for generation has been English, and the source language to the MT system has been a language of a different type (Czech). We have been able to produce automatically a fairly deeply analyzed sentence structure of the source language (albeit imperfect, of course). The incorporation of the deep analysis makes the whole approach very novel - so far no large-coverage translation system has tried to operate with such a structure, and the application to more diverse languages than, say, English and French, has made it an exciting enterprise.

Within the generation process, we have focused on the structural (syntactic) part, even though we had to develop a simple morphological generation module as well to be able to fully evaluate the final result. Statistical methods have been used throughout (except for a comparison to an existing state-of-the-art system in rule-based NL generation). Evaluation has been carried out using the BLEU metric and compared mainly to a baseline GIZA++-based system available previously (as created in the past by some of the team members).

Although the final results were below those generated by the simpler GIZA++ system, we believe that using deeper analysis for machine translation is the way to go. Needless to say, the system we compared to carefully researched and developed

systems was the result of only six weeks' work (albeit very intensive indeed).

Significant part of the pre-workshop work was devoted to resource creation for the complete Czech-English MT system. After further expansion and finalization, these resources will be made available publicly (in a much broader and complete form than used during the workshop).

In this report, we summarize the resources, findings, methods and experiments we have made during the 2002 JHU/CLSP Workshop. We hope that you will be able to find - at least - an inspiration for the future work on both Machine Translation and NL Generation here.

Chapter 2

Summary of Resources

Jan Hajič, Martin Čmejrek, Jan Cuřín, Jiří Havelka, Vladislav Kuboň

The MAGENTA system generates English analytical dependency trees from four different input options:

1. English tectogrammatical trees, automatically created from the Penn Treebank;
2. English tectogrammatical trees, human-annotated;
3. English tectogrammatical trees, automatically created from the Penn Treebank, improved by information from the Proposition Bank;
4. So-called "Czenglish" tectogrammatical trees, automatically created from the Czech input text. This input option represents an attempt to develop a full MT system based on dependency trees.

In the sequel, we summarize resources available before (Sections 2.1–2.4) as well as those created during the workshop (Section 2.5). Sections 2.6–2.11 describe automatic procedures used for preparation of both training and testing data for all four input options used in the MAGENTA system. Section 2.12 describes the process of filtering dictionaries used in the transfer procedure.

2.1 The Prague Dependency Treebank

The Prague Dependency Treebank project[1] aims at complex annotation of a corpus containing about 1.8M word occurrences (about 80,000 running text sentences) in Czech. The annotation, which is based on dependency syntax, is carried out in three steps: morphological, analytical, and tectogrammatical. The first two have been finished so far, presently, there are about 18,000 sentences tectogrammatically annotated. See (Hajič et al., 2001) and (Hajičová, Panevová, and Sgall, 2000) respectively for details of analytical and tectogrammatical annotation.

[1] version 1.0; LDC catalog no.: LDC2001T10, ISBN: 1-58563-212-0, `http://ufal.mff.cuni.cz/pdt`

2.2 The Penn Treebank

The Penn Treebank project[2] consists of about 1,500,000 tokens. Its bracketing style is based on constituent syntax, and comprises the surface syntactic structure, various types of null elements representing underlying positions for wh-movement, passive voice, infinitive constructions etc., and also predicate-argument structure markup. The largest component of the corpus consists of about 1 million words (about 40,000 sentences) from the Wall Street Journal newspaper. Only this part of the Penn Treebank corpus was used in the MAGENTA project.

2.3 The Proposition Bank

The PropBank project adds annotation of basic semantic propositions to the Penn Treebank corpus. For a verb, there is a list of syntactic frames (frameset), which have ever occurred in the annotated data; each position in the frame is associated with a semantic role in the predicate-argument structure of a given verb. The annotation started from the most frequent verbs (all occurrences of one verb are annotated in the same time) and continues to less frequent ones. See (Kingsbury, Palmer, and Marcus, 2002a) for further details.

2.4 Existing Czech-English parallel corpora

Two considerable resources of Czech-English parallel texts were available before the workshop and were mentioned in previous experiments related to statistical Czech-English MT: Reader's Digest Výběr ((Al-Onaizan et al., 1999), (Cuřín and Čmejrek, 2001)), and IBM AIX and AS/400 operating system guides and messages translations (Cuřín and Čmejrek, 1999). The Reader's Digest Výběr corpus (58,137 sentence pairs) contains sentences from a broad domain, very free translations, while IBM corpus (119,886 sentence pairs) contains domain specific sentences, literal, almost word-by-word translations.

According to the automatic sentence alignment procedure, only 57% sentence pairs from the Reader's Digest Výběr corpus are 1-1 matching sentence pairs, compared to 98% of 1-1 sentence pairs from the IBM corpus.

Both corpora are automatically morphologically annotated by automatic BH tagging tools (Hajič and Hladká, 1998). None of these corpora contain any syntactic annotation.

2.5 English to Czech translation of Penn Treebank

MAGENTA system uses syntactically annotated parallel texts as training data. Before the workshop preparation work started, there were no Czech-English parallel data manually syntactically annotated. We decided to translate a considerable part of the existing syntactically annotated English corpus (Penn Treebank) by human translators rather than to syntactically annotate existing Czech-English paral-

[2] version 3; LDC catalog no.: LDC99T42, ISBN: 1-58563-163-9

lel texts. The translators were asked to translate each English sentence as a single Czech sentence and also to stick to the original sentence construction when possible. Before the beginning of the workshop, 11,189 WSJ sentences were translated into Czech by human translators (Table 2.1). The translation project continues, still after the workshop, aiming at translating the whole Penn Treebank.

For both training and evaluation measured by BLEU metrics, about 500 sentences were retranslated back from Czech into English by 4 different translators (Table 2.2).

data category	#sentence pairs
training	6,966
devtest[1]	242
step devtest[2]	2,737
evaltest[1]	248
step evaltest[2]	996

Table 2.1: English - Czech sentence pairs

data category	#sentences
devtest	259
evaltest	256

Table 2.2: WSJ sentences retranslated from Czech to English by 4 different translators

2.6 English Analytical Dependency Trees

Apart from various input options, the tree-to-tree transducer used by the MAGENTA system always generates analytical trees. This section describes the automatic preparation of the output part of the training data from Penn Treebank.

2.6.1 Marking Heads in English

The concept of the head of a phrase is important when transforming the phrase tree topology into the dependency one. We used Jason Eisner's scripts (Eisner, 2001) for marking head constituents in each phrase.

2.6.2 Lemmatization of English

The Penn Treebank data contain manually assigned POS tags and this information substantially simplifies lemmatization. The lemmatization procedure just searches the list of all triples of word form, POS tag and lemma extracted from a large corpus, for a triple with a matching word form and POS and chooses the lemma from

[1] covered by 4 human retranslations into English

[2] not covered by human retranslations

```
wsj_1700.mrg:5::
(S (NP~-SBJ (DT @the the)
      (@NN @aim aim))
  (@VP (MD @would would)
    (@VP~ (@VB @be be)
       (S~-PRD (NP~-SBJ-1 (@-NONE- @* *))
           (@VP (TO @to to)
              (@VP~ (@VB @end end)
                 (NP~ (@NP (DT @the the)
                      (NN @guerrilla guerrilla)
                      (@NN @war war))
                   (PP (@IN @for for)
                     (NP~ (@NP (@NN @control control))
                        (PP (@IN @of of)
                          (NP~ (@NPR (@NNP @Cambodia Cambodia)))))))
                 (PP-MNR (@IN @by by)
                     (S~-NOM (NP~-SBJ (@-NONE- @*-1 *-1))
                         (@VP (@VBG @allowing allow)
                           (NP~ (DT @the the)
                              (@NPR (NNP @Khmer Khmer)
                                 (@NNP @Rouge Rouge)))
                           (NP~ (@NP (DT @a a)
                                (JJ @small small)
                                (@NN @share share))
                              (PP (@IN @of of)
                                (NP~ (@NN @power power)))))))))))
  (. @. .))
```

Figure 2.1: Example of a lemmatized sentence with marked heads: *"The aim would be to end the guerrilla war for control of Cambodia by allowing the Khmer Rouge a small share of power."*. Terminal nodes consist of a sequence of part-of-speech, word form, lemma, and a unique id. The names of the head constituent names start with @. (In the noun phrase *Khmer Rouge* the word *Rouge* was marked as head by mistake.)

this triple. A large corpus of English (365M words, 13M sentences) was automatically POS tagged by MXPOST tagger (Ratnaparkhi, 1996) and lemmatized by the *morpha* tool (Minnen, Carroll, and Pearce, 2001). The resulting list contains 910,216 triples.

Lemmatization procedure makes two attempts to find a lemma:

- first, it tries to find a triple with a matching word form and its (manually assigned) POS;
- if it fails, it makes a second attempt with the word form converted to lowercase.

If it fails in both attempts, then it chooses the given word form as the lemma. For technical reasons, a unique identifier is assigned to each token in this step. Figure 2.1 contains an example of a lemmatized sentence with marked heads.

2.6.3 Transformation of Phrase Trees into Analytical Representations

The transformation of the lemmatized Penn Treebank phrase trees with marked heads to analytical trees consists of three steps:

1. Structural transformation

   The transformation from the phrase tree to the dependency tree is defined recursively:

   - Terminal nodes of the phrase are converted to nodes of the dependency tree.
   - Constituents of a non-terminal node are converted into separate dependency trees. The root node of the dependency tree transformed from the head constituent becomes the main root. Dependency trees transformed from the left and right siblings of the head constituent are attached to the main root as the left or right children, respectively.
   - Nodes representing traces are removed and their children are reattached to the parent of the trace.
   - Handling of coordination in PDT is different from the Penn Treebank annotation style and Jason Eisner's head assigning scripts (described in (Eisner, 2001), Sect. 6.2); in the case of a phrase containing a coordinating conjunction (`CC`), we consider the rightmost `CC` as the head. The treatment of apposition is a more difficult task, since there is no explicit annotation of this phenomenon in the Penn Treebank; constituents of a noun phrase separated by commas (and not containing `CC`) are considered to be in apposition and the rightmost comma is the head.

2. Assignment of analytical functions

   The information from the phrase tree and the structure of the dependency tree are both used for analytical function assignment.

   - WSJ function tag to analytical function mapping: some function tags of a phrase tree correspond to analytic functions in an analytical tree and can be mapped to them: `SBJ` $\rightarrow$ `Sb`, `DTV` $\rightarrow$ `Obj`, `LGS` $\rightarrow$ `Obj`, `BNF` $\rightarrow$ `Obj`, `TPC` $\rightarrow$ `Obj`, `CLR` $\rightarrow$ `Obj`, `ADV` $\rightarrow$ `Adv`, `DIR` $\rightarrow$ `Adv`, `EXT` $\rightarrow$ `Adv`, `LOC` $\rightarrow$ `Adv`, `MNR` $\rightarrow$ `Adv`, `PRP` $\rightarrow$ `Adv`, `TMP` $\rightarrow$ `Adv`, `PUT` $\rightarrow$ `Adv`.
   - Assignment of analytical functions using local context: for assigning analytical functions to the remaining nodes, we use simple rules taking into account POS, the name of the constituent headed by a node in the original phrase tree. In the rules this information for the current node, its parent and grandparent can be used. For example, the rule

     $$\texttt{mPOS} = \texttt{DT} | \texttt{mAF} = \texttt{Atr}$$

     assigns the analytical function `Atr` to every determiner, the rule

     $$\texttt{mPOS} = \texttt{MD} | \texttt{pPOS} = \texttt{VB} | \texttt{mAF} = \texttt{AuxV}$$

assigns the function tag AuxV to a modal verb headed by a verb, etc. The attribute mPOS representing the POS of the node is obligatory for every rule. The rules are examined primarily in the order of the longest prefix of the POS of the given node and secondarily in the order as they are listed in the rule file. The ordering of rules is important since the first matching rule found assigns the analytical function and the search is finished.

3. PDT specific operations

   Differences between PDT and Penn Treebank annotation schemes, mainly the markup of coordinations, appositions, and prepositional phrases are handled by this step.

   - Coordinations and appositions: the analytical function, which was originally assigned to the head of coordination or apposition respectively is propagated to children nodes with the attached suffix _Co or _Ap and the head nodes get the analytical function Coord or Apos.
   - Prepositional phrases: the analytical function originally assigned to the preposition node is propagated to its child and the preposition node is labeled AuxP.
   - Sentences in the PDT annotation style always contain a root node labeled AuxS, which, as the only one in the dependency tree, does not correspond to any terminal of the phrase tree; the root node is inserted above the original root. While in the Penn Treebank, the final punctuation is a constituent of the sentence phrase, in the analytical tree, it is moved under the sentence root node.

After these rearrangements modifying the local context of some nodes, the analytical function assignment procedure attempts to label the remaining empty positions.

data category	#sentences
training	42,697
devtest	248
step devtest	3,384
evaltest	249
step evaltest	1,416

Table 2.3: Penn Treebank sentences automatically converted into Analytical and Tectogrammatical representation

2.7 English Tectogrammatical Dependency Trees

The transformation of Penn Treebank phrase trees into Tectogrammatical representation reuses the preprocessing (marking heads and lemmatization) described in Sections 2.6.1 and 2.6.2, and consists of the following three steps:

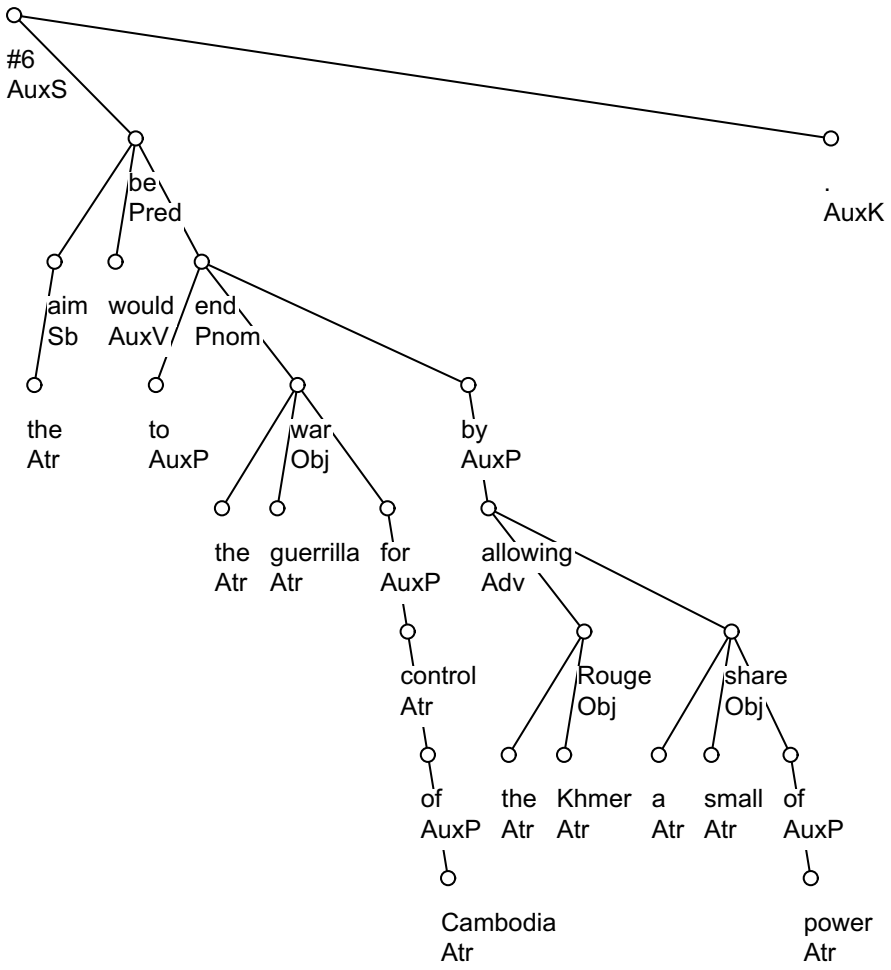


Figure 2.2: Example of an analytical tree automatically converted from Penn Treebank: "*The aim would be to end the guerrilla war for control of Cambodia by allowing the Khmer Rouge a small share of power.*" (In the noun phrase *Khmer Rouge* the word *Rouge* was marked as head by mistake.)

1. Structural Transformation - the topology of the tectogrammatical tree is derived from the topology of the PTB tree, and each node is labeled with the information from the PTB tree. In this step, the concept of head of a PTB subtree plays a key role;

2. Functor Assignment - a functor is assigned to each node of the tectogrammatical tree;

3. Grammateme Assignment - morphological (e.g. Tense, Degree of Comparison) and syntactic grammatemes (e.g. TWHEN_AFT(er)) are assigned to each node of the tectogrammatical tree. The assignment of the morphological attributes is based on PennTreebank tags and reflects basic morphological properties of the language. The syntactic grammatemes capture more specific information about deep syntactic structure. At the moment, there are no automatic tools for the assignment of the latter ones.

The whole procedure is described in detail in (Žabokrtský and Kučerová, 2002).

In order to gain a "gold standard" annotation, roughly 1,000 sentences have been annotated manually (see Table 2.4). These data are assigned morphological gramatemes (the full set of values) and syntactic grammatemes, and the nodes are reordered according to topic-focus-articulation.

data category	#sentences
training	561
devtest	248
step devtest	199
evaltest	249
step evaltest	0

Table 2.4: Penn Treebank sentences manually assigned Tectogrammatical representations

2.8 Part-of-Speech Tagging and Lemmatization of Czech

The Czech translations of Penn Treebank were automatically tokenized and morphologically tagged, each word form was assigned a basic form - *lemma* - by (Hajič and Hladká, 1998) tagging tools.

2.9 Analytical parsing of Czech

Czech analytical parsing consists of a statistical dependency parser for Czech (Hajič et al., 1998) and a module for automatic analytical functor assignment (Žabokrtský, Sgall, and Džeroski, 2002). For efficiency reasons, sentences longer than 60 words were excluded from the corpus in this step.

Figure 2.4 contains an example of a Czech analytical tree.

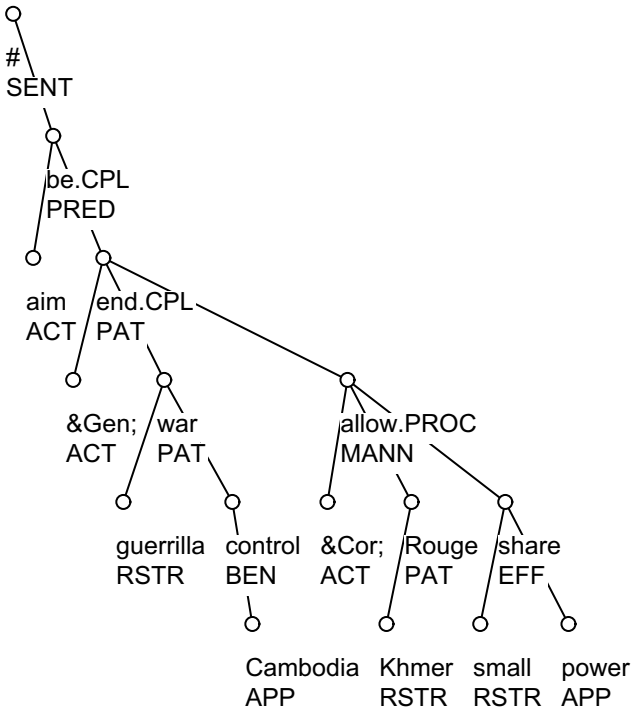


Figure 2.3: Example of a tectogrammatical tree automatically converted from Penn Treebank: "*The aim would be to end the guerrilla war for control of Cambodia by allowing the Khmer Rouge a small share of power.*" (In the noun phrase *Khmer Rouge* the word *Rouge* was marked as the head by mistake.)

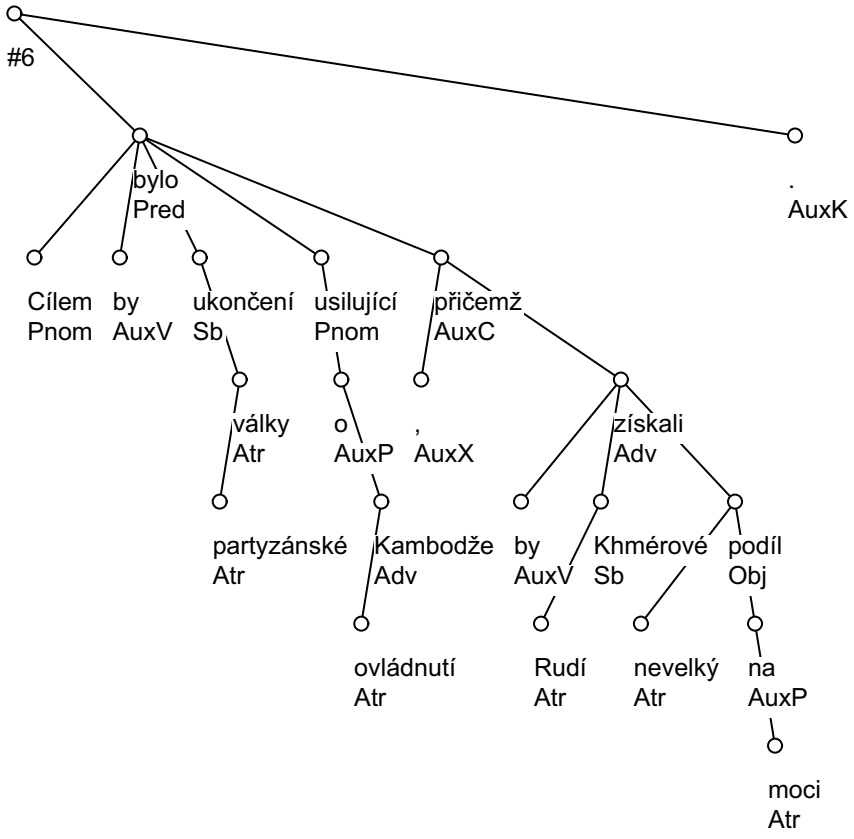


Figure 2.4: Example of a Czech analytical tree automatically parsed from input text: "*Cílem by bylo ukončení partyzánské války usilující o ovládnutí Kambodže, přičemž by Rudí Khmérové získali nevelký podíl na moci.*" (As a result of automatic parsing, this tree contains some errors in attachment and analytical function assignment: specifically, the phrase headed by "usilující" (*trying*) should have been modifying "války" (*war*), not "bylo" (*to be*, the main verb of the sentence))

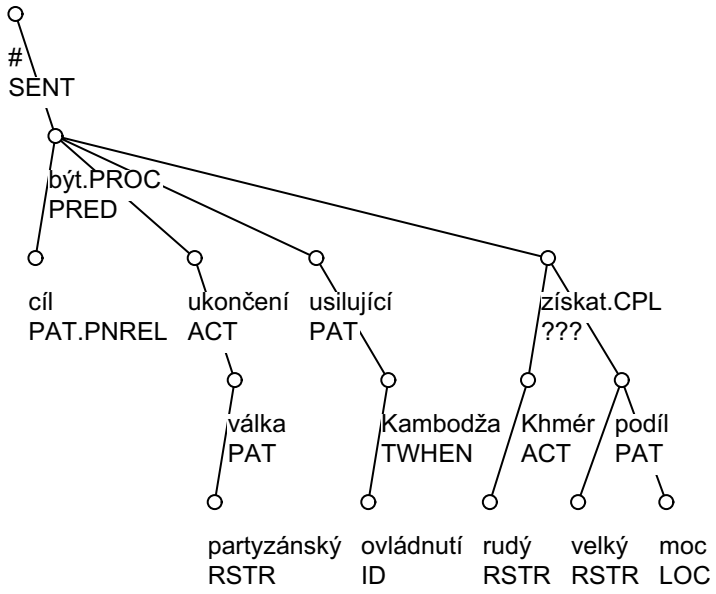


Figure 2.5: Example of a Czech tectogrammatical tree automatically converted from the analytical one: "*Cílem by bylo ukončení partyzánské války usilující o ovládnutí Kambodže, přičemž by Rudí Khmérové získali nevelký podíl na moci.*" (The incorrect structure from the analytical tree in Figure 2.4 persists.)

2.10 Tectogrammatical parsing of Czech

During the tectogrammatical parsing of Czech, the analytical tree structure is converted into the tectogrammatical one. These transformations are described by linguistic rules (Böhmová, 2001). Then, tectogrammatical functors are assigned by a C4.5 classifier (Žabokrtský, Sgall, and Džeroski, 2002).

Figure 2.5 contains an example of a Czech tectogrammatical tree.

2.11 Tectogrammatical Lexical Transfer - "Czenglish" tectogrammatical representation

In this step, tectogrammatical trees automatically created from Czech input text are transfered into so-called "Czenglish" tectogrammatical trees. The transfer procedure itself is a lexical replacement of the trlemma attribute of autosemantic nodes by its English equivalent found in the Czech-English probabilistic dictionary. Because of multiple translation possibilities, the output structure is a forest of "Czenglish" tectogrammatical trees represented in a packed-tree format (Langkilde, 2000).

2.11.1 Translation equivalent replacement algorithm (for 1-1, 1-2 entry-translation mapping).

For each Czech tectogrammatical tree (TGTree) do:

1. Start at the root
2. In the dictionary, find translation equivalents for ”trlemma” of this node
3. If there is only one translation, add the appropriate TN-tags to this node, continue with step 9

   If there is more than one translation:
4. Change the current node into OR_node
5. For each child of the current node create a new ID_node, set the parent of the child to this ID_node
6. Create new WORD_node for each translation, set parents of the new nodes to the OR_node
7. If there is a two-word translation, create a new node for the dependent word and set its parent to the appropriate WORD_node created in 6)
8. For each ID_node created in step 5 set multiple parents to all WORD_nodes created in step 6
9. Backtrack to the next node in TGTree and continue with step 2

Figure 2.6 contains an example of the “Czenglish” tectogrammatical packed-tree.

For practical reasons such as time efficiency and integration with the Tree-to-tree transducer, a simplified version, taking into account only the first most probable translation was used during the time of the workshop. Also 1-2 translations were handled as 1-1 — two words in one “trlemma” attribute.

2.12 Czech-English Word-to-Word Translation Dictionaries

2.12.1 Manual Dictionary Sources

There were three different sources of Czech-English manual dictionaries available, two of them were downloaded form the Web (WinGED, GNU/FDL), and one was extracted from the Czech/English EuroWordNet. See dictionaries parameters in Table 2.5.

2.12.2 Dictionary Filtering

For a subsequent use of these dictionaries for a simple transfer from the Czech to the English tectogrammatical trees (see Section 2.11) a relatively huge number of

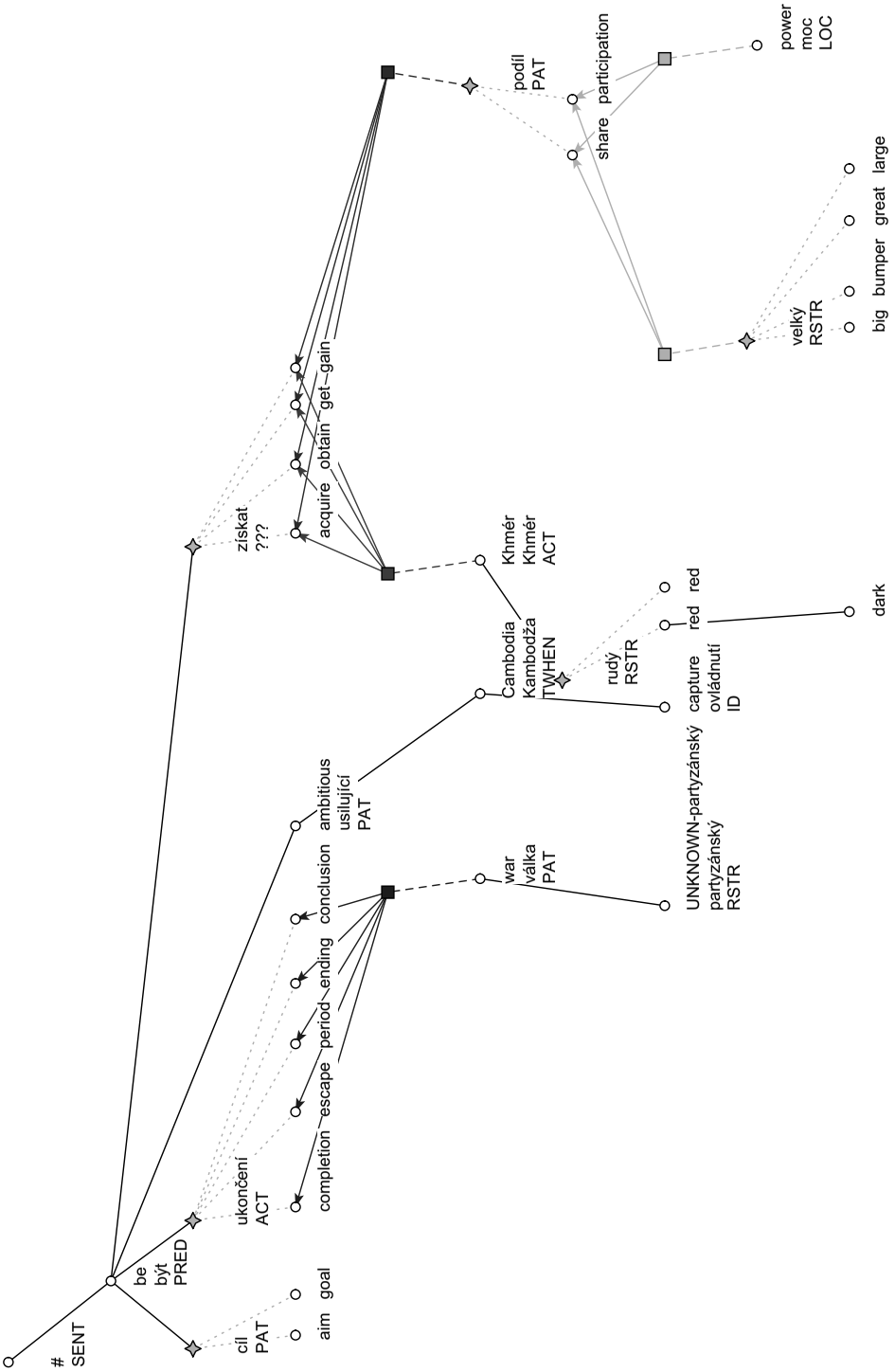


Figure 2.6: Example of a packed tree representation of a forest of Czenglish tectogrammatical trees resulting from the sentence: "*Cílem by bylo ukončení partyzánské války usilující o ovládnutí Kambodže, přičemž by Rudí Khmérové získali nevelký podíl na moci.*"

Dictionary	*# entries*	*# translations*	*Weight*
EuroWordNet	12,052	48,525	**3**
GNU/FDL	12,428	17,462	**3**
WinGED	16,296	39,769	**2**
merged	33,028	87,955	—

Table 2.5: Dictionary parameters and weights

possible translations for each entry[3] had to be filtered. The aim of the filtering is to exclude synonyms from the translation list, i.e. to choose one representative per meaning.

First, all dictionaries are converted into a unified XML format (See description of steps *a8822*, *b8822* in Table 2.6) and merged together preserving information about the source dictionary (*c8822*, *d8822*).

This merged dictionary consisting of entry/translation pairs (Czech entries and English translations in our case) is enriched by the following procedures:

- Frequencies of English word obtained from large English monolingual corpora are added to each translation (*e8822*). See description of the corpora in Section 2.6.2.
- Czech POS tag and stem are added to each entry using the Czech morphological analyzer (*f8822*, (Hajič and Hladká, 1998)).
- English POS tag is added to each translation (*g8822*). If there is more than one English POS tag obtained from the English morphological analyzer (Ratnaparkhi, 1996), the English POS tag is "disambiguated" according to the Czech POS in the appropriate entry/translation pair.

We select few relevant translations for each entry taking into account the sum of the weights of the source dictionaries (see dictionary weights in Table 2.5), the frequencies from English monolingual corpora, and the correspondence of the Czech and English POS tags (*j8822*).

2.12.3 Scoring Translations Using GIZA++

To make dictionaries more sensitive to a specific domain, which is in our case the domain of financial news, and because of the use of stochastic methods in the subsequent stages (such as transduction of English TGTrees to ALTrees), it would help to have the translations somehow weighted.

By extending this dictionary by the training part of the Czech-English parallel corpus (7,412 sentences from WSJ) and by running GIZA++ training (translation models 1-4, see (Och and Ney, 2000) on it (*steps a8824–e8824*), we obtained a probabilistic Czech-English dictionary. As a result, the entry/translation pairs seen in the parallel corpus become more probable. For entry/translation pairs not seen in the parallel text, the probability distribution among translations is uniform. The

[3]For example for WinGED dictionary it is 2.44 translations per entry in average, and excluding 1-1 entry/translation pairs even 4.51 translations/entry.

translation is “GIZA++ selected” if its probability is higher than a threshold, which is set to 0.10 in our case.

The final selection (*l8822*) contains translations selected by both the dictionary and GIZA++ selectors. In addition, translations not covered by the original dictionary can be included in the final selection, if they were newly discovered in the parallel corpus by GIZA++ training and their probability is significant (higher than the most probable translation so far).

The translations of the final selection are used in the transfer (*steps h8801 or i8801*). See the sample of the dictionary in Figure 2.7.

```
<e>zesílit<t>V

   [FSG]<tr>increase<trt>V<prob>0.327524
   [FSG]<tr>reinforce<trt>V<prob>0.280199
   [FSG]<tr>amplify<trt>V<prob>0.280198
   [G]<tr>re-enforce<trt>V<prob>0.0560397
   [G]<tr>reenforce<trt>V<prob>0.0560397

<e>výběr<t>N

   [FSG]<tr>choice<trt>N<prob>0.404815
   [FSG]<tr>selection<trt>N<prob>0.328721
   [G]<tr>option<trt>N<prob>0.0579416
   [G]<tr>digest<trt>N<prob>0.0547869
   [G]<tr>compilation<trt>N<prob>0.0547869
   []<tr>alternative<trt>N<prob>0.0519888
   []<tr>sample<trt>N<prob>0.0469601

<e>selekce<t>N

   [FSG]<tr>selection<trt>N<prob>0.542169
   [FSG]<tr>choice<trt>N<prob>0.457831

<e>rozšířit<t>V

   [FSG]<tr>widen<trt>V<prob>0.20402
   [FSG]<tr>enlarge<trt>V<prob>0.20402
   [G]<tr>expand<trt>V<prob>0.138949
   [G]<tr>extend<trt>V<prob>0.130029
   [G]<tr>spread<trt>V<prob>0.0822508
   []<tr>step<trt>V<prob>0.0516784
   []<tr>let<trt>X<prob>0.0459122
   []<tr>stretch<trt>V<prob>0.0427784
   []<tr>larger<trt>V<prob>0.040804
   []<tr>broaden<trt>V<prob>0.040804
   []<tr>ground-handling<trt>N<prob>0.0136013
   []<tr>make larger<trt>V<prob>0.01
   []<tr>let_out<trt>V<prob>0.01
   []<tr>reconsider<trt>V<prob>0.00515253

     [S] ... dictionary weight selection
     [G] ... GIZA++ selection
     [F] ... final selection
```

Figure 2.7: Sample of the Czech-English dictionary used for the tranfer.

step	functionality summary
	8801 – Czech data
a8801	tokenization of Czech WSJ files
b8801	morphology & tagging
c8801	preprocessing necessary for Collins' parser
d8801	statistical dependency parser for Czech
e8801	analytical function assignment
f8801	rule based conversion of analytical representation into tectogrammatical representation
g8801	C 4.5 based assignment of tectogrammatical functors
h8801	lexical transfer into "Czenglish" packed forest representation
i8801	simplified lexical transfer into "Czenglish", first translation
	8802 – English data
a8802	marking heads in Penn Treebank trees
b8802	lemmatization of Penn Treebank
c8802	conversion of Penn Treebank trees into analytical trees
d8802	conversion of Penn Treebank trees into tectogrammatical trees
	8822 – Czech-English Dictionary Filtering
a8822	creating unified SGML format of dictionaries from various input formats
b8822	filtering out garbage
c8822	conversion into XML
d8822	preparing dictionary for POS annotation
e8822	adding frequencies from large monolingual corpus to English translations
f8822	morphological analysis of Czech entries
g8822	morphological analysis of English translations
h8822	merging temporary dictionaries from steps e8822, f8822 and g8822 into one XML dictionary
i8822	converting the whole dictionary (without any filtering criteria) to a parallel plain text corpus to be used as GIZA++ training data
j8822	selecting translations according to dictionary weights and converting the selected sub-dictionary to a parallel plain text corpus to be used as GIZA++ training data
k8822	merges results of GIZA++ dictionary training (e8824) with XML dictionary.
l8822	selecting translations for transfer according to dictionary weights and GIZA++ translation probabilities
m8822	stores Czech-English translation dictionary to be used for transfer (h8801, i8801)
	8824 – Czech-English Probabilistic Dictionary Training
a8824	creating parallel corpus from Czech tectogrammatical trees (g8801) and English tectogrammatical trees (d8802)
b8824	creating plain text parallel corpus of trlemmas for GIZA++ training
c8824	extending training corpus by corpus obtained from i8822 or j8822
d8824	GIZA++ training, model 4
e8824	converting GIZA++ output into XML

Table 2.6: Summary of used scripts

Chapter 3

The Generation System MAGENTA

Jason Eisner, Jan Hajič, Dan Gildea,
Yuan Ding, Terry Koo, Kristen Parton

The Generation system assumes that we have a tectogrammatical (TR) tree (as a result of source language analysis and transfer), and its task is to transform this tree to the correct target-language sentence by converting it first to an intermediate representation, namely, a surface-syntactic dependency tree, called an analytical (AR) tree.

In the project, we have divided the task into four successive steps:

1. Global Tree Preprocesing/transformation (additional target-language labeling) - section 3.3;
2. Tree-to-tree transducer (TR to AR, unordered) - section 3.1 and section 3.2;
3. Language Model (ordering the AR tree nodes) - section 3.4;
4. English Morphology and Punctuation - section 3.5.

The first step (additional supportive tree labeling using global tree information) has only been implemented separately (aiming at correct preposition generation) and it is not part of the system as evaluated. However, it is meant to be an example of a module that can help to better guide the core of the system, namely the tree transformation step.

3.1 A Generative Model for Pairs of Trees

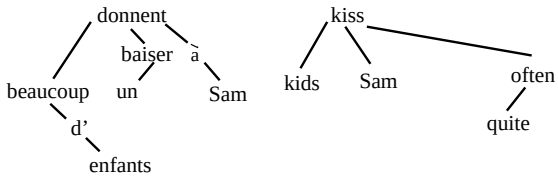
In this section, we present a description of the core of the generation system, namely a generative model for pairs of trees that enables (locally) non-isomorphic "transformations" of tree structures. It is usable not only for NL generation, but also for machine translation, deep parsing etc.[1]

[1]This section also appeared separately as (Eisner, 2003). It was also supported by ONR grant N00014-01-1-0685, "Improving Statistical Models Via Text Analyzers Trained from Parallel Corpora." The views expressed are the author's.

3.1.1 Tree-to-Tree Mappings

Statistical machine translation systems are trained on pairs of sentences that are mutual translations. For example, (*beaucoup d'enfants donnent un baiser à Sam, kids kiss Sam quite often*). This translation is somewhat free, as is common in naturally occurring data. The first sentence is literally *Lots of'children give a kiss to Sam.*

In the present section 3.1.1, we outline "natural" formalisms and algorithms for training on pairs of *trees*. Our methods work on either dependency trees (as shown) or phrase-structure trees. Note that the depicted trees are not isomorphic.



Our main concern is to develop models that can align and learn from these tree pairs despite the "mismatches" in tree structure. Many "mismatches" are characteristic of a language pair: e.g., preposition insertion (*of* $\rightarrow \epsilon$), multiword locutions (*kiss* $\leftrightarrow$ *give a kiss to; misinform* $\leftrightarrow$ *wrongly inform*), and head-swapping (*float down* $\leftrightarrow$ *descend by floating*). Such systematic mismatches should be learned by the model, and used during translation.

It is even helpful to learn mismatches that merely tend to arise during free translation. Knowing that *beaucoup d'* is often deleted will help in aligning the rest of the tree.

When would learned tree-to-tree mappings be useful? Obviously, in MT, when one has parsers for both the source and target language. Systems for "deep" analysis and generation might wish to learn mappings between deep and surface trees (Böhmová et al., 2001) or between syntax and semantics (Shieber and Schabes, 1990). Systems for summarization or paraphrase could also be trained on tree pairs (Knight and Marcu, 2000). Non-NLP applications might include comparing student-written programs to one another or to the correct solution.

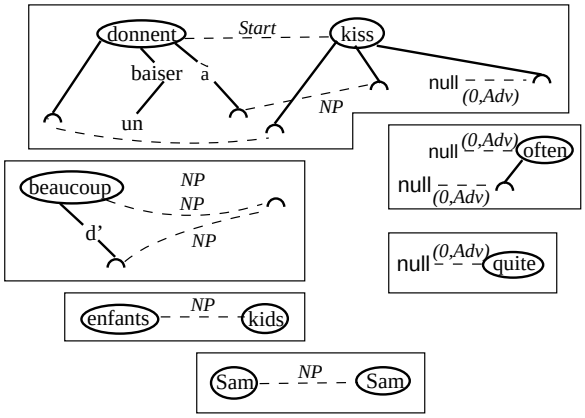
Our methods can naturally extend to train on pairs of *forests* (including packed forests obtained by chart parsing). The correct tree is presumed to be an element of the forest. This makes it possible to train even when the correct parse is not fully known, or not known at all.

3.1.2 A Natural Proposal: Synchronous TSG

We make the quite natural proposal of using a synchronous tree substitution grammar (STSG). An STSG is a collection of (ordered) pairs of aligned **elementary trees**. These may be combined into a **derived** pair of trees. Both the elementary tree pairs and the operation to combine them will be formalized in later sections.

As an example, the tree pair shown in the introduction might have been derived by "vertically" assembling the 6 elementary tree pairs below. The $\frown$ symbol denotes a **frontier node** of an elementary tree, which must be replaced by the circled **root** of another elementary tree. If two frontier nodes are linked by a dashed line

labeled with the **state** X, then they must be replaced by two roots that are also linked by a dashed line labeled with X.



The elementary trees represent idiomatic translation "chunks." The frontier nodes represent unfilled roles in the chunks, and the states are effectively nonterminals that specify the type of filler that is required. Thus, *donnent un baiser à* ("give a kiss to") corresponds to *kiss*, with the French subject matched to the English subject, and the French indirect object matched to the English direct object. The states could be more refined than those shown above: the state for the subject, for example, should probably be not *NP* but a pair (N_{pl}, NP_{3s}).

STSG is simply a version of synchronous tree-adjoining grammar or STAG (Shieber and Schabes, 1990) that lacks the adjunction operation. (It is also equivalent to top-down tree transducers.) What, then, is new here?

First, we know of no previous attempt to *learn* the "chunk-to-chunk" mappings. That is, we do not *know* at training time how the tree pair of section 3.1.1 was derived, or even what it was derived from. Our approach is to reconstruct *all possible derivations*, using dynamic programming to decompose the tree pair into aligned pairs of elementary trees in all possible ways. This produces a packed forest of derivations, some more probable than others. We use an efficient inside-outside algorithm to do Expectation-Maximization, reestimating the model by training on all derivations in proportion to their probabilities. The runtime is quite low when the training trees are fully specified and elementary trees are bounded in size.[2]

Second, it is not *a priori* obvious that one can reasonably use STSG instead of the slower but more powerful STAG. TSG can be parsed as fast as CFG. But without an adjunction operation,[3], one cannot break the training trees into linguistically minimal units. An elementary tree pair A = *(elle est finalement partie, finally she left)* cannot be further decomposed into B = *(elle est partie, she left)* and C = *(finalement, finally)*. This appears to miss a generalization. Our perspective is that the generalization should be picked up by the statistical model that defines the probability of elementary tree pairs. $p(A)$ can be defined using mainly the same

[2](Goodman, 2002) presents efficient TSG parsing with unbounded elementary trees. Alas, that clever DOP-to-CFG reduction does not appear to generalize to our synchronous case. (It would need exponentially many nonterminals to keep track of a matching of unboundedly many frontier nodes.) Nor does it permit arbitrary models of elementary-tree probabilities.

[3]Or a sister-adjunction operation, for dependency trees.

parameters that define $p(B)$ and $p(C)$, with the result that $p(A) \approx p(B) \cdot p(C)$. The balance between the STSG and the statistical model is summarized in the last paragraph of this section.

Third, our version of the STSG formalism is more flexible than previous versions. We carefully address the case of empty trees, which are needed to handle free-translation "mismatches." In the example, an STSG cannot replace *beaucoup d'* ("lots of") in the NP by *quite often* in the VP; instead it must delete the former and insert the latter. Thus we have the alignments *(beaucoup d', ϵ)* and *(ϵ, quite often)*. These require innovations as shown. The **tree-internal** deletion of *beaucoup d'* is handled by an empty elementary tree in which the root is itself a frontier node. (The subject frontier node of *kiss* is replaced with this frontier node, which is then replaced with *kids*.) The **tree-peripheral** insertion of *quite often* requires an English frontier node that is paired with a French null.

We also formulate STSGs flexibly enough that they can handle both phrase-structure trees and dependency trees. The latter are small and simple (Alshawi, Bangalore, and Douglas, 2000): tree nodes are words, and there need be no other structure to recover or align. Selectional preferences and other interactions can be accommodated by enriching the states.

Any STSG has a weakly equivalent SCFG that generates the same string pairs. So STSG (unlike STAG) has no real advantage for modeling *string* pairs.[4] But STSGs can generate a wider variety of *tree* pairs, e.g., non-isomorphic ones. So when actual trees are provided for training, STSG can be more flexible in aligning them.

3.1.3 Past Work

Most statistical MT derives from IBM-style models (Brown et al., 1993), which ignore syntax and allow arbitrary word-to-word translation. Hence they are able to align any sentence pair, however mismatched. However, they have a tendency to translate long sentences into word salad. Their alignment and translation accuracy improves when they are forced to translate shallow phrases as contiguous, potentially idiomatic units (Och, Tillmann, and Ney, 1999).

Several researchers have tried putting "more syntax" into translation models: like us, they use statistical versions of synchronous grammars, which generate source and target sentences in parallel and so describe their correspondence.[5] This approach offers four features absent from IBM-style models: (1) a recursive phrase-based translation, (2) a syntax-based language model, (3) the ability to condition a word's translation on the translation of syntactically related words, and (4) polynomial-time optimal alignment and decoding (Knight, 1999).

Previous work in statistical synchronous grammars has been limited to forms of synchronous context-free grammar (Wu, 1997; Alshawi, Bangalore, and Douglas, 2000; Yamada and Knight, 2001). This means that a sentence and its translation must have isomorphic syntax trees, although they may have different numbers of

[4] However, the *binary-branching* SCFGs used by (Wu, 1997) and (Alshawi, Bangalore, and Douglas, 2000) are strictly less powerful than STSG.

[5] The joint probability model can be formulated, if desired, as a channel model times a much better-trained language model.

surface words if null words ϵ are allowed in one or both languages. This rigidity does not fully describe real data.

The one exception is the synchronous DOP approach of (Poutsma, 2000), which obtains an STSG by decomposing *aligned* training trees in all possible ways (and using "naive" count-based probability estimates). However, we would like to estimate a model from unaligned data.

3.1.4 A Probabilistic TSG Formalism

For expository reasons (and to fill a gap in the literature), first we formally present *non*-synchronous TSG. Let Q be a set of **states**. Let L be a set of **labels** that may decorate nodes or edges. Node labels might be words or nonterminals. Edge labels might include grammatical roles such as Subject. In many trees, each node's children have an order, recorded in labels on the node's outgoing edges.

An **elementary tree** is a a tuple $\langle V, V^i, E, \ell, q, s\rangle$ where V is a set of **nodes**; $V^i \subseteq V$ is the set of **internal nodes**, and we write $V^f = V - V^i$ for the set of **frontier nodes**; $E \subseteq V^i \times V$ is a set of **directed edges** (thus all frontier nodes are leaves). The graph $\langle V, E\rangle$ must be connected and acyclic, and there must be exactly one node $r \in V$ (the **root**) that has no incoming edges. The function $\ell : (V^i \cup E) \rightarrow L$ labels each internal node or edge; $q \in Q$ is the **root state**, and $s : V^f \rightarrow Q$ assigns a **frontier state** to each frontier node (perhaps including r).

A TSG is a set of elementary trees. The generation process builds up a **derived tree** T that has the same form as an elementary tree, and for which $V^f = \emptyset$. Initially, T is chosen to be any elementary tree whose root state $T.q = \mathsf{Start}$. As long as T has any frontier nodes, $T.V^f$, the process expands each frontier node $d \in T.V^f$ by **substituting** at d an elementary tree t whose root state, $t.q$, equals d's frontier state, $T.s(d)$. This operation replaces T with $\langle T.V \cup t.V - \{d\}, T.V^i \cup t.V^i, T.E' \cup t.E, T.\ell \cup t.\ell, T.q, T.s \cup t.s - \{\langle d, t.q\rangle\}\rangle$. Note that a function is regarded here as a set of $\langle$input, output$\rangle$ pairs. $T.E'$ is a version of $T.E$ in which d has been been replaced by $t.r$.

A **probabilistic TSG** also includes a function $p(t \mid q)$, which, for each state q, gives a conditional probability distribution over the elementary trees t with root state q. The generation process uses this distribution to randomly choose which tree t to substitute at a frontier node of T having state q. The initial value of T is chosen from $p(t \mid \mathsf{Start})$. Thus, the probability of a given derivation is a product of $p(t \mid q)$ terms, one per chosen elementary tree.

There is a natural analogy between (probabilistic) TSGs and (probabilistic) CFGs. An elementary tree t with root state q and frontier states $q_1 \ldots q_k$ (for $k \geq 0$) is analogous to a CFG rule $q \rightarrow t\ q_1 \ldots q_k$. (By including t as a terminal symbol in this rule, we ensure that distinct elementary trees t with the same states correspond to distinct rules.) Indeed, an equivalent definition of the generation process first generates a **derivation tree** from this **derivation CFG**, and then combines its terminal nodes t (which are elementary trees) into the derived tree T.

3.1.5 Tree Parsing Algorithms for TSG

Given a a grammar G and a derived tree T, we may be interested in constructing the forest of T's possible derivation trees (as defined above). We call this **tree parsing**, as it finds ways of decomposing T into elementary trees.

Given a node $c \in T.v$, we would like to find all the potential elementary subtrees t of T whose root $t.r$ could have contributed c during the derivation of T. Such an elementary tree is said to **fit** c, in the sense that it is isomorphic to some subgraph of T rooted at c.

The following procedure finds an elementary tree t that fits c. Freely choose a connected subgraph U of T such that U is rooted at c (or is empty). Let $t.V^i$ be the vertex set of U. Let $t.E$ be the set of outgoing edges from nodes in $t.V^i$ to their children, that is, $t.E = T.E \cap (t.V^i \times T.V)$. Let $t.\ell$ be the restriction of $T.\ell$ to $t.V^i \cup t.E$, that is, $t.\ell = T.\ell \cap ((t.V^i \cup t.E) \times L)$. Let $t.V$ be the set of nodes mentioned in $t.E$, or put $t.V = \{c\}$ if $t.V^i = t.E = \emptyset$. Finally, choose $t.q$ freely from Q, and choose $s : t.V^f \to Q$ to associate states with the frontier nodes of t; the free choice is because the nodes of the derived tree T do not specify the states used during the derivation.

How many elementary trees can we find that fit c? Let us impose an upper bound k on $|t.V^i|$ and hence on $|U|$. Then in an m-ary tree T, the above procedure considers at most $\frac{m^k-1}{m-1}$ connected subgraphs U of order $\leq k$ rooted at c. For dependency grammars, limiting to $m \leq 6$ and $k = 3$ is quite reasonable, leaving at most 43 subgraphs U rooted at each node c, of which the biggest contain only c, a child c' of c, and a child or sibling of c'. These will constitute the internal nodes of t, and their remaining children will be t's frontier nodes.

However, for each of these 43 subgraphs, we must jointly hypothesize states for all frontier nodes and the root node. For $|Q| > 1$, there are exponentially many ways to do this. To avoid having exponentially many hypotheses, one may restrict the form of possible elementary trees so that the possible states of each node of t can be determined somehow from the labels on the corresponding nodes in T. As a simple but useful example, a node labeled *NP* might be required to have state *NP*. Rich labels on the derived tree essentially provide supervision as to what the states must have been during the derivation.

The tree parsing algorithm resembles bottom-up chart parsing under the derivation CFG. But the input is a tree rather than a string, and the chart is indexed by nodes of the input tree rather than spans of the input string.[6]

1. **for** each node c of T, in bottom-up order
2. **for** each $q \in Q$, **let** $\beta_c(q) = 0$
3. **for** each elementary tree t that fits c
4. increment $\beta_c(t.q)$ by $p(t \mid t.q) \cdot \prod_{d \in t.V^f} \beta_d(t.s(d))$

The β values are inside probabilities. After running the algorithm, if r is the root of T, then $\beta_r(\mathsf{Start})$ is the probability that the grammar generates T.

$p(t \mid q)$ in line 4 may be found by hash lookup if the grammar is stored explicitly, or else by some probabilistic model that analyzes the structure, labels, and states of the elementary tree t to compute its probability.

One can mechanically transform this algorithm to compute outside probabilities, the Viterbi parse, the parse forest, and other quantities (Goodman, 1999). One can also apply agenda-based parsing strategies.

For a fixed grammar, the runtime and space are only $O(n)$ for a tree of n nodes.

[6] We gloss over the standard difficulty that the derivation CFG may contain a unary rule cycle. For us, such a cycle is a problem only when it arises solely from single-node trees.

The grammar constant is the number of possible fits to a node c of a fixed tree. As noted above, there usually not many of these (unless the states are uncertain) and they are simple to enumerate.

As discussed above, an inside-outside algorithm may be used to compute the expected number of times each elementary tree t appeared in the derivation of T. That is the E step of the EM algorithm. In the M step, these expected counts (collected over a corpus of trees) are used to reestimate the parameters $\vec{\theta}$ of $p(t \mid q)$. One alternates E and M steps till $p(\text{corpus} \mid \vec{\theta}) \cdot p(\vec{\theta})$ converges to a local maximum. The prior $p(\vec{\theta})$ can discourage overfitting.

3.1.6 Extending to Synchronous TSG

We are now prepared to discuss the synchronous case. A synchronous TSG consists of a set of **elementary tree pairs**. An elementary tree pair t is a tuple $\langle t_1, t_2, q, m, s \rangle$. Here t_1 and t_2 are elementary trees without their own root or frontier states (that is, $t_j = \langle V_j, V_j^i, E_j, \ell_j \rangle$). But $q \in Q$ is a single root state for the *pair*. $m \subseteq V_1^f \times V_2^f$ is a matching between t_1's and t_2's frontier nodes.[7] Let $\bar{m} = m \cup \{(d_1, \textsf{null}) : d_1 \text{ is unmatched in } m\} \cup \{(\textsf{null}, d_2) : d_2 \text{ is unmatched in } m\}$. Finally, $s : \bar{m} \to Q$ assigns a state to each frontier node pair or unpaired frontier node.

In the figure of section 3.1.2, *donnent un baiser à* has 2 frontier nodes and *kiss* has 3, yielding 13 possible matchings. Note that least one English node must remain unmatched; it still generates a full subtree, aligned with null.

As before, a derived tree pair T has the same form as an elementary tree pair. The generation process is similar to before. As long as $T.\bar{m} \neq \emptyset$, the process expands some node pair $(d_1, d_2) \in T.\bar{m}$. It chooses an elementary tree pair t such that $t.q = T.s(d_1, d_2)$. Then for each $j = 1, 2$, it substitutes t_j at d_j if non-null. (If d_j is null, then $t.q$ must guarantee that t_j is the special null tree.)

In the probabilistic case, we have a distribution $p(t \mid q)$ just as before, but this time t is an elementary tree *pair*.

Several natural algorithms are now available to us:

- **Training.** Given an unaligned tree pair (T_1, T_2), we can again find the forest of all possible derivations, with expected inside-outside counts of the elementary tree pairs. This allows EM training of the $p(t \mid q)$ model.

  The algorithm is almost as before. The outer loop iterates bottom-up over nodes c_1 of T_1; an inner loop iterates bottom-up over c_2 of T_2. Inside probabilities (for example) now have the form $\beta_{c_1,c_2}(q)$. Although this brings the complexity up to $O(n^2)$, the real complication is that there can be many fits to (c_1, c_2). There are still not too many elementary trees t_1 and t_2 rooted at c_1 and c_2; but each (t_1, t_2) pair may be used in many elementary tree pairs t, since there are exponentially many matchings of their frontier nodes. Fortunately, most pairs of frontier nodes have low β values that indicate that their subtrees cannot be aligned well; pairing such nodes in a matching would result in poor global probability. This observation can be used to prune the space of matchings greatly.

[7] A **matching** between A and B is a 1-to-1 correspondence between a subset of A and a subset of B (perhaps all of A, B).

- **1-best Alignment** (if desired). The training algorithm above finds the joint probability $p(T_1, T_2)$, summed over all alignments. Use its Viterbi variant to find the single *best* derivation of the input tree pair. This derivation can be regarded as the optimal syntactic alignment.[8]

- **Decoding.** We create a forest of possible synchronous derivations (cf. (Langkilde, 2000)). We chart-parse T_1 much as in section 3.1.5, but fitting the left side of an elementary tree *pair* to each node. Roughly speaking:

1. **for** c_1 = null and then $c_1 \in T_1.V$, in bottom-up order
2. **for** each $q \in Q$, **let** $\beta_{c_1}(q) = -\infty$
3. **for** each probable $t = (t_1, t_2, q, m, s)$ whose t_1 fits c_1
4. max $p(t \mid q) \cdot \prod_{(d_1,d_2)\in\bar{m}} \beta_{d_1}(s(d_1, d_2))$ into $\beta_{c_1}(q)$

We then extract the max-probability synchronous derivation and return the T_2 that it derives. This algorithm is essentially alignment to an *unknown* tree T_2; we do not loop over its nodes c_2, but choose t_2 freely.

In line 3 of the above pseudocode, for each possible elementary tree t_1 that fits c_1, we must call a **proposer** module to find all reasonably probable ways to extend it to an elementary tree pair t. This amounts to finding possible translations of the local structure t_1. The proposer is further discussed in section **??** below. It need not score the translations, only propose them, since each proposed t will be scored by the model (the call to $p(t \mid q)$ at line 4).

3.1.7 Status of the Implementation

We have outlined an EM algorithm to learn the probabilities of elementary tree pairs by training on pairs of full trees, and a Viterbi decoder to find optimal translations. It implements Step 2 of the generation system (see the start of Chapter 3).

For the within-language translations, it sufficed to use a simplistic, fixed model of $p(t \mid q)$ that relied entirely on morpheme identity.

Team members are now developing real, trainable models of $p(t \mid q)$, such as log-linear models on meaningful features of the tree pair t. Cross-language translation results await the plugging-in of these interesting models. The algorithms we have presented serve only to "shrink" the modeling, training and decoding problems from full trees to bounded, but still complex, elementary trees.

Our elementary trees are dependency trees whose internal nodes are decorated with words and whose edges are decorated with grammatical functions. Thus, pairing elementary trees with 2 and 3 internal nodes represents a 2-word-to-3-word phrasal translation. The trees' frontier nodes control what happens to the phrase's dependents during translation.

We have tested the code by successfully aligning English TR (deep structure) to English AR (surface structure). For this we used only a simple "stub" probability model $p(t \mid q)$ that exploits lexical identity, but is not allowed to use subconstituent order information (as that would make the problem too easy). We have also demonstrated decoding with this simple "stub" model.

To use the system in real generation (English TR $\rightarrow$ English AR), we await a real, trainable model of $p(t \mid q)$ for this step. Given such a model, one would be

[8] As free-translation post-processing, one could try to pair up "moved" subtrees that align well with each other, according to the chart, but were both paired with null in the global alignment.

able to use our TR-to-AR alignments to retrain the model parameters, and repeat just as for any EM algorithm. We are interested in log-linear models of $p(t \mid q)$ for generation and also for the other steps of translation.

3.2 The Proposer

The proposer's job in a tree-to-tree translation system was discussed at the end of section 3.1.6. In our case of generation (TR-to-AR translation), the proposer must be able to suggest possible translations of each TR fragment.

In principle, a proposer could be derived directly from the probability model $p(t \mid q)$; indeed, its proposals might change at successive iterations of EM as that model's parameters are changed.

However, it is also possible to use a heuristic proposer based on table lookup and other tricks. That is our current strategy: we attempt to look up known elementary trees, and handle unknown ones (badly) by translating word-by-word. Better heuristics for translating unknown elementary trees could be imagined. Note that the proposer is allowed to overgenerate (at the cost of speed), as poor proposals will simply be scored low by the probability model.

The task of the proposer for the decoder is the following:

- Collecting Feature Patterns on TR
- Construct AR using observed possible TR-AR transform
- For unobserved TR, using naive mapping on AR

At the training time, the proposer observes the TR to AR mappings by running through a treewalker. A typical mapping of TR structure to AR structure is shown in Fig. 3.1.

After a set of such mappings have been collected, two sets of parameters of each of the mappings are extracted. Those parameters are used to fill two databases: The TR-AR transform database and the TR feature database. The TR-AR transform database stores each of the mappings, including the lexical transfer and the structural variance. The TR feature database stores the features of the TR that each mapping is conditioned on. The scheme of the proposer at the training stage is shown in Fig. 3.2.

At the decoding stage, the proposer first take an input TR tree, and fills it into the TR side of the TR-AR pair. Then the proposer looks into each of the observed feature combinations in the TR feature database, fills the features, and make a query into the TR-AR transform database. If the query returns a set of non-empty value(s), that set of AR trees is proposed to the decoder. Otherwise, the AR tree is constructed using a naive rule by simply replacing the lemmas. The scheme of the proposer at the decoding stage is shown in Fig. 3.3, and the Fig. 3.4 shows how a query is made by the proposer.

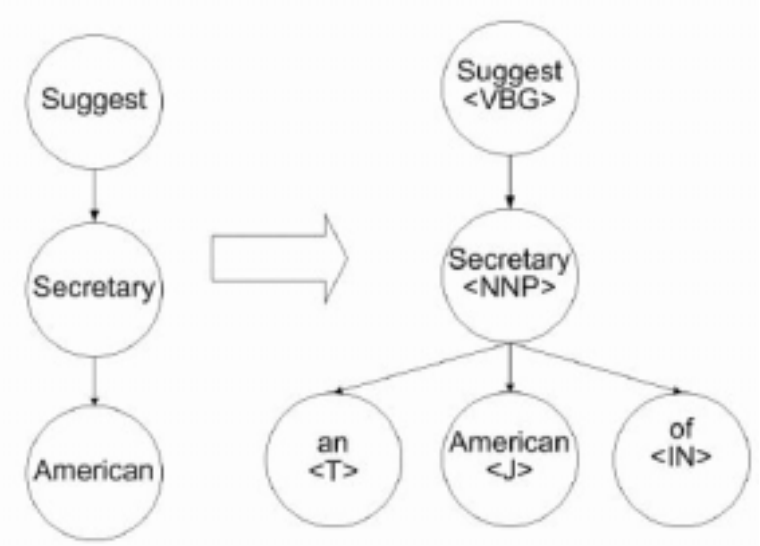


Figure 3.1: TR to AR mapping

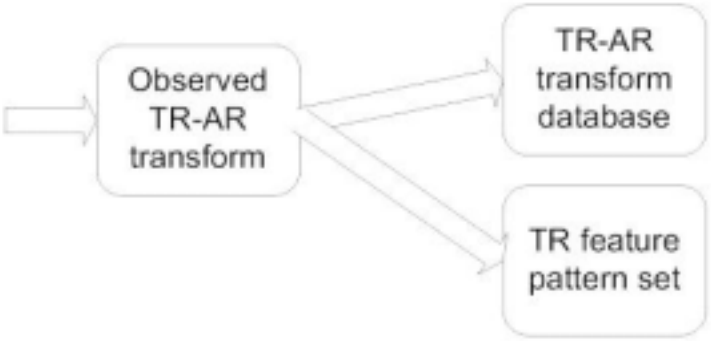


Figure 3.2: Proposer: training phase

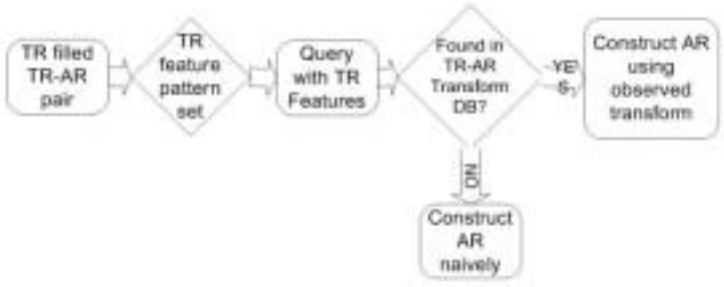


Figure 3.3: Proposer: decoding stage

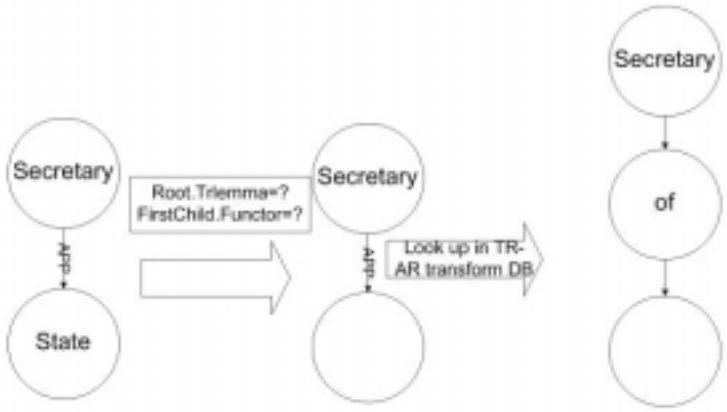


Figure 3.4: Query made by the Proposer

3.3 Using Global Tree Information: Preposition Insertion

exist in the AR. This section describes our implementation of a classifier that indicates where prepositions must be inserted into the TR. Because the system operates on the tree as a whole, it has access to global tree information and can make more accurate decisions than the Proposer. Although we only discuss preposition insertion in this section, the techniques we present are applicable to other tree transformations such as insertion of determiners or auxiliary verbs. All such modules in conjunction, if integrated with the generation system, implement its Step 1 (see the start of Chapter 3).

The remainder of this section is split into four subsections. The first two subsections present, respectively, an overview of the C5.0 data mining tool, which we used to train our classifier, and the general technique of using classifiers to capture global tree information. The next subsection presents our preposition insertion implementation, describing the operation of our classifier, how we prepared its testing and training data, how we improved its performance, and how it performs. The final subsection concludes with suggestions for future work.

3.3.1 Description of C5.0

C5.0 is a data mining tool that generates classifiers in the form of decision trees or rule sets. We present a brief overview of the C5.0 data mining tool. The reader already familiar with C5.0 may skip this sub-section.

The training data for C5.0 is set of *cases*, where each case consists of a *context* and a *classification*. The context can be made of any number of variables which can be real numbers, dates, enumerated discrete values, or functions on the other variables (such as `area := width*height` or `young := age < 20`). The classification is the category to which C5.0 should assign this case. C5.0 has a number of

training options, a few of which we describe below.

Discrete Value Sub-setting C5.0 can make decisions based on subsets of discrete values instead of individual discrete values. On a data set with many discrete values, discrete value sub-setting reduces fragmentation of the training data.

Adaptive Boosting C5.0 can run several trials of classifier generation on given sets of training and test data. The first classifier generated may do well but will probably make errors on some regions of the test data. C5.0 takes these errors into account and trains the second classifier with an emphasis on these error-prone data. This process continues iteratively until some specified number of trials is complete or the accuracy of the classifier becomes either very high or very low.

Misclassification Costs C5.0 allows misclassification costs to be specified individually. This is useful in when one kind of error is more grave than another; for instance, when diagnosing a potentially fatal disease, we would rather misdiagnose a healthy patient as sick than vice versa. By specifying a high cost for misclassifying `sick` as `healthy` and a low cost for misclassifying `healthy` as `sick`, we can instill a bias in the classifier C5.0 produces.

When using a C5.0 classifier, the context is presented as input, and the classifier attempts to deduce the correct classification. Source code is available for a free C program called `sample`, which takes a trained C5.0 decision tree or rule set and classifies new cases with it.

3.3.2 Using Classifiers to Capture Tree Information

We now discuss the general technique of using classifiers to aid our tree transduction. Recall that our tree transduction operates by transforming small tree fragments at a time and then composing those fragments into whole trees. Unfortunately, the TR tree fragments only provide local information limited by the size of the fragment. This lack of information makes it difficult for the Proposer to choose the correct AR transformation.

In actuality, the entire TR tree is available to the transducer, but we choose not to transduce the entire tree at one go for data sparseness and computational efficiency reasons. However, some TR to AR transformations depend on long-range relationships or other data which are lost when the tree is broken into small fragments. We would like to capture the relevant long-range information in some local form that the Proposer can easily access.

Our approach is to preprocess the TR tree with a classifier, attaching arbitrary informative labels to individual nodes. When the Proposer receives a tree fragment, it reads the labels off of the nodes in the fragment, incorporating the labels' information in its decisions.

3.3.3 The Preposition Insertion Classifier

The classifier operates by labeling individual nodes of the TR tree. The possible labels are `nothing`, indicating that no preposition should be inserted above this node, or `insert_X`, where `X` represents a preposition to insert above the labeled node. `X` can either be a single preposition such as "of" or a compound preposition such as "because_of". In the case of a compound preposition such as "because_of", "of" should be inserted as the parent to the labeled node and "because" should be inserted as the parent of "of".

Unfortunately, we were unable to integrate the preposition insertion classifier into the tree transduction system, due to time constraints. However, the classifier itself is fully implemented and has been tested as an independent module.

The remainder of this subsection is broken into three parts which describe, respectively, how we created the training data for our classifier, how we attempted to improve its performance, and how the classifier performs.

Preparing the Training Data

We now discuss how we prepared training data for our preposition insertion classifier. To begin, we have only trained and tested the classifier on data from Input 1. This is because Input 1 has a large amount of data, covering all 24 WSJ sections, and was the earliest available Input of this size. Naturally, we did not train or test the classifier on the parts of the WSJ which were reserved for testing.

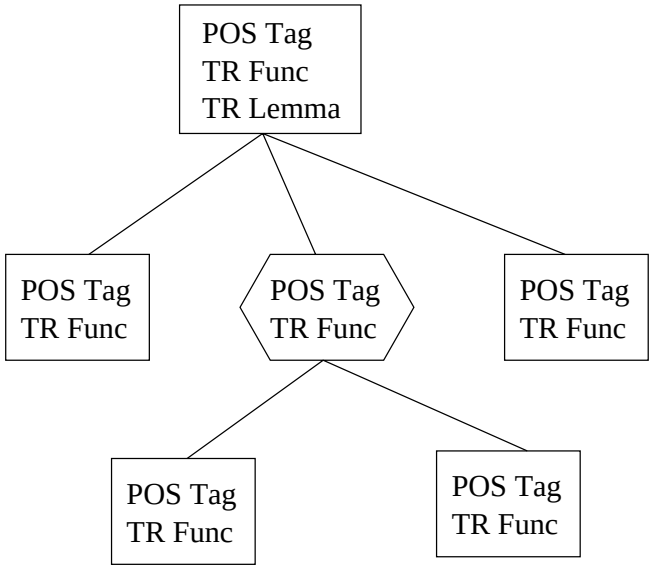


Figure 3.5: Inputs to the classifier when classifying current node (hexagon).

The classifier's inputs, displayed in Figure 3.5, are the attributes of the node to

be classified and a few of the surrounding nodes. Specifically, we pass the POS tag and TR functor of the node being classified, its parent node, its left and right brother nodes, and its leftmost and rightmost children. In addition, the TR lemma of the parent node is also included. We would ideally include more TR lemmas but doing so increases the size of the problem drastically and causes C5.0 to exit with an out of memory error.

One caveat about our inputs is that the POS tags in the classifier's input are not full POS tags — at transduction time, only simple POS tags are available. This simple POS tag can distinguish between basic categories such as noun, verb, and adjective, but does not make distinctions within these categories, such as `VBZ` and `VBD`. We simulate this by using only the first character of the full POS tag in the input to the classifier.

To create the data, we used a Perl script that traverses a TR tree, creating a single case entry for each node. The contextual information is gathered in fairly straightforward fashion, by examining the parent, left and right brother, and leftmost and rightmost children of the current node. The classification is a bit more tricky to find, however.

Since the TR trees from Input 1 are automatically generated from English surface text, they contain the original prepositions, marked as hidden nodes. The hidden prepositions are also attached at a different position in the TR than the AR; if a preposition is attached above node X in the AR tree, it is attached as the leftmost child of X in the TR tree. As a second check, we examine the corresponding AR tree and make sure that the hidden prepositions in the TR tree appear in the AR tree as parents of the current node, with AR functor `AuxP` and POS tag `IN`. These two quick checks will accept some subordinating conjunctions as prepositions, but this is reasonable since the two phenomena are closely related. Finally, the prepositions that pass both checks are concatenated with `insert`, forming the current node's classification. If no prepositions were found, the classification `nothing` is used.

In the previous paragraph, we have left out two important mechanisms for reasons of clarity. These mechanisms are conjunction and apposition hopping and skipping. We first describe conjunction and apposition hopping.

Suppose that we always looked at the literal parent node when gathering context information. In the TR of a sentence such as "John talked to Mary and to Jane" (see Figure 3.6), the parent of "Mary" and "Jane" is the conjunction "and". However, "and" provides no information about the relationship between "talked" and "Mary" or "Jane" represented by the preposition "to". We should have ignored "and" and looked at the parent of "and", "talked". Rather, "and" is simply a grouping node and the true parent, with respect to the preposition "to", is "talked". Through an identical mechanism, appositions also exhibit this spurious parenthood. To fix these errors, we look for the parent by hopping upward until we find a node that is neither a conjunction nor an apposition.

Conjunction and apposition skipping complements hopping. In the sentence "John

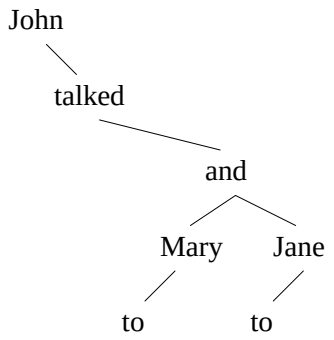


Figure 3.6: TR tree for “John talked to Mary and to Jane”

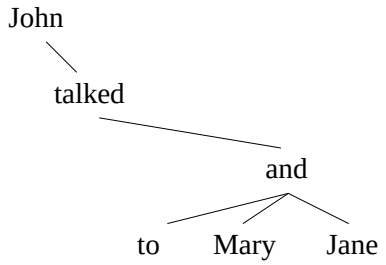


Figure 3.7: TR tree for “John talked to Mary and Jane”

talked to Mary and Jane” (see Figure 3.7), “and” would be classified `insert_to` by our naive data creation script. However, the “and” is not the actual target of the preposition “to”. Rather, the targets are “Mary” and “Jane”. We solve this problem by simply skipping over conjunction and apposition nodes without creating any data for them; we let conjunction and apposition hopping generate the appropriate data when “Mary” and “Jane” are processed. We must, however, ensure that the correct prepositions are detected when “Mary” and “Jane” are processed. Thus, we alter the hopping process to check for prepositions attached to any of the nodes on the path to the true parent. In the TR of the sentence “John talked to Mary and Jane”, “to” is attached to “and”, which is between “talked” and both “Mary” and “Jane”. This means that “Mary” and “Jane” are processed as if they had the preposition “to” inserted above them, even though they do not. Note that this makes the TR trees of Figures 3.6 and 3.7 identical with respect to the training data they produce. Since the two sentences are nearly identical in terms of meaning, and certainly interchangeable, we have decided that this is acceptable behavior.

Improving Performance

We now discuss how we improved the performance of our classifier. Our efforts have focused on increasing *insertion recall*, which we define as the number of correct preposition insertions the classifier makes divided by the total number of prepositions in the test set. We emphasize insertion recall because the classifier, when integrated, will make suggestions to the Proposer. Ideally, these suggestions

always include the correct answer. We focus only on insertions because our system attains perfect recall on `nothing`, for trivial reasons which we will explain later.

Our first performance problems derive from the high prior probability of the classification `nothing`, which makes up roughly $9/10$ of all classifications. On the other hand, there are roughly 330 classifications corresponding to preposition insertions, all sharing the remaining $1/10$ of the probability space. This makes the classifier tentative about inserting prepositions; if there is any uncertainty about whether to insert a preposition, the classifier will generally opt for `nothing` since it is right 9 times out of 10. Unfortunately, this tentativeness causes poor insertion recall.

We first attempted to deal with this problem by using C5.0's differential misclassification costs feature to favor insertions and disfavor `nothing`. However, we made little improvement with this approach and actually performed worse in some cases. We also attempted to use the adaptive boosting, hoping that C5.0 could detect that it was classifying many prepositions insertions as `nothing` and compensate for that in successive trials. However, attempts to train using adaptive boosting also met with failure: C5.0 aborted the training every time because the scores of the classifiers became too low.

The solution we eventually arrived at was to train on data that consisted only of preposition insertion cases. Our intuition was that if the `nothing` cases were causing confusion, then we should remove them. The classifier we trained on the preposition-only data is able to get significantly higher insertion recall than the classifier we trained on all the data, even though both classifiers are trained on the same preposition insertion cases. The obvious disadvantage is that a classifier trained on preposition-only data will never make a classification of `nothing`. We solve this problem by modifying the classifier to return two classifications: `nothing` and the preposition insertion determined by the decision tree. This explains why, as we mentioned above, we have perfect recall on `nothing`.

Notice that we are now returning multiple suggested classifications. Although this increases the number of possibilities that the tree transduction must evaluate, the increased insertion recall more than offsets the computational disadvantage. The natural step at this point is to alter our decision tree so that it returns multiple classifications.

Our first and most primitive application of this idea is what we call *N Thresholding*. To explain, when the normal classifier evaluates an input, it creates confidence values for each of the possible classifications and returns the classification with the highest confidence as its decision. A classifier using N Thresholding is the same except it returns the best N_{thresh} classifications. This method is able to get high insertion recall, but only when N_{thresh} is large. Furthermore, when N_{thresh} is large, there are many cases where the decision tree's highest or second highest confidence classification is correct, but the remaining classifications are still returned as dead weight. Manual study of the confidence values of in these cases shows that most of the confidence mass is concentrated in these one or two highest-ranked

classifications. Additionally, when the correct answer is in one of the lower-ranked classifications, the confidence mass is generally more spread out. These two observations lead us to our next thresholding technique.

The next kind of thresholding we developed is what we call *C Thresholding*, where C stands for "confidence". Instead of returning a fixed number of classifications, C Thresholding can return a variable number. C Thresholding operates by picking the smallest set of M classifications such that the sum of the confidence values of the M classifications is greater than fC_{total}. f is an adjustable parameter and C_{total} is the sum of confidence values for all classifications. Thus, if the confidence mass is concentrated in one or two classifications, the threshold will probably be passed using just those one or two classifications. If, on the other hand, the confidence mass is spread evenly, more classifications will be returned. This behavior follows the trends we have observed.

Note that although $C_{total} \leq 1$, it is not necessarily 1; hence the need to multiply f by C_{total} to create the threshold value. We set a hard limit on how many classifications can be returned, which is also an adjustable parameter. Thus far, however, we have used the hard limit 15 for all of our classifiers, and anticipate that changing the hard limit would not have any profound effects. For comparison to the N Thresholding classifier on a given set of cases, we calculate the "effective N_{thresh}" of a C Thresholding classifier as the average number of classifications suggested.

Unfortunately, when compared at equal values of N_{thresh}, the C Thresholding classifier does not perform much better than the N Thresholding classifier. This is strange, since manual study of insertion cases shows that C Thresholding consistently returns fewer classifications than N Thresholding.

The problem lies in the `nothing` cases. Since the classifier has been trained on preposition-only data, it has no knowledge of what to do with `nothing` cases. When it is presented with the input for a `nothing` case, it will produce a fairly even spread of very low confidences. As we have mentioned earlier, this causes the C Thresholding technique to return multiple classifications. After more manual study, we made the key observation that C_{total} is small for these `nothing` cases. This leads us to our next and final thresholding technique.

Our final thresholding technique is what we call *Aggressive C Thresholding*. This is identical to C Thresholding except the threshold value is $f(C_{total})^2$. Preposition insertion cases will have C_{total} reasonably close to 1, so the effect of squaring it is small and the number of classifications returned is similar to C Thresholding. However, for `nothing` classifications, C_{total} will be small, and squaring it will force the threshold value very low. Therefore, in these `nothing` cases, Aggressive C Thresholding returns far fewer classifications, usually only 1, than normal C Thresholding. Aggressive C Thresholding thus combines the accuracy of C Thresholding on preposition insertion cases with high economy on `nothing` cases.

Our current best classifier is trained on preposition-only data and uses Aggressive

N Thresholding		C Thresholding		Aggressive C	
N_{thresh}	**Recall (%)**	**Eff.** N_{thresh}	**Recall (%)**	**Eff.** N_{thresh}	**Recall (%)**
2	70.1568	2.8719	74.2187	**2.0697**	**71.1497**
3	76.3963	**2.9981**	**74.6136**	2.0992	71.6462
		3.1038	75.2680	2.1158	71.7026
		3.2714	77.4343	2.1493	73.4514
		3.3517	78.2241	2.1666	73.9366
		3.5324	78.8334	2.1702	73.9479
		3.7341	79.5555	2.1799	74.0494
4	80.5935	**4.0257**	**80.1647**	2.2271	74.5797
		4.4184	82.9065	2.2812	76.5655
5	82.9967	4.8659	84.1137	2.3397	76.7799
		5.3083	84.7004	2.3583	76.8927
6	84.8810	**5.9295**	**86.4380**	2.4066	77.2199
7	86.2011	6.6144	87.7581	2.4884	77.9871
8	87.2955	7.7973	89.0105	2.6519	78.8559
9	87.9838	**9.0266**	**89.8793**	**2.9678**	**80.1647**
10	88.6720	**10.1724**	**90.2516**	3.2876	80.8643
11	89.1459	12.5654	90.6916	3.5069	81.3156

Figure 3.8: Number of Classifications Returned and Percent Recall for the three thresholding methods. Values in both C Thresholding columns are **bold**-ed when their effective N_{thresh} is close to an integer. For easier comparison, the N Thresholding values have been moved next to their closest companions in the C Thresholding column.

C Thresholding to modulate its suggestions.

Results

We now present the results of testing our preposition insertion classifier. All of the results we include are of a classifier trained on preposition-only data, and using various thresholding techniques.

Figures 3.8 and 3.9 do not show any clear winner between N Thresholding and C Thresholding. Both, however, show that Aggressive C Thresholding has a clear advantage over the other two. The Aggressive C Thresholding classifier can classify with 80% insertion recall, using only about 3 classifications per case. Of course, this performance can be improved upon, but it is nevertheless an impressive start.

3.3.4 Future Work

There are a number of tasks we should undertake following this research. Clearly, the first task is integrating the above classifier with the Proposer. This will include training the classifier again on the other Inputs. Once integrated, we can evaluate the impact that the classifier has on overall English generation performance. From

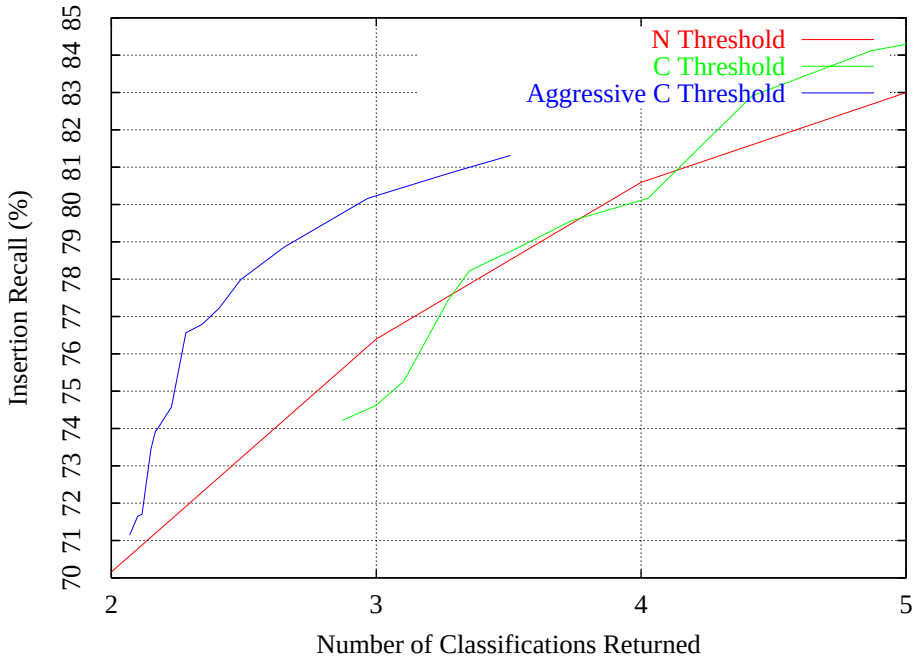


Figure 3.9: Percent Recall vs. Number of Classifications Returned for the three thresholding methods.

there, we will create more classifiers if the technique turns out to be useful, which we expect to be the case.

We should also improve the performance of the preposition insertion classifier. For instance, recall that we use only one TR lemma in the input because C5.0 cannot allocate enough memory. We would like to group TR lemmas, which have over 30,000 different values, into a smaller number of categories. Instead of directly using the TR lemmas as input, we can use the derived categories. Provided that the number of categories is small enough, we could use the categorized TR lemmas of multiple nodes.

Another improvement would focus on the creation of training data. Recall that we accept some subordinating conjunctions as prepositions. If we were able to separate prepositions from subordinating conjunctions, we could create a more uniform set of training data, which in turn might lead to a more accurate decision tree.

Finally, we could look at using several different classifier generators, with the hope that one might provide us significantly better performance.

To conclude, we have shown the effective of the general technique of using classifiers to capture tree information. We have also created a working classifier which achieves high recall on preposition insertions. We hope that in the future the preposition insertion classifier and other classifiers like it will be integrated into the tree transduction.

3.4 The Word Order Language Model

In this chapter we describe a tree-based language model used to choose the word ordering for the final output of the generation system. It implements the Step 3 of the generation system (see the start of Chapter 3). The model was used in conjunction which the tree transduction model described by section 3.1–section 3.3, which proposed a best *unordered* tree for each sentence. The language model was used to select the best ordering for the children of each node in the tree; thus the final word order was constrained by the input tree's bracketing.

The language model was trained on the hand-annotated parse trees of the Penn Treebank and evaluated on held-out data from the Treebank before being integrated into the generation system. We describe the two main type of probability models investigated, the surface bigram and tree bigram, and give performance results on re-ordering manually annotated parse trees in this chapter. Results will be given both for reordering the original trees from the Penn Treebank and the transformed "analytic representations" produced from them.

3.4.1 Surface Bigram

This model chooses the best ordering of the tree according to a bigram model of the resulting surface string. This model follows the approach to generation taken by (Langkilde and Knight, 1998), where a symbolic system generates a forest of candidate trees which are then ranked by a surface bigram model. This can be thought of as a division of the generation task into well-formedness and semantic accuracy on the one hand, which are guaranteed by the symbolic system, and fluency, which is estimated by the surface statistics.

The probability model used is the straightforward n-gram model commonly used in speech recognition:

$$P(w_{1..l}) = \prod_{i=1}^{l} P(w_i|w_{i-1})$$

where the word sequence $w_{1..l}$ is augmented with special START and STOP symbols at the beginning (w_0) and end (w_l) of the string.

We smoothed the bigram probabilities by backing off to the part-of-speech tags:

$$P(w_i|w_{i-1}) = \lambda\tilde{P}(w_i|w_{i-1}) + (1-\lambda)\tilde{P}(w_i|t_i)\tilde{P}(t_i|t_{i-1})$$

where $t_{1..l}$ are the part-of-speech tags given in the input tree, and $\tilde{P}$ represents the empirical distribution of the training data.

One advantage of the surface model is that it can easily be trained on large amounts of unannotated text. For our experiments, we made use of 23 million words of Wall Street Journal text, in addition to the roughly 1 million words of the Wall Street Journal portion of the Penn Treebank. This text was automatically part-of-speech tagged using the tagger of (Ratnaparkhi, 1996).

The search through possible re-orderings takes place in a bottom-up fashion using dynamic programming. Given the first and last words of one subtree's word

order, the probabilities for the orderings of material outside of a subtree are independent of the remaining internal ordering decisions. Thus, we compute and store the best order for one node's children for each combination of first and last words.

3.4.2 Tree-Based Bigram

This probability model, based on the model used for syntactic parsing by (Collins, 1997), is also composed of word-pair probabilities, but in this case the pairs are head-modifier relations from the tree itself, rather than adjacent words in the surface string. This probability model also predicts, and conditions on, the nonterminal labels in the tree, and thus makes use of more information than the surface bigram.

The model can be thought of as a variety of lexicalized probabilistic context-free grammar, with the rule probabilities factored into three distributions. The first distribution gives probability of the syntactic category H of the head child of a parent node with category P, head word *Hhw*, and head tag (the part of speech tag of the head word) *Hht*:

$$P_h(H|P, Hht, Hhw)$$

The head word and head tag of the new node H are defined to be the same as those of its parent. The remaining two distributions generate the non-head children one after the other, moving from the head child outward. A special STOP symbol is generated to terminate the sequence of children for a given parent. Each child is generated in two steps: first its syntactic category C and head tag *Cht* are chosen given the parent's and head child's features and a function Δ representing the distance from the head child:

$$P_c(C, Cht|P, H, Hht, Hhw, \Delta, d)$$

where d indicates direction, left or right, of the new child from the head. Then the new child's head word *Chw* is chosen:

$$P_{cw}(Chw|P, H, Hht, Hhw, \Delta, d, C, Cht)$$

For each of the three distributions, the empirical distribution of the training data is interpolated with less specific backoff distributions.

We follow (Collins, 1999) in special handling of punctuation, conjunctions, and base noun phrases. In a probabilistic context-free grammar, the ordering decisions for material outside a subtree are independent of internal ordering decisions, meaning that we store only one best ordering for each node's children. Strictly speaking, the adjacency feature of Collins' model (Δ above) does not obey this context-free property, as the probabilities for higher-level ordering depend on whether the head word of a constituent appears at it edge. For simplicity we did not include a search through values of Δ in our implementation.

Head-Modifier vs. Tree Bigram Probabilities

Of the three probabilities computed by the Collins model, the first, P_h is the same for all re-orderings of a fixed tree, assuming that the head child is known and held constant. For the probabilities P_c and P_{cw}, only the values of the variable d

(left or right) and Δ (the distance) vary among re-orderings. Although the model contains bilexical probabilities P_{cw}, which can be compared to our surface bigram probabilities, these are computed between heads and their modifiers, and the set of head-modifier relations do not change with re-ordering.

The exception to this in the original Collins model is the handling of base noun phrases, for which the previously generated child in the sequence is always considered the head for the purposes of probability estimation. In terms of our probabilities, this mean that H, Hhw, and Hht are defined to be the previously generated child rather than the head child, meaning that for the base noun phrase "the back cat", the probabilities for generating the non-head children moving from "cat" to the left are:

$P_c(C$=JJ, *Cht*=JJ$|P$=NPB, H=NN, *Hht*=NN, *Hhw*=cat, ...)
$P_{cw}($*Chw*=black$|P$=NPB, H=NN, *Hht*=NN, *Hhw*=cat, ...)
$P_c(C$=DT, *Cht*=DT$|P$=NPB, H=JJ, *Hht*=JJ, *Hhw*=black, ...)
$P_{cw}($*Chw*=the$|P$=NPB, H=JJ, *Hht*=JJ, *Hhw*=black, ...)
$P_c(C$=STOP$|P$=NPB, H=DT, *Hht*=DT, *Hhw*=the, ...)

where NN is the tag for noun, JJ for adjective, and DT for determiner, rather than:

$P_c(C$=JJ, *Cht*=JJ$|P$=NPB, H=NN, *Hht*=NN, *Hhw*=cat, ...)
$P_{cw}($*Chw*=black$|P$=NPB, H=NN, *Hht*=NN, *Hhw*=cat, ...)
$P_c(C$=DT, *Cht*=DT$|P$=NPB, H=NN, *Hht*=NN, *Hhw*=cat, ...)
$P_{cw}($*Chw*=the$|P$=NPB, H=NN, *Hht*=NN, *Hhw*=cat, ...)
$P_c(C$=STOP$|P$=NPB, H=NN, *Hht*=NN, *Hhw*=cat, ...)

One reason for this special treatment of base noun phrases is to force determiners to appear at the left edge of the phrase, adjecent to the STOP symbol. For constituents other than base noun phrases, the only difference between re-orderings among children to one side of the head is value of Δ. This feature is in fact a vector of three binary values: whether the modifer is immediately adjacent to the head, whether there is an intervening verb, and whether there is intervening punctuation. The model often does not distinguish at all between re-orderings among those children not immediately adjacent to the head.

In order to make the model more discriminative for the purpose of choosing an ordering, we used the "rolling head word" for all constituents, not just base noun phrases. We call the resulting model a tree-bigram model, in order to distinguish it from both the surface bigram and head-modifier (original Collins) models.

3.4.3 Results

Table 3.1 shows results from both the surface bigram and tree-based language model for the original Penn Treebank trees as well as for the Analytic Representation format. Our performance metric is the percentage of nodes whose children are assigned the correct ordering by the model. No partial credit is assigned for orderings that are close but not identical to the correct ordering.

The *Chance* column represents the performance one would obtain by choosing ordering randomly. For example, if the all nodes in all the trees had two children, this figure would be .5; if all nodes had three children, it would be $\frac{1}{3}$. Because of

the large numbers of unary nodes in our trees, chance performance is in fact higher than .5.

For reasons of computational complexity, we did not attempt to re-order nodes with more than six children. This is reflected in the column label *Max*, which gives the percentage of nodes with 6 or fewer children. This is the maximum the algorithm could get correct, as nodes with 7 or more children are always counted as incorrect.

	Total	Chance	Max.	Tree Bigram	Surface Bigram
Penn Treebank	52984	62.1	99.3	95.6	89.0
Penn Treebank w/o punc.	52984	63.5	99.9	96.7	88.7
Analytic Rep.	48206	69.7	98.9	91.2	86.5
Analytic Rep. w/o punc.	48206	68.9	99.6	92.9	86.0

Table 3.1: Word-order results

The "tree-bigram" modification of the Collins model improved performance, from 93.9% for the original Collins model to the 95.6% shown for the tree-bigram model in Table 3.1 for the Penn Treebank. The increase in performance is small in comparison to the amount by which both tree-based models outperform the surface bigram, which achieved 89.0% accuracy. This may seem surprising, since the original Collins probability model was designed and tuned to assign structure to a fixed string, and might not be expected to discriminate between re-orderings of a valid tree. We believe that it does better because it has access to more information. Making use of the syntactic categories of the tree nodes provides a useful level of backoff beyond the part of speech information used by the bigram model. The tree-based model also makes use of lexical head information not available to the surface bigram.

Analysis: Unlabeled Dependency Model

In order to measure how much of the performance of the tree-based model was due to the syntactic category information and how much was simply due to the information as to the subtrees head, a model was trained using the same tree structure as the Penn Treebank, and the same heads for each constituent, but with all constituent labels replaced with an 'X' symbol. This representation is similar to an unlabeled lexical dependency graph, with the exception that intermediate nodes with the same head are preserved from the original Penn Treebank tree (for example the VP node under an S, both of which are headed by the sentence's main verb). One motivation for this experiment was to see how important it was for the translation system to provide syntactic categories in trees it generates, or whether unlabeled dependency graphs would be sufficient.

This model performed at 92.1% on our test set, significantly worse than the 95.6% of the full tree bigram model on the same data, but still better than the 89.0% of the surface bigram (first row of Table 3.1). For this reason, the decision was made to generate syntactic labels as part of the translation model rather than unlabeled dependency graphs.

Sample Output

Below are the first five sentences of Section 23 of the Wall Street Journal, showing both the original text (a) and the output of the re-ordering model (b):

1a) No , it was n't Black Monday .

1b) it was n't Black Monday . , No

2a) But while the New York Stock Exchange did n't fall apart Friday as the Dow Jones Industrial Average plunged 190.58 points – most of it in the final hour – it barely managed to stay this side of chaos .

2b) But while the New York Stock Exchange did n't fall apart Friday as the Dow Jones Industrial Average plunged 190.58 points – most of it in the final hour – it barely managed to stay this side of chaos .

3a) Some " circuit breakers " installed after the October 1987 crash failed their first test , traders say , unable to cool the selling panic in both stocks and futures .

3b) Some " circuit breakers " installed after the October 1987 crash failed their first test , traders say , unable to cool the selling panic in both stocks and futures .

4a) The 49 stock specialist firms on the Big Board floor – the buyers and sellers of last resort who were criticized after the 1987 crash – once again could n't handle the selling pressure .

4b) once again The 49 stock specialist firms on the Big Board floor – – the buyers and sellers of last resort who were criticized after the 1987 crash could n't handle the selling pressure .

5a) Big investment banks refused to step up to the plate to support the beleaguered floor traders by buying big blocks of stock , traders say .

5b) Big investment banks refused to step up to the plate to support the beleaguered floor traders by buying big blocks of stock traders say . ,

Two of the five sentences are identical to the original, one differs only in the misplacement of a comma, and the remaining two are re-ordered in ways that still read more or less coherently and preserve the original meaning.

3.5 English Morphological Generation

English morphological generation module (implementation of Step 4, see the start of Chapter 3) has been developed in the form of a simple table that maps English lemmas and tags to wordforms. The table has been created with the help of the freely available tool "morpha" (Minnen, Caroll, and Pearce, 2001) and adapted to match the style of lemmatization used throughout the project.

As a part of the English morphological generation subsystem, orthography and punctuation insertion has been worked on and is part of the final system that has been evaluated.

Chapter 4

The Generation System ARGENT

Dragomir Radev

This part of the system is an alternative to a fully statistically trained generation system. The purpose was to compare the rule-based approach (with hand-written rules within an off-the-shelf system, FUF in this case), in the given workshop time-frame, to a fully statistical system described earlier (and evaluated in Chapter 6).

4.1 Introduction

This section describes ARGENT, a rule based generation system. ARGENT is built on top of the FUF sentence generator (Elhadad, 1991; Elhadad, 1993).

FUF is a general-purpose generation framework. It takes as input a lexicalized representation of a sentence and unifies it with a large systemic grammar of English, called Surge, to fill in syntactic constraints, build a syntactic tree, choose closed class words, and eventually linearize the tree as a sentence. FUF has been used in the past for generation in a large number of domains such as sports reports(Robin, 1994), multimedia explanations(Feiner and McKeown, 1991), and summaries (Radev and McKeown, 1998).

ARGENT includes a manually-built functional unification grammar which was developed from section 00 of the Penn Treebank. The results shown in this section are based on the dev-test and eval sections of the Treebank also used in the rest of this report.

The input of ARGENT is a sentence in the PDT framework with all tectogrammatical roles specified. Figure 4.1 shows a sample input in PDT format. This representation includes base forms for all content words as well as their functors. The root node corresponds to the word “join” with a functor PRED (predicate). The input corresponds to the (world famous) first sentence (wsj-0000-001) of the Penn Treebank “Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.”. The following example (Figure 4.2) matches sentence wsj-0012-003 “Alan Spoon, recently named Newsweek president, said Newsweek’s ad rates would increase 5% in January.”

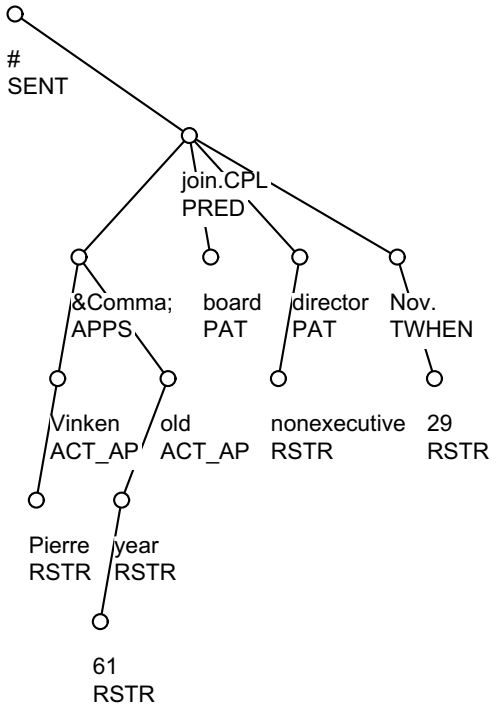


Figure 4.1: Sample PDT tree for (wsj-0000-001).

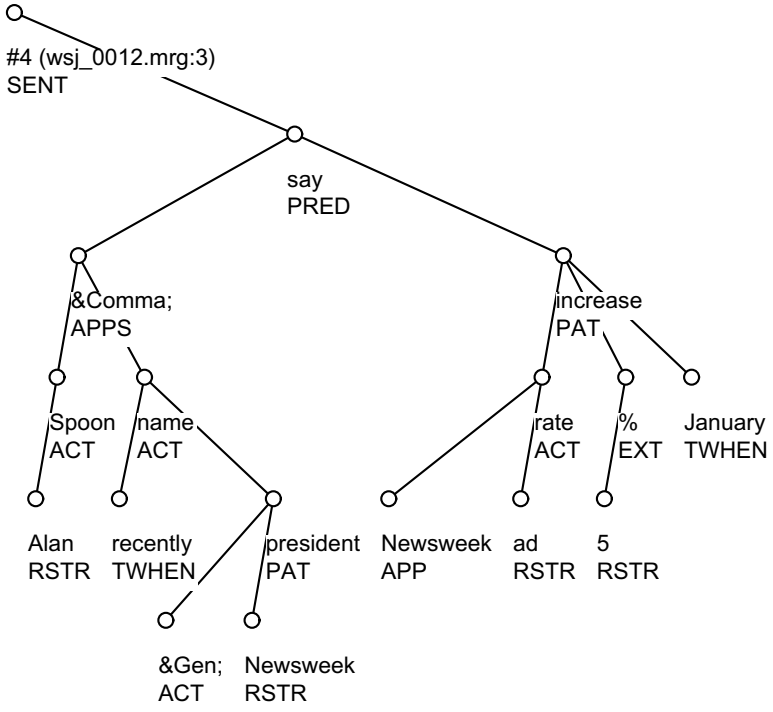


Figure 4.2: Sample PDT tree for (wsj-0012-003).

4.2 ARGENT architecture

In order to get from the PDT representation back to a running sentence, ARGENT goes through the following stages:

1. PDT is converted to a (FUF-style) FD format (Figure 4.3). All grammatemes are ignored. Order is the same as in PDT (at each level). An equivalent representation is shown in Figure 4.4.

```
(setq fdfs/d8802c.wsj_0001.tgt.000.fsp
  '((in ((cat clause)
         (PRED ((lex "join")
                (APPS ((lex "\,")
                       (ACT ((lex "Vinken")
                             (RSTR ((lex "Pierre")))))
                       (ACT-1 ((lex "old")
                               (RSTR ((lex "year")
                                      (RSTR ((lex "61")))))))))
                (PAT ((lex "board")))
                (PAT-1 ((lex "director")
                        (RSTR ((lex "nonexecutive")))))
                (TWHEN ((lex "Nov.")
                        (RSTR ((lex "29"))))))))))
```

Figure 4.3: FD representation for wsj-000-01.

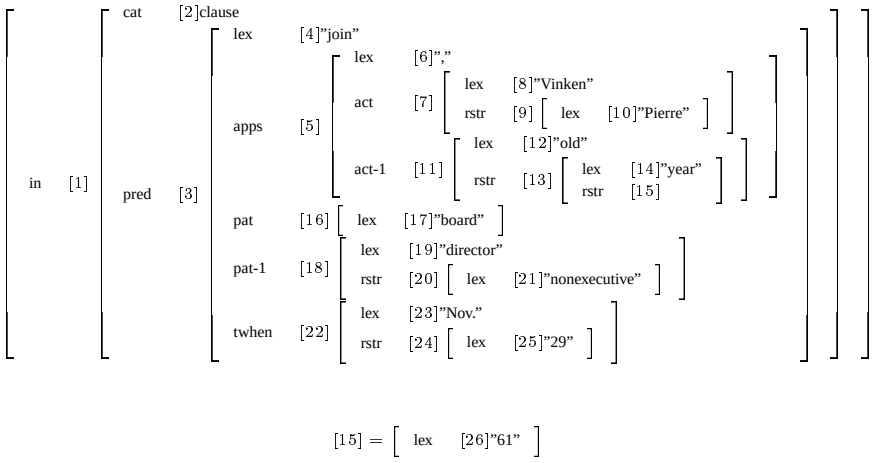


Figure 4.4: FD for "Pierre Vinken" (wsj-0001-01) before lexicalization.

2. The FD is unflattened and lexicalized with the ARGENT FUF grammar. At this stage, it is decided what additional words to add to the representation. The unflattening process converts the dependency representation to a constituent representation, e.g., transforming the PRED node to a syntactic subtree headed by a CLAUSE node. The representation at this stage of the Pierre Vinken sentence is shown in Figure 4.5.

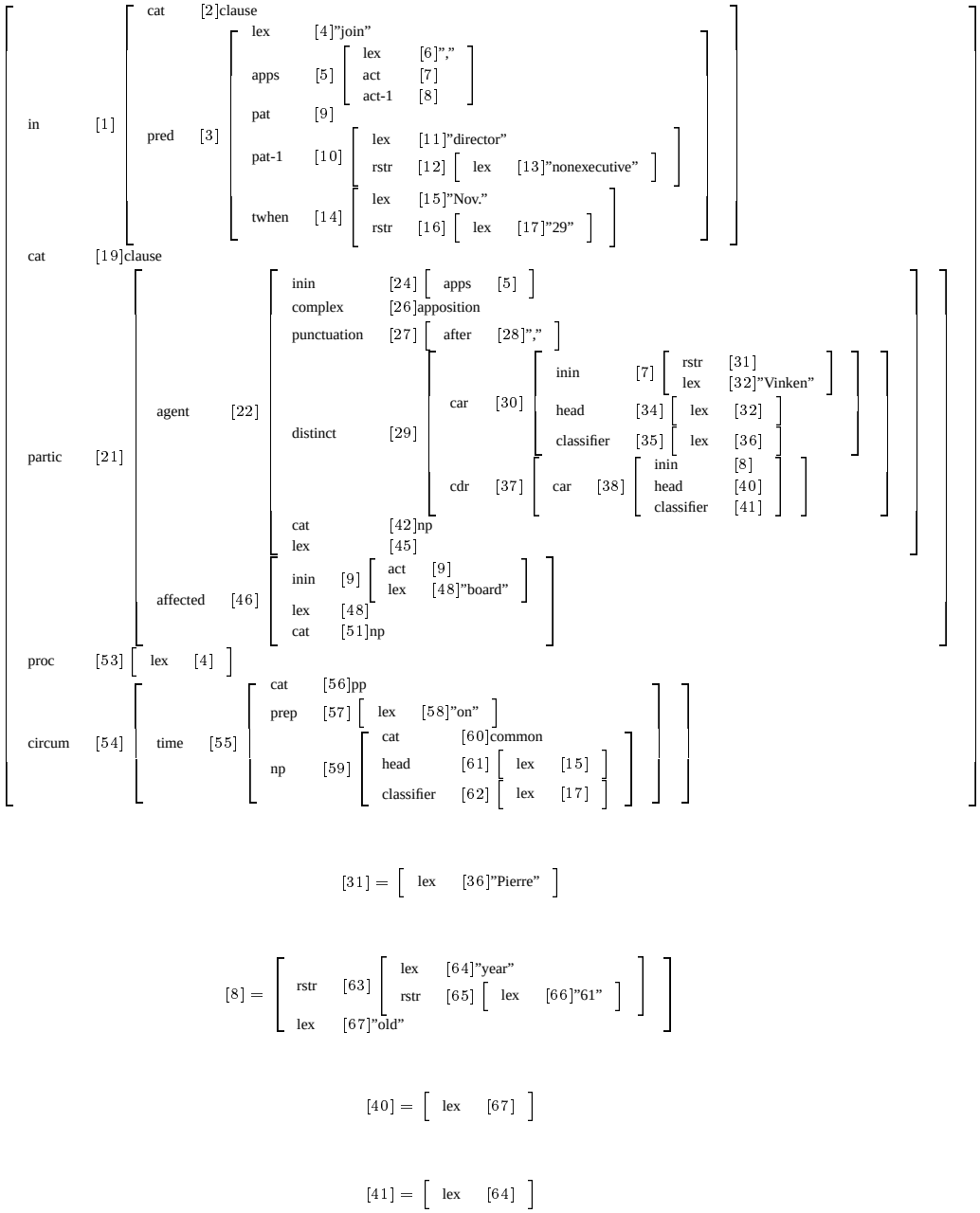


Figure 4.5: FD for "Pierre Vinken" (wsj-0001-01) after lexicalization.

3. The result is unified with Surge which produces a fully grammatical running (surface) sentence.

One implementation remark is needed here. The PDT representation allows for double fillers for given slots. For example, Figure 4.1 contains two PAT fillers at the /PRED level. These are automatically converted to PAT and PAT1 in the FUF framework. The order is the same in which they appear in the PDT version.

4.3 Example

An example of generation follows. Figure 4.6 shows the second (Allan Spoon) FD before lexicalization. Three examples will be shown, each getting closer to the correct surface sentence.

1. The baseline sentence that one could generate from the PDT representation is “Alan Spoon, recently name Newsweek president, say Newsweek ad rate increase 5% January”.

2. The FD shown in Figure 4.8 generates the (incorrect) sentence ”Alan Spoon, the president named, says that Newsweek’s ad rate increases the 5 % January.”. At this level, one can notice several improvements, in particular the presence of a relative clause headed by “say”, proper subject-verb agreement (“says”, “increases”), correct possessive (“Newseek’s”) as well as some omissions (e.g., “recently”).

3. The FD shown in Figure 4.7 generates the sentence ”Alan Spoon, Newsweek president name, said that Newsweek’s ad rates increased five pct in January.”.

The difference between these last two FDs is that the first one is automatically generated by ARGENT from the PDT representation of the sentence while the second one is the ideal FD from which Surge would produce the correct output. In other words, the goal of the ARGENT grammar is to move away from the baseline towards the correct output. Adding more rules could take the system closer to the correct output. With the current set of manual rules (developed over a three week period in weeks 3-5 of the summer workshop), the output of ARGENT is more like the middle example above. The example in Figure 4.7 was created manually for illustration purposes. It is not an actual output of ARGENT.

4.4 The ARGENT grammar

So how does ARGENT get from the PDT representation and to the input to Surge as shown in Figure 4.8? It makes use of close to one hundred head-based rules which convert different functors and their descendants to a constituent format. Figure 4.9 shows some sample tectogrammatical functors. Each of these corresponds to a set of rules in the ARGENT grammar.

In place of functors, FUF uses the following constituents: a *process*, a number of participants *partic* (e.g., agent or affected), and a number of circumstantials (*circum*). Figure 4.10 shows the equivalent SURGE constructs.

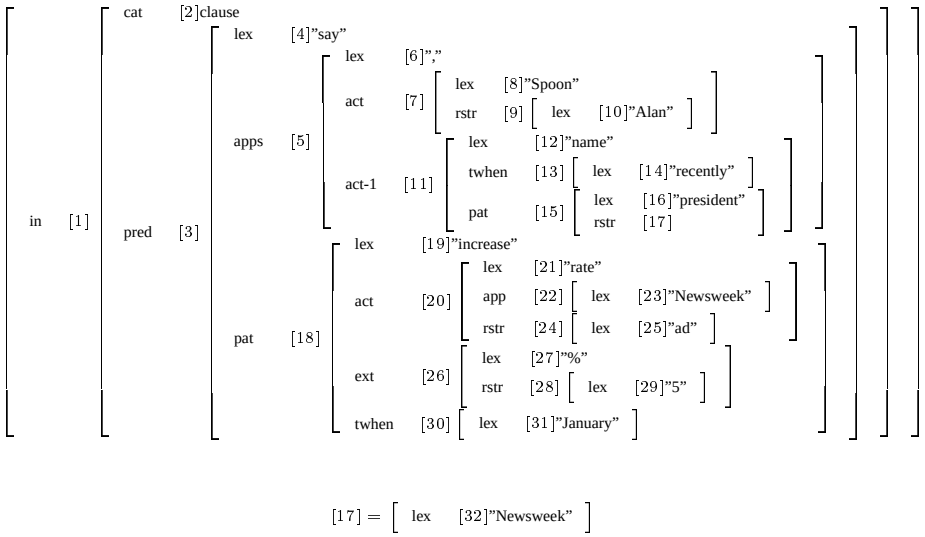


Figure 4.6: ARGENT FD for ”Alan Spoon” before lexicalization.

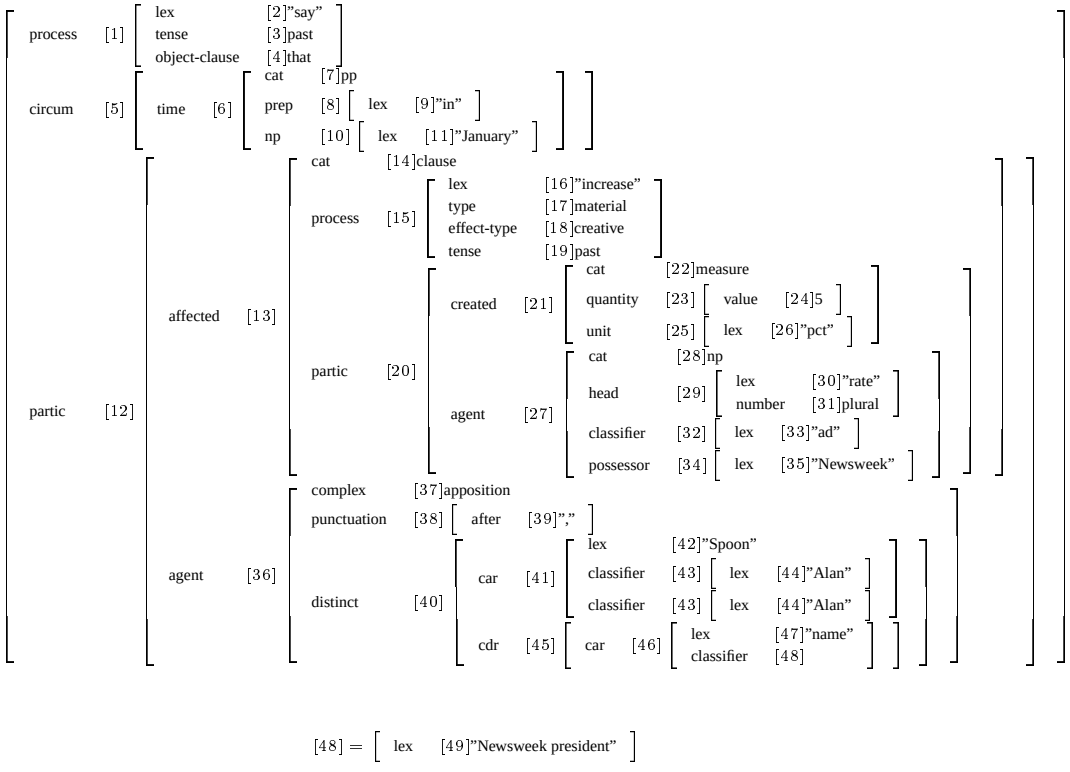


Figure 4.7: Target FD for ”Alan Spoon” (wsj-0012-003) after lexicalization.

```
((IN
  ((CAT CLAUSE)
   (PRED
    ((LEX "say")
     (APPS
      ((LEX ",") (ACT {PARTIC SAYER ININ})
       (ACT-1
        ((LEX "name") (TWHEN ((LEX "recently")))
         (PAT ((LEX "president") (RSTR ((LEX "Newsweek")))))))))
     (PAT
      ((LEX "increase") (ACT {PARTIC VERBALIZATION PARTIC AGENT ININ})
       (EXT {PARTIC VERBALIZATION PARTIC AFFECTED ININ})
       (TWHEN ((LEX "January") (RSTR ((LEX NONE)))))
       (PRED {IN PRED PAT}) (MANN ((LEX NONE))) (LOC ((LEX NONE)))))
     (TWHEN ((LEX NONE))) (MANN ((LEX NONE))) (LOC ((LEX NONE)))))))
 (CAT CLAUSE) (M-LEX-PRED 0C)
 (PROC ((TYPE VERBAL) (OBJECT-CLAUSE THAT) (LEX {IN PRED LEX})))
 (PARTIC
  ((SAYER
    ((IN {IN PRED}) (M-IN-ACT-CHECK 1)
     (ININ
      ((APPS {PARTIC SAYER IN APPS}) (LEX "Spoon")
       (RSTR ((LEX "Alan"))) (APP NONE)))
     (CAT NP) (M-LEX-NPLIKE 3) (M-LEX-NP 1)
     (HEAD ((LEX {PARTIC SAYER ININ LEX})))
     (CLASSIFIER ((LEX {PARTIC SAYER ININ RSTR LEX})))
     (QUALIFIER
      ((CAT CLAUSE) (PUNCTUATION ((BEFORE ","))) (MOOD PAST-PARTICIPLE)
       (PROC {PARTIC SAYER ININ APPS ACT-1})
       (PARTIC ((AFFECTED {PARTIC SAYER ININ APPS ACT-1 PAT})))))))
   (VERBALIZATION
    ((IN {IN PRED PAT}) (CAT CLAUSE) (M-LEX-PRED 3)
     (PARTIC
      ((AGENT
        ((IN {PARTIC VERBALIZATION IN PRED}) (M-IN-ACT-CHECK 0)
         (ININ
          ((ACT {PARTIC VERBALIZATION PARTIC AGENT IN ACT})
           (LEX "rate") (APP ((LEX "Newsweek"))) (RSTR ((LEX "ad")))))
         (CAT NP) (M-LEX-NPLIKE 1) (M-LEX-NP 1)
         (HEAD ((LEX {PARTIC VERBALIZATION PARTIC AGENT ININ LEX})))
         (CLASSIFIER
          ((LEX {PARTIC VERBALIZATION PARTIC AGENT ININ RSTR LEX})))
         (POSSESSOR
          ((LEX {PARTIC VERBALIZATION PARTIC AGENT ININ APP LEX})
           (CAT PROPER)))
         (M-LEX-NP-APP 0)))
       (AFFECTED
        ((IN {PARTIC VERBALIZATION IN PRED}) (M-IN-EXT-CHECK 0)
         (ININ
          ((PAT {PARTIC VERBALIZATION PARTIC AFFECTED IN EXT})
           (ACT NONE) (LEX "\%") (RSTR ((LEX "5"))) (APP NONE)))
         (CAT NP) (M-LEX-NPLIKE 2) (M-LEX-NP 1)
         (HEAD ((LEX {PARTIC VERBALIZATION PARTIC AFFECTED ININ LEX})))
         (CLASSIFIER
          ((LEX {PARTIC VERBALIZATION PARTIC AFFECTED ININ RSTR LEX})))))))
     (PROC ((LEX {PARTIC VERBALIZATION IN PRED LEX})))
     (LEX-CSET ({^ PARTIC AGENT} {^ PARTIC AFFECTED}))
     (CIRCUM
      ((TIME ((LEX {PARTIC VERBALIZATION IN PRED TWHEN LEX})))))))))
 (LEX-CSET
  ({^ PARTIC SAYER} {^ PARTIC VERBALIZATION PARTIC AGENT}
   {^ PARTIC VERBALIZATION PARTIC AFFECTED} {^ PARTIC
  VERBALIZATION})))
```

Figure 4.8: ARGENT-augmented FD for ”Alan Spoon” after lexicalization.

Functor	Description	Example
ACT	actor	**Pierre Vinken** joined the board.
PAT	patient	Pierre Vinken joined **the board.**
APPS	apposition	Pierre Vinken, **61 years old**...
PRED	predicate	Pierre Vinken **joined** the board.
TWHEN	time-when	He joined the board **on November 29**.

Figure 4.9: Sample tectogrammatical functors.

Construct	Example
process	joined
partic/agent	Pierre Vinken
partic/affected	the board
circum	on November 29

Figure 4.10: Sample SURGE constructs.

Let's now go over a specific PDT to FUF translation example. If the PRED of a PDT sentence is "say" and its PAT is headed by a verb (as opposed to a noun), the PRED-PAT PDT subtree needs to be translated as a much more elaborate tree in FUF. The resulting tree needs to indicate that the verb "say" in this case subcategorizes for a relative clause and it needs to build up the syntactic subtree for the relative clause. In FUF, this corresponds to the grammar snippet shown in Figure 4.11. The input IN-PRED is converted to a subtree consisting of a "proc" (process) and a "partic" (participant). The "proc" subcategorizes for an "object-clause". The "partic" subtree itself contains two constituents: a "sayer" and a "verbalization". The "sayer" is mapped to the subject of the predicate "pred" while the verbalization is realized as a clause which recursively expands IN-PRED-PAT. The "lex-cset" operator is FUF-specific and indicates to FUF's lexical chooser which nodes need to be expanded at the next level. In this example, four subconstituents need to be expanded: partic-sayer, partic-verbalization-partic-agent, partic-verbalization-partic-affected, and partic-verbalization.

The macro in-act-check (shown in Figure 4.12) searches for the head of the "actor" constituent. It first looks for an apposition ("apps"), if there is no such element in the input, it looks for a conjunction ("conj"), and finally, for an "act".

The following figure (Figure 4.13) shows how an apposition is further expanded. A PDT-style APPS contains in this example, two PATs (represented as PAT and PAT-1). In Surge, the corresponding construct is a "distinct" with two components, matching the PAT and PAT-1 portions of the input, respectively.

Figure 4.14 shows the general structure of the ARGENT grammar. Each node corresponds to a major component of the grammar.

Figure 4.15 shows the most frequent paths in section 00 of the Penn Treebank. These paths were implemented in the grammar.

4.5 Evaluation

We have evaluated our manual grammar using the DT and EV sections of the PDT corpus as a blind test section. All grammar development was done on section 00

```
((alt (((in ((pred ((lex "say"))))))
       ((in ((pred ((lex "report")))))))))

 (proc ((type verbal)
        (object-clause that)
        (lex {^2 in pred lex})))

 (partic ((sayer ((in {^3 in pred})
                  (:! in-act-check)
                  (cat np)
                  ))
          (verbalization ((in {^3 in pred pat})
                          (cat clause)
                          (in ((pred {^4 in pred pat})))
                          )
                         )
          ))

 (lex-cset ({^ partic sayer}
            {^ partic verbalization partic agent}
            {^ partic verbalization partic affected}
            {^ partic verbalization}
            ))
 )
```

Figure 4.11: Lexicon entry for verbs like ”say” and ”report” in the ”X says that Y” subcategorization where “that Y” is a relative clause.

```
(def-alt in-act-check
  (
   ((in ((apps ((act given)))))
    (in ((apps ((act-1 given)))))
    (inin ((apps {^2 in apps})))))
   ((in ((conj ((act given)))))
    (in ((conj ((act-1 given))))))
   ((in ((act any)))
    (inin ((act {^2 in act}))))
   )
  )
```

Figure 4.12: Actor head guessing macro.

```
((in ((apps given)))
 (in ((apps ((pat given)))))
 (in ((apps ((pat-1 given)))))
 (complex apposition)
 (punctuation ((after ",")))
 (distinct ~(((:& lex-np)
              (in {^3 in apps pat}))
             ((:& lex-np)
              (in {^4 in apps pat-1})))))
```

Figure 4.13: Converting an APPS to Surge-style apposition.

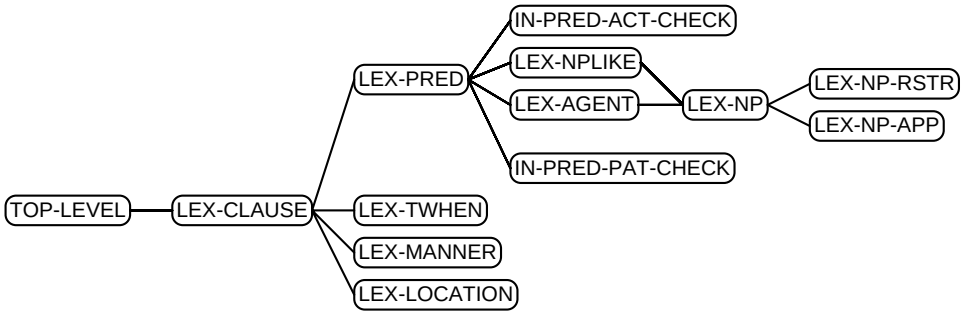


Figure 4.14: General structure of the ARGENT grammar.

Frequency	Path
1642	/PRED
1430	/PRED/ACT
1420	/PRED/PAT
653	/PRED/ACT/RSTR
617	/PRED/PAT/ACT
554	/PRED/PAT/PAT
526	/PRED/PAT-1
421	/PRED/PAT/RSTR
346	/PRED/TWHEN
263	/PRED/ACT/RSTR-1
218	/PRED/APPS
214	/PRED/PAT/RSTR-1
214	/PRED/PAT/PAT/RSTR
169	/PRED/PAT-1/ACT
168	/PRED/CONJ
161	/PRED/APPS/ACT
159	/PRED/LOC
152	/PRED/PAT/APP
149	/PRED/PAT/ACT/RSTR
149	/PRED/PAT-1/PAT
147	/PRED/ACT/APP
143	/PRED/APPS/ACT/RSTR
139	/PRED/APPS/ACT-1
138	/CONJ/PRED-1
135	/PRED/CONJ/PAT-1
126	/PRED/RHEM
126	/PRED/MANN
124	/PRED/PAT-1/RSTR
119	/PRED/PAT/TWHEN
115	/PRED/PAT/PAT/ACT

Figure 4.15: Most frequent dependency paths in section 00 of the Penn Treebank.

```
((CAT CLAUSE)
 (PRED
  ((LEX "join")
   (APPS
    ((LEX ",") (ACT {PARTIC AGENT DISTINCT CAR LOAD})
     (ACT-1 {PARTIC AGENT DISTINCT CDR CAR LOAD})))
   (PAT {PARTIC AFFECTED LOAD})
   (PAT-1 ((LEX "director") (RSTR ((LEX "nonexecutive")))))
   (TWHEN ((LEX "Nov.") (RSTR ((LEX "29"))))) (ACT ((APP NONE)))
   (LOC ((LEX NONE))) (MANN ((LEX NONE)))))
 (PROCESS ((LEX {PROCESS LOAD LEX}) (LOAD {PRED})))
 (PARTIC
  ((AFFECTED
    ((LOAD ((RSTR NONE) (APPS ((APP NONE))) (LOC NONE) (LEX "board")))
     (LEX {PARTIC AFFECTED LOAD LEX}) (DETERMINER NONE)))
   (AGENT
    ((COMPLEX APPOSITION) (PUNCTUATION ((AFTER ","")))
     (DISTINCT
      ((CAR ((CLASSIFIER
              ((LEX {PARTIC AGENT DISTINCT CAR LOAD RSTR LEX})))
             (LOAD ((RSTR ((LEX "Pierre"))) (LEX "Vinken")))
             (LEX {PARTIC AGENT DISTINCT CAR LOAD LEX})
             (DETERMINER NONE)))
       (CDR ((CAR ((CLASSIFIER
                    ((CAT NOUN-COMPOUND)
                     (HEAD
                      ((LEX
                        {PARTIC AGENT DISTINCT CDR CAR LOAD RSTR LEX})))
                     (CLASSIFIER
                      ((LEX
                        {PARTIC AGENT DISTINCT CDR CAR LOAD RSTR RSTR LEX})))))
                   (LOAD
                      ((RSTR ((RSTR ((LEX "61"))) (LEX "year")))
                       (LEX "old")))
                   (LEX {PARTIC AGENT DISTINCT CDR CAR LOAD LEX})
                   (DETERMINER NONE)))
              (CDR NONE)))))
     (LHS {PARTIC AGENT DISTINCT CAR LOAD})
     (RHS {PARTIC AGENT DISTINCT CDR CAR LOAD})))))
 (CIRCUM
  ((PERSPECTIVE
    ((CAT PP)
     (NP
      ((CAT COMMON) (DEFINITE NO)
       (CLASSIFIER ((LEX {PRED PAT-1 RSTR LEX})))
       (HEAD ((LEX {PRED PAT-1 LEX}))))))))
   (TIME ((CAT PP) (PREP ((LEX "on")))
          (NP
           ((CAT COMMON) (DETERMINER NONE)
            (HEAD ((LEX {PRED TWHEN LEX})))
            (CLASSIFIER ((LEX {PRED TWHEN RSTR LEX}))))))))))
```

Figure 4.16: FD for Pierre Vinken after lexicalization.

of the PDT and development testing (validation) on section 01 of the PDT. The results are generated using BLEU (BiLingual Evaluation Understudy) (Papineni et al., 2002) which is a linear combination of n-gram precision with n from 1 to 4, along with a penalty for brevity.

Results using two methods are reported. D5 refers to the latest version of the ARGENT grammar while D5-r also includes a postprocessing stage in which all words appearing in the PDT representation but which were not used by ARGENT are concatenated to the end of the ARGENT output. Doing so automatically increases unigram precision which contributes to a higher BLEU score.

Method	**00**	**DT**	**EV**
D5	0.146	0.081	0.067
D5-r	0.260	0.155	0.145

Table 4.1: Bleu scores.

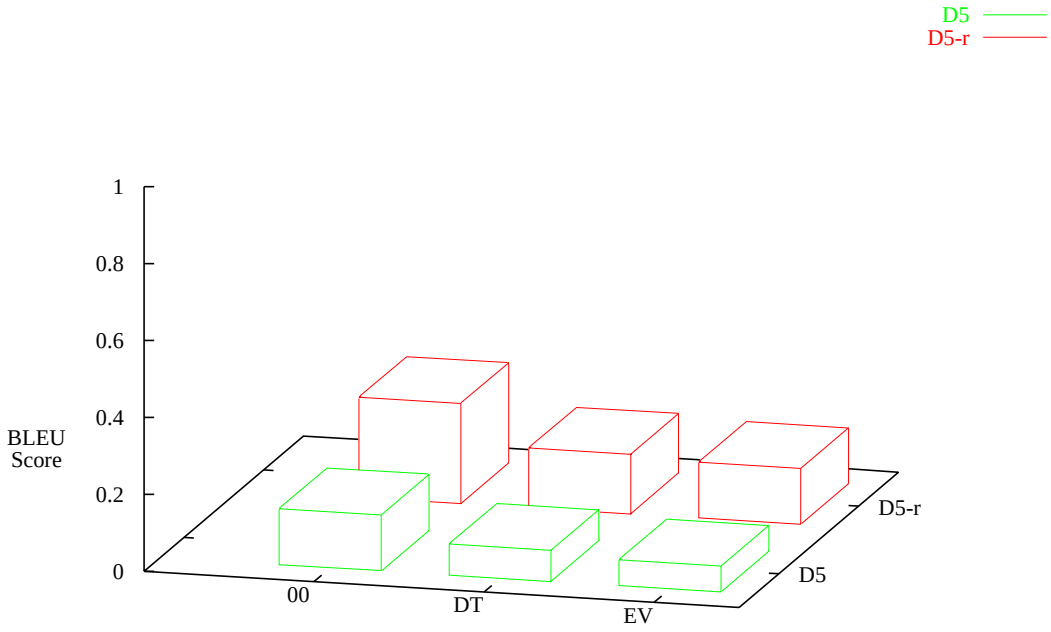


Figure 4.17: Results.

4.6 Future work

Clearly, building PDT to FUF rules by hand is a time-consuming, and possibly unnecessary task. One of the main steps to follow this work is to develop a machine learning framework to build tree-to-tree alignments from PDT to FUF. Such

work will hopefully be built on top of the existing tree-to-tree framework described elsewhere in this report.

One can cast the following FUF "realization switches" as decisions to be made by the learning component: tense/number, order of adverbs, choice of determiners, punctuation, etc. In that sense, the PDT to FUF translation is not fundamentally different from the tree-to-tree translation described in the report. One major difference, however, is that the size of the trees (especially on the FUF side) can be significantly larger than the sizes prescribed in the tree-to-tree framework. On the other hand, the FUF trees can possibly be represented at a higher granularity, treating each consistuent as an independent unit.

Chapter 5

English Tectogrammatical Parsing

Owen Rambow, Bonnie Dorr and Ivona Kučerová

5.1 Introduction

Throughout this project, we have assumed that the starting point for the generation of English, namely, the Tectogrammatical Representation of English sentences is given and available, both for training and testing (evaluation). The latter was certainly true; see later the Chapter on system evaluation. However, for training, despite having the manually annotated Penn Treebank available, it is *not* available at the Tectogrammatical Level of annotation assumed by the Machine Translation scenario we have adopted. Nevertheless, we have tried to analyze the possibilities of using the Penn Treebank and its recently developed "extension" (the so-called PropBank) to obtain such a deep representation of its sentences automatically, possibly without too much additional manual work.

The development of the Penn Treebank (PTB) (Marcus, Santorini, and Marcinkiewicz, 1993; Marcus et al., 1994) has had an immense effect on the development of natural language processing (NLP) by providing training and testing data for approaches based on machine learning, including statistical models. It has inspired other treebanking efforts in many languages, including the Prague Dependency Treebank (PDT) (Böhmová et al., 2001; Hajič et al., 2001). However, since the development of the PTB, it has become clear that for many NLP applications, parsing to a a level of representation is needed that is "deeper" than the surface-syntactic phrase structure of the PTB. Furthermore, work in generation using machine learning cannot use the PTB because the representation is too shallow as a starting point for generation. Thus, a more richly annotated corpus is needed, in particular, a corpus that includes certain semantic notions. Annotation efforts for languages other than English have been able to incorporate this requirement from the beginning. For example, the PDT includes both the Analytical and the Tectogrammatical level of representation. However, for English, such resources have been created only recently . One such resource is the PropBank (Kingsbury, Palmer, and Marcus, 2002b), which superimposes an annotation for verbal predicates and their arguments and adjuncts on the PTB. In the PropBank, the annotation of the relation

between verb and dependent is "local", i.e., only relevant to a single verb meaning. However, for many applications we need a "global" semantic labeling scheme such as that provided by the Tectogrammatical Representation (TR) of the PDT, with labels such as ACT (actor) and ORIG (origin) whose meaning is not specific to the verb. The question arises whether and how the PropBank can be extended to reflect global semantic information.

The direct motivation for this paper is the observation by Hajičová and Kučerová (2002) that the global semantics of the Tectogrammatical Representation (TR) of the Prague school cannot be derived directly from the local semantics of the PropBank, since it does not contain sufficient detail: TR makes distinctions not made in the PropBank. The authors suggest that it may, however, be derivable from the PropBank with the aid of an intermediary representation that also uses global semantic labels such as Lexical-Conceptual Structure (LCS), or VerbNet (VN). The proposal is worth investigating: it seems reasonable to derive TR labels from other representations of global semantics. While TR, LCS, and VN use different labels, we expect there to be some consistency. For example, LCS src should correspond to VerbNet Source and TR ORIG. While the three representations — TR, LCS, VN — are based on different approaches to representing the meaning of a sentence, all three approaches assume that there is a sharable semantic intuition about the meaning of the relation between a verb and each of its dependents (argument or adjunct). Of course, the semantic labels themselves differ (as in the case of src, Source, and ORIG), and furthermore, often one approach makes finer-grained distinctions than another, for example VN has one category Time, while TR has many subcategories, including THL (temporal length) and THWHEN (time point) and so on. Nonetheless, in these cases, the different label sets are compatible in meaning, in the sense that we can define a one-to-many mapping between label sets in the different frameworks. More precisely, we expect one of three situations to hold for a given pair of labels from label sets $\mathcal{A}$ and $\mathcal{B}$:

- A label a in $\mathcal{A}$ corresponds to exactly one label b in $\mathcal{B}$, and b corresponds only to a (bijective case).
- A label a in $\mathcal{A}$ corresponds to a set of labels B in $\mathcal{B}$, and each element b of B corresponds only to a (one-to-many case).
- A label b in $\mathcal{B}$ corresponds to a set of labels A in $\mathcal{A}$, and each element a of A corresponds only to b (many-to-one case).

The case in which there are overlapping meanings, with a_1 from $\mathcal{A}$ corresponding to b_1 and b_2 from $\mathcal{B}$, and a_2 from $\mathcal{A}$ corresponding to b_2 and b_3 from $\mathcal{B}$, should be excluded.

There are two positions one may take. Given that global semantic labels express relationships which are meaningful across verbs, and assuming that researchers in different frameworks share certain semantic intuitions, we may claim that labels are (possibly) compatible across frameworks. On the other hand, we may claim that in such difficult semantic issues, it is unreasonable to expect different frameworks to have converged on label sets with compatible meanings. The issue is not just one of academic interest — it is also of great practical interest. If the usefulness of parsing is to be increased by developing semantically annotated corpora (a very

costly process), it is important to know whether an annotation in, for example, LCS will allow us to automatically derive a corpus annotated in TR. If not, the value of a corpus of LCS labels will be reduced, since it will be relevant to a smaller community of researchers (those working in the framework of LCS). While to some researchers the answer to the question of inter-framework compatibility of labels may be intuitively obvious, we are not aware of any serious empirical study of this question. Such a study must necessarily be corpus-based or experimental, as only the data will reveal how the frameworks *use* their labels (as opposed to defining them), which is what this question is about.

In this paper, we present the results of investigating the relationship between PropBank, TR, LCS, and VN labels based on an annotated corpus. The conclusion we put forward is that global semantic labels are not only framework-specific, but also lexically idiosyncratic *within* each framework. This means that labels are not compatible between frameworks, and do not necessarily express the same semantic intuition. (It of course does not mean that these labels are used inconsistently within any one framework.) As a result, we argue that corpora should not be annotated in terms of global semantic labels (such as TR, LCS, or VN). Instead, we argue that corpora should be annotated with local semantic labels (as has already been done in the PropBank), and global semantic labels should be generated automatically using framework-specific lexicons (i.e., verb-specific lists of label mappings for arguments). Such lexicons represent an important resource in their own right.

This paper is structured as follows. We start by introducing a vocabulary to talk about types of resources in general in Section 5.2. We then present four different ways of labeling corpora with semantic information: PropBank in Section 5.3, TR in Section 5.4, VerbNet in Section 5.5, and LCS in Section 5.6.[1] While these approaches are part of larger theories of syntax and lexical semantics, we are for the purpose of this paper only interested in the label set they use to annotate the relation between a verbal predicate and its arguments and adjuncts; we will therefore refer to these theories in a reductive manner as "labeling schemes". We then compare the global-semantic labeling schemes to each other in Section 5.7 and find labeling to be lexically idiosyncratic and framework-specific. In Section 5.8 we return to the original question of Hajičová and Kučerová (2002) and report on experiments using machine learning to derive rule sets for annotating a corpus with TR labels. These results confirm the conclusions of Section 5.7.

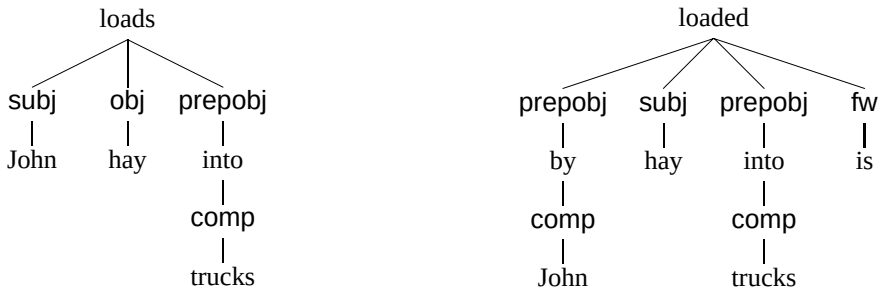
5.2 Types of Corpus Labels

Surface syntax reflects the relation between words at the surface level. Consider the following pair of sentences, whose structure is shown in Figure 5.1:

(1) a. John loads hay into trucks
    b. Hay is loaded into trucks by John

In this example, where two sentences differ only in the voice of the verb,

[1]These labeling schemes in themselves are not the original work presented in this paper, we summarize them here for the convenience of the reader. The original work is investigating the relation between and among them.



John loads hay into trucks

Hay is loaded into trucks by John

Figure 5.1: Surface syntactic representation for the sentences in (1)

John loads hay into trucks

Hay is loaded into trucks by Johnny

Figure 5.2: Deep-syntactic representation

the first two arguments of the verb, *John* and *hay*, have different roles depending on voice. The (dependency representation recoverable from the) PTB has a surface-syntactic labeling scheme, though deeper labels can be inferred from tags and traces.

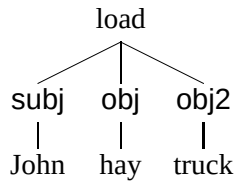
Deep syntax normalizes syntactically productive alternations (those that apply to all or a well-defined class of verbs, not lexically idiosyncratically). This primarily refers to voice, but (perhaps) also other transformations such as dative shift. The deep-syntactic representation for the two trees in Figure 5.1 (i.e., the two sentences in (1) is shown in Figure 5.2. However, the deep-syntactic representation does not capture verb-specific alternations, such as the container-content alternation found with *load*:

(2) a. John loads hay into trucks

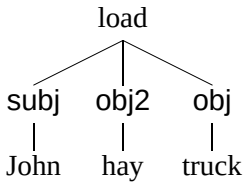
b. John loads trucks with hay

In these two sentences, the semantic relationship between the verb and its three arguments is the same in both sentences, but they are realized differently syntactically: *hay* is the deep direct object in one, *trucks* in the other. This is shown in the two trees in Figure 5.3.

Instead, we can choose numerical labels (arg0, arg1, ...) on the arguments which abstract away from the syntactic realization and only represent the semantic relation between the particular verb meaning and the arguments. These **local semantic** labels have no intrinsic meaning and are significant only when several



John loads hay into trucks



John loads trucks with hay

Figure 5.3: Deep-syntactic representation: missed generalization

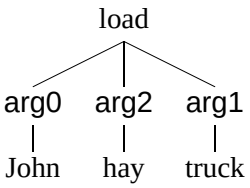


Figure 5.4: Local semantic representation for both (2a) and (2b)

syntactic realizations of the same verb meaning are contrasted. An example is shown in Figure 5.4.

Now consider the following two sentences:

(3) a. John loads hay into trucks

b. John throws hay into trucks

Semantically, one could claim that (3b) merely adds manner information to (3a), and that therefore the arguments should have the same relationships to the verb in the two cases. However, since these are different verbs (and *a fortiori* different verb meanings) there is no guarantee that the local semantic arc labels are the same. In a **global semantic** annotation, the arc labels do not reflect syntax at all, and are meaningful across verbs and verb meanings. The labels reflect generalizations about the types of relations that can exist between a verb and its argument, and the representation in Figure 5.5 applies to sentences (3a) and (3b).[2]

5.3 PropBank

The PropBank (Kingsbury, Palmer, and Marcus, 2002b) annotates the Penn Wall Street Journal Treebank II with dependency structures (or 'predicate-argument'

[2]The FrameNet project (Baker, Fillmore, and Lowe, 1998) uses semantic labels which are local, but apply not to one verb meaning, but to a set of verb meanings that refer to the same frame (i.e., situation). For example, *buy*, *sell*, *cost* and so on all refer to the commercial transaction frame, realizing different participants of the frame in different syntactic ways. However, since the frame elements such as Buyer or Rate (=price) do not refer to an abstract notion of the relationship between a proposition and its argument, but rather to a specific set of verbs and a specific argument, the approach is closer in spirit to a local semantic approach. Perhaps a better term for FrameNet would be "regional semantics".

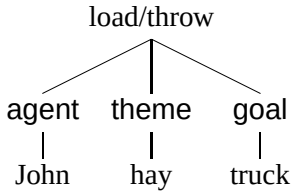


Figure 5.5: Global semantic representation for (3a) (with *load*) and (3b) (with*throw*); the labels used are for illustrative purposes

structures), using sense tags for each word and local semantic labels for each argument and adjunct. The argument labels are numbered and used consistently across syntactic alternations for the same verb meaning, as shown in Figure 5.4. Adjuncts are given special tags such as TMP (for temporal), or LOC (for locatives) derived from the original annotation of the Penn Treebank. In addition to the annotated corpus, PropBank provides a lexicon which lists, for each meaning of each annotated verb, its *roleset*, i.e., the possible arguments in the predicate and their labels. An example, the entry for the verb *kick*, is given in Figure 5.6. The notion of "meaning" used is fairly coarse-grained, and it is typically motivated from differing syntactic behavior. Since each verb meaning corresponds to exactly one roleset, these terms are often used interchangeably. The roleset also includes a "descriptor" field which is intended for use during annotation and as documentation, but which does not have any theoretical standing. Each entry also includes examples. Currently there are frames for about 1600 verbs in the corpus, with a total of 2402 rolesets.

ID	kick.01	
Name	drive or impel with the foot	
VN/Levin classes	11.4-2, 17.1, 18.1, 23.2 40.3.2, 49	
Roles	Number	Description
	0	Kicker
	1	Thing kicked
	2	Instrument (defaults to foot)
Example	[John]$_i$ tried [*trace*$_i$]$_{\mathrm{ARG0}}$ to kick [the football]$_{\mathrm{ARG1}}$	

Figure 5.6: The unique roleset for *kick*

5.4 Tectogrammatical Representation

The Tectogrammatical Representation (TR) of the Prague School (Sgall, Hajičová, and Panevová, 1986) is a dependency representation that contains only autosemantic (=meaning-bearing) words. The arcs are labeled with rich set of labels. What distinguishes TR from other labeling schemes is that it is hybrid: the deep subject and deep object of a verb are always given the labels ACT (for actor) and PAT (for patient), respectively. The deep indirect object is given one of three la-

PropBank		TR	
Role	Description	Form	label
0	Bidder	subject	ACT
1	Target	*for*	EFF
		to	AIM
2	Amount bid	object	PAT

Figure 5.7: TR extension to PropBank entry for *bid*, roleset name "auction"

bels, EFF(ect), ADDR(essee), or ORIG(in). Other arguments and free adjuncts are drawn from list of 42 global semantic labels, such as AIM, BEN(eficiary), LOC(ation), MAN(ner), and a large number of temporal adjuncts such as THL (temporal length) and THWHEN (time point).

For the TR, we have a small gold standard. Approximately 1,500 sentences of the PTB were annotated with TR dependency structure and arc labels. A total of 36 different labels are used in this corpus. The sentences were prepared automatically by a computer program (Žabokrtský and Kučerová, 2002) and then corrected manually. We will refer to this corpus as the TRGS (which should not be confused with the PDT, which is a much larger corpus in Czech), and to the code as AutoTR. It uses heuristics that can access the full PTB annotation.

In addition, there is a lexicon of tectogrammatical entries for English based on (a subset of) the PropBank lexicon. The mapping was done for 662 predicates (all PropBank entries that were done by January 2002). Every entry contains an original PropBank lexical information with examples, information about Levin class membership and appropriate tectogrammatical mapping. The mapping is only defined for entries that are explicitly listed in the original PropBank entry; no others were created. Figure 5.7 shows the entry for the verb *bid*. Note that the mapping to TR is indexed both on the Propbank argument and on the syntactic realization ("form"), so that arg1 may become EFF or AIM, depending on the preposition that it is realized with.

We evaluated the quality of the PropBank-to-TR lexicon by comparing results on those arguments in the TRGS whose verbs are also in the lexicon (727 argument instances). The AutoTR program has an error rate of 15.3% on this data, while the lexicon's error rate is only 12.2%. We performed an error analysis by randomly sampling 15 instances (of the 89 errors). In 9 instances, there were inconsistencies between the lexicon and the TRGS. (Of these, one instance was furthermore inconsistent in the TRGS.) In four instances, there appeared to be an error in the lexicon. And in two instances, there was an error in our automatic alignment of the data due to a mismatch of the syntactic analysis in the TRGS and in the PTB. We conclude that all these problems are in principle fixable.

5.5 VerbNet

VerbNet (Kipper, Dang, and Palmer, 2000) is a hierarchical verb lexicon with syntactic and semantic information for English verbs, using Levin verb classes (Levin, 1993) to systematically construct lexical entries. The first level in the hierarchy is constituted by the original Levin classes, with each class subsequently refined to

Actor, Agent, Theme, Patient, Asset, Attribute, Beneficiary, Cause, Destination, Experiencer, Instrument, Location, Material, Patient, Product, Recipient, Source, Stimulus, Time, Topic

Figure 5.8: Inventory of thematic role labels used in VerbNet

account for further semantic and syntactic differences within a class. Each node in the hierarchy is characterized extensionally by its set of verbs, and intensionally by a list of the arguments of those verbs and syntactic and semantic information about the verbs. The argument list consists of thematic labels from a set of 20 possible such labels (given in Fig. 5.8), and possibly selectional restrictions on the arguments expressed using binary predicates. The syntactic information maps the list of thematic arguments to deep-syntactic arguments. The semantic information for the verbs is expressed as a set (i.e., conjunction) of semantic predicates, such as *motion, contact, transfer_info*.[3] Currently, all Levin verb classes have been assigned thematic roles and syntactic frames, and 123 classes, with more than 2500 verbs, are completely described, including their semantic predicates.

In addition, a PropBank-to-VerbNet lexicon maps the rolesets of PropBank to VerbNet classes, and also the PropBank argument labels in the rolesets to VerbNet thematic role labels. Fig. 5.9 shows an example of the mapping of roleset *install.01* with VerbNet class *put-9.1*. The mapping is currently not complete: some verb meanings in PropBank have not yet been mapped, others are mapped to several VerbNet classes as the PropBank verb meanings are sometimes coarser than or simply different from the VerbNet verb meanings (many PropBank rolesets are based on a financial corpus and have a very specific meaning).

PropBank		VN
Role	Description	label
0	Putter	Agent
1	Thing put	Theme
2	Where put	Destination
VerbNet-Levin class		9.1

Figure 5.9: Entry in PropBank-to-VerbNet lexicon for *put* (excerpt)

Using this lexicon, we have augmented the PropBank-annotated Penn Treebank with VerbNet annotations automatically. In theory, we could simply look up the corresponding VerbNet argument for each annotated PropBank argument in the corpus. However, there are several impediments to doing this. First, the PropBank annotation of the Penn Treebank does not currently include the roleset, i.e., the verb meaning: of all the PropBank-annotated verbs in the TRGS, in only 74.7% of cases do we have access to the PropBank meaning (roleset). Second, because the PropBank-to-VerbNet lexicon is not yet complete (as just described), only 42.1% of verbs (instances) have exactly one VerbNet-Levin class assigned

[3] Both for VerbNet and LCS, the semantic information about each verb is not directly germane to this paper.

Verb	jog
Class	51.3.2.a.ii
Theta	_th,src(),goal()

Figure 5.10: LCS definitions of *jog* (excerpt)

to them. Therefore, only 46.1% of argument instances can be assigned VerbNet thematic roles automatically (18 different labels are used) However, the coverage will increase as (i) PropBank annotates rolesets in the corpus and (ii) the annotation of the PropBank lexicon with VerbNet information progresses. In principle, there is no reason why we cannot achieve a near 100% automatic coverage of the hand-annotated PropBank arguments in the Penn Treebank with VerbNet thematic roles.

5.6 Lexical Conceptual Structure

Lexical Conceptual Structure (LCS) is a compositional abstraction with language-independent properties that transcend structural idiosyncrasies (Jackendoff, 1983; Dorr and Olsen, 1997). LCS captures the semantics of a lexical item through a combination of semantic structure (specified by the shape of the graph and its structural primitives and fields) and semantic content (specified through constants). The semantic structure of a verb is something the verb inherits from its Levin verb class, whereas the content comes from the specific verb itself.

The lexicon entry for one sense of the English verb *jog* is shown in Figure 5.10. This entry includes several pieces of information such as the word's semantic verb class, its thematic roles ("Theta" – in this case, th, src, and goal), and the LCS itself (not shown here, as it is not directly relevant to this paper). The LCS specifies how the arguments — identified by their thematic roles — contribute to the meaning of the verb.

Figure 5.11 contains a list of thematic roles. The theta-role specification indicates the obligatory and optional roles by an underscore (_) and a comma (,), respectively. The roles are ordered in a canonical order normalized for voice (and dative shift): subject; object; indirect object; etc, which corresponds to surface order in English. Thus, the _th_loc grid is not the same as the _loc_th grid (*The box holds the ball* as opposed to *The water fills the box*).

agent, theme, experiencer, information, src (source), goal, perceived item, pred (identificational predicate), locational predicate, mod-poss (possessed item modifier), mod-pred (property modifier)

Figure 5.11: Inventory of LCS thematic roles (extract)

To derive LCS thematic labels for arguments and adjuncts in the PropBank, we make use of the Lexical Verb Database (LVD). This resource contains hand-constructed LCSs organized into semantic classes — a reformulated version of

Predict	From	No mlex	With mlex	n
VN	LCS	30.3%	13.0%	399
LCS	VN	22.8%	9.5%	399
TR	VN	36.1%	14.4%	97
VN	TR	56.7%	8.3%	97
TR	LCS	42.3%	20.5%	78
LCS	TR	41.0%	11.5%	78

Figure 5.12: Error rates for predicting one label set from another, with and without using feature mlex (the governing verb's lexeme); n is the number of tokens for the study

the semantic classes in (Levin, 1993). The LVD contains 4432 verbs in 492 classes with more specific numbering than the original Levin numbering (e.g., "51.3.2.a.ii"), a total of 11000 verb entries. For the mapping, we used as keys into the LVD both the lexeme and the Levin class as determined by VerbNet (see Section 5.5), adjusting the class name to account for the different extensions developed by Verbnet and LCS. Each key returns a set of possible theta grids for each lookup. We then form the intersection of the two sets, to get at the theta grid for the verb in its specific meaning. If this intersection is empty, we instead form the union. (This complex approach maximizes coverage.) We then map to each argument a set of possible theta roles (note that even if there are two possible theta grids, one of the arguments may receive the same role under both). This approach yields 54.7% of verb instances in the TRGS with a unique theta-grid, and 47.7% of argument/adjunct instances, with a unique theta role. (The lower figure is presumably due to the fact that verbs with fewer arguments are more likely to have unique theta grids.) A total of 13 LCS roles are used for these instances.

VN label	TR label	tokens	types	sample verbs
Topic	EFF	29	2	say X
Predicate	EFF	12	7	view Y as X
	AIM	2	2	use Y to do X
	CPR	1	1	rank Y as X
	COMPL	1	1	believe Y that X
	LOC	1	1	engage Y in X
Attribute	EFF	4	3	rate Y X
	EXT	1	1	last X
	THL	1	1	last X
	DIFF	1	1	fall X
	LOC	1	1	price Y at X

Figure 5.13: Exhaustive mapping of three VerbNet labels to TR labels other than ACT and PAT (the argument being labeled is *X*)

5.7 Relation Between Semantic Labels

We now address the question of how similar the three annotation schemes are, i.e., the semantic part of TR, LCS, and VerbNet. To test the correspondence between global semantic labels, we use Ripper (Cohen, 1996) to predict one label set, given another. Using a set of attributes, Ripper greedily learns rule sets that choose one of several classes for each data set. Because in this section we are using Ripper to analyze the data, not to actually learn rule sets to apply to unseen data (as we do in Section 5.8), we report here the error rate on the training data.

For these experiments, we use all arguments from the TRGS which are also labeled in the PropBank, 1268 data points. For VN and LCS, we exclude all data points in which either the predictor label or the predicted label are not available from the mappings described in Sections 5.4, 5.5, and 5.6, respectively. In the case of TR (which is always available), we exclude cases with the ACT and PAT features, as they are determined syntactically. If there is a one-to-one correspondence between two label sets, we expect a zero error rate for both directions; if the correspondence is one-to-many (i.e., one label set is more detailed than the other), we expect a zero error rate for at least one direction.

Instead, what we find are error rates between 22.8% and 56.7%, for all directions. Crucially, we find these error rates greatly reduced (with error reduction ranging between 51% and 85%) if we also allow the lexeme of the governing verb to be a feature. The results are summarized in Figure 5.12. All differences are significant, using the usual Ripper test (the difference between the results must be larger than twice the sum of each run's standard deviation). As expected, in each pair, the richer label set (as measured by the number of labels used in the TRGS) is better at predicting the less rich label set.

By way of illustration, we will look in more detail at the way in which three VN labels, Topic, Predicate, and Attribute, map to TR categories. The data is summarized in Figure 5.13.[4] As we can see, for all three labels, the most common TR label (and in the case of Topic, the only TR label) is EFF. However, closer inspection reveals this not to be the case. VerbNet makes a distinction between the communicated content (*John said he is happy*) which is a Topic, a Predicate of another dependent of the verb (*they view/portray/describe the sales force as a critical asset*, where *a critical asset* is a predicate true of the sales force), and an Attribute of another actant of the verb (*they value/estimate the order at $326 million/rate the bond AAA*).[5] TR considers all these cases to be EFFects of an act of communication or judgment. Conversely, TR makes a distinction between an EFFect of a human action (of communication or judgment, such *they value/estimate the order at $326 million/rate the bond AAA*) and different types of states of affairs, for example a DIFFerence (*UAL stock has fallen 33%*) or a length of time (THL, *the earth quake lasted 15 seconds*). To VN, these are all Attributes.

But note that in nearly all cases considered in the table in Figure 5.13, the governing verb determines the label assignment both for TR and VN.[6] Thus, both

[4] We exclude tokens whose TR labels are ACT or PAT, as these labels are determined entirely syntactically.

[5] Intuitively, a *predicate* is a function from entities to truth values, while an *attribute* is a function from entities to an open set of possible values (such as dollar amounts).

[6] The exceptions are in TR: *use Y to do X* is sometimes EFF, sometimes AIM, while *last X* is

in the general Ripper experiments and in these specific examples, we see that there is no general mapping among the labels; instead, we must take the governing verb into account. We conclude that assigning labels is both framework specific and lexically idiosyncratic within each framework.

```
Final hypothesis is:
ORIG if fw=from and vn!=_ (2/1).
CAUS if fw=because (2/0).
COND if fw=if (3/0).
MOD if lemma=probably (2/0).
DIR3 if pb=ARG2 and pba=DIR (2/0).
AIM if fw=to and vrole=adj (12/4).
MANN if pba=MNR (20/1).
ADDR if pb=ARG2 and vn=Recipient and
  lemma!=blame and lemma!=article (7/0).
ADDR if lemma=audience (2/1).
ADDR if mlemma=assure and pb=ARG1 (2/0).
TWHEN if pba=TMP (55/6).
EFF if vn=Topic and mlemma=say (25/0).
EFF if vrole='2' and fw=as (12/1).
ACT if vrole='0' (366/16).
default PATC (502/67).
```

Figure 5.14: Sample generated rule set (excerpt — "fw" is the function word for the argument, "mlex" the governing verb's lexeme, "pba" the modifier tag from the Penn Treebank)

5.8 Predicting TR Labels

We now turn to experiments for learning rule sets for choosing TR labels from all other labels (the task described by Hajičová and Kučerová (2002), the original inspiration for this work). We again use Ripper, as in Section 5.7. The task is to predict the TR label, and we experiment with different feature sets. Given our analysis in Section 5.7, we predict that using other global semantic labels, i.e., VN or LCS, will *not* improve performance. However, we expect syntactic (including lexical) features and local semantic features (PropBank) to contribute to performance. We observe that it is not clear what the topline is, given some inconsistency in the gold standard; the experience reported above from very small hand-inspected data sets suggests an inconsistency rate of between 5% and 10%.

We use the following syntactic features: **PTB lean** (argument lemma, governing verb's lemma, part-of-speech, and function word, if any); **Full PTB** (PTB lean + TR label of mother, extended tag of PTB, node labels of path to root); **VRole** (the deep-syntactic argument, as derived from the PTB by head percolation and voice normalization); and **AutoTR**, the computer script AutoTR writtem to determine TR labels. We also use these semantic features: PropBank, LCS, VerbNet. A sample rule set (with features PTB-lean, Vrole, Propbank, and VerbNet) is shown in Figure 5.14. The rules are checked from top to bottom, when one applies the listed

sometimes EXTent, sometimes temporal length (THL). We assume these are labeling inconsistencies.

label is chosen. The numbers in parentheses indicate the number of times the rule applies correctly (before the slash) and incorrectly (after the slash). Clearly, there is some overfitting happening in this particular ruleset (for example, in the rule to choose ADDR if the lemma is *audience*).

The results for the machine learning experiments are summarized in Figure 5.15. These are based on five-fold cross-validation on a set of 1268 data points (those arguments of the TRGS labeled by PropBank, with mismatches related to different syntactic treatment of conjunction removed). Note that because of the greedy nature of Ripper, a superset of features may (and often does) produce worse results than a subset. In general, any two results are statistically significant if their difference is between three and five; there are too many combinations to list all. Compared to the baseline of the hand-written AutoTR code, the combination of PTB Lean, Vrole, and PropBank provides an error reduction of 24.5% with respect to a (possibly unrealistic) 0% error topline. The error reduction is 75.8% with respect to default baseline of always choosing PAT, the most common label (i.e., running Ripper with no features), and the 0% topline.

Semantics Syntax	None	PropBank	PB&LCS	PB&VN
None	59.23%	24.30%	23.27%	22.25%
Vrole	30.44%	19.80%	18.38%	17.75%
PTB	18.15%	15.70%	16.17%	16.02%
PTB & Vrole	16.09%	15.14%	15.46%	14.67%
PTB Lean & Vrole	16.80%	14.36%	15.15%	14.51%

Figure 5.15: Results (error rate) for different combinations of syntactic features (left column) and semantic features (top row); baseline error rate using hand-written AutoTR code is 19.01%

We now highlight some important conclusions (all are statistically significant unless otherwise stated). First, some syntax always helps, whether or not we have semantics (compare the rows labeled "None" and any of the rows below it). This is not surprising, as some of the TR labels (ACT and PAT) are defined fully syntactically. Second, the PTB-lean feature set does as well as the full PTB set, no matter what semantic information is used (compare rows labeled "PTB & Vrole" and "PTB Lean & Vrole"). In particular, the TR label of mother, the extended tag of the PTB, and the node labels of path to root do not help. Third, using the PropBank improves on using just syntactic information (compare the columns labeled "None" and "PropBank" — not all pairwise comparisons are statistically significant). Fourth, as predicted, there is no benefit to adding global semantic information once local semantic information is used (compare the column labeled "PropBank" to the columns labeled "PB&LCS" and "PB&VN").

In related work, Gildea and Jurafsky (2002) predict generic FrameNet labels (similar to the VN or LCS labels). They achieve an error rate of 17.9% using no other semantic information. While this error rate is similar to the ones we report here (in the row labeled "None"), there are some important differences: their testing data only contains seen predicates (unlike ours), but our task is facilitated by the fact that the most common labels in TR are defined syntactically.

5.9 Conclusions

As we have seen, there are problems in mapping among VerbNet, LCS, and TR. Most truly global semantic labels are both framework-specific and lexically idiosyncratic: different frameworks (and possibly researchers in the same framework) do not divide up the space of possible labels in the same way. As a result, in automatically labeling a corpus with TR labels, using LCS or VerbNet does not improve on using only syntactic (including lexical) and local semantic information, contrary to the suggestion of Hajičová and Kučerová (2002). While this may at first seem like an unfortunate conclusion, we note that the solution seems to be fairly simple: the creation of lexicons. Lexicons are useful (even crucial) for consistent annotation, they are general repositories of linguistic knowledge, and they can be used for many NLP tasks. Thus the creation of lexicons along with a single set of annotations is a simple way to allow for translation to other annotation frameworks, since the lexical idiosyncracies are taken into account in the lexicon. For example, if we have a PropBank-style annotation for our corpus, and a (framework-specific, lexically idiosyncratic) PropBank-to-T lexicon, where T is the desired labeling scheme, then we can automatically relabel the corpus with the labels of T. Human intervention will only be required when T makes finer distinctions in verb or argument meaning than the scheme used for the annotation of the corpus. This approach can also be used when T represents a very domain- or task-specific labeling, in which case annotating a whole corpus just with these labels would be a very large investment with little prospect for resuse, as the labels would probably not be reusable by other projects.

Acknowledgments

This work originated as part of a Johns Hopkins University summer workshop on machine translation. We would like to thank Jan Hajič and his team for help and support. We would also like to thank Matthias Trautner Kromann for suggesting the use of machine learning in comparing the global labeling schemes. Palmer, Kipper, and Rambow were supported by grants NSF-ISLE 5-35505 and DARPA N66001-00-1-8915 to the University of Pennsylvania. Kučerová was supported by the Czech Ministry of Education project LN00A063 and ME642. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

This chapter will appear as an article in the Prague Bulletin of Mathematical Linguistics, published by Charles University, as a separate publication in 2003.

Chapter 6

Evaluation

Terry Koo and Jan Hajič

6.1 Overview

We evaluated our systems with IBM's BLEU evaluation metric. Because BLEU scores are a relative measure, we also created baseline and upper bound systems to use as reference points. Additionally, we created a competitor system using GIZA++, so that we could compare our performance to that of a good word-to-word system.

The remainder of this chapter describes, in order, the BLEU evaluation metric, the baseline system, the upper bound system, the GIZA++ system, the evaluation mechanics, and the results.

6.2 BLEU

We used IBM's BLEU metric because of its convenience and accuracy. BLEU is automatic, so it does not require expensive and time-consuming human evaluations. Additionally, BLEU has been shown to correlate with the judgments of humans (Papineni et al., 2001).

We obtained a Perl implementation of the baseline BLEU metric, as described by (Papineni et al., 2001). The following paragraphs describe the operation of the baseline BLEU metric as we have used it.

BLEU has two inputs: the candidate translation and a set of reference translations. First, the candidate and reference translations are broken into sentences. Each sentence is then processed into sets of 1-grams, 2-grams, 3-grams and 4-grams. Each of the candidate sentence's n-grams are checked for inclusion in the union of the reference sentences' n-grams, and matching n-grams are tallied. To guard against overly repeated n-grams (i.e. "the the the the the"), a given n-gram is prevented from matching more times than the maximum number of times it appears in any of the reference sentences. For each level of n-gram, the number of matching n-grams in the entire candidate translation is divided by the total number of n-grams in the

candidate translation. This yields four numbers which are the *modified n-gram precision* for 1-grams through 4-grams.

With the above scheme, shorter sentences can get higher modified n-gram precision scores than longer ones, since the number of times an n-gram can match is bounded. At the extreme, one can imagine a sentence composed of a single 4-gram that matches the reference; this would get perfect scores for all n-gram levels. To counteract the advantage of short sentences, BLEU uses a *brevity penalty*. The brevity penalty is a decaying exponential in c/r, where c is the length of the candidate translation in words, and r is the length of a best-matched reference translation. Specifically, r is the sum of the lengths of the reference sentences which are closest in length to their corresponding candidate sentence. The brevity penalty multiplied by the geometric mean of the four modified n-gram precision scores gives the final BLEU score.

We made one modification to the BLEU implementation, which was to add sentence ID's. Initially, we had trouble with dropped sentences because the original Perl code assumed the sentences in the candidate and reference translations were in the same order. A dropped sentence would change the candidate translation's order and cause an incorrect evaluation. Our modification allows BLEU to use sentence ID's to match sentences in the candidate and reference translations, rather than depending on a rigid ordering.

The number and variety of reference translations used affects the BLEU score and its accuracy. There may be many ways to correctly translate a given sentence, but the BLEU metric will only acknowledge the words and phrasing found in the reference translations. The more reference translations there are, and the more variety there is among them, the better the BLEU score can recognize valid translations.

In our own evaluations, we used 5 reference translations. Each reference contained roughly 500 matching sentences selected from WSJ sections 22, 23, and 24. Of the five references, one was the original Penn Treebank sentences and the other four were translations to English of a Czech translation of the Penn Treebank, done by four separate humans. We feel this gives us a good level of coverage in our reference translations.

6.3 Upper Bound System

The upper bound for the performance of our system would be translation by humans. Accordingly, our upper bound comparison system is composed of the references translations themselves.

To evaluate the references, we held out each reference in turn and evaluated it against the remaining four, averaging the five BLEU scores at the end. For the purposes of a meaningful comparison, all of the results we present were created using the same 5-way averaging.

6.4 Baseline System

Each of the four Inputs is associated with its own baseline. Each baseline, however, operates off the same principle: output the TR lemmas of each input TR tree in some order.

For Inputs 1 and 3, the TR lemmas are output in a randomized order. These Inputs are automatically generated from surface text and the TR trees capture the true surface word ordering. However, our generation system for these Inputs views the input TR trees as unordered. Thus the baseline is similarly deprived of word order information.

For Inputs 2 and 4, the TR lemmas are output in the order they appear in the TR tree. These inputs are, respectively, the manually annotated and Czech transfer TR trees. Their word orderings are meaningful; the manually annotated TR captures the deep word order while the Czech transfer TR captures the Czech word ordering. Our generation system can make use of this input word ordering. Therefore, the baseline should also be able to take advantage of this.

6.5 GIZA++ System

The GIZA++ system was created by Jan Cuřín and trained on 30 million words of bilingual text. This system is a representative of the word-to-word machine translation systems with which our own system will compete.

6.6 Evaluation Mechanics

Our 500 reference sentences were split into two test sets: a devtest set which we used to evaluate our system during its development, and an evaltest set which remained untouched until its use in the final evaluation at the conclusion of the workshop.

The evaluation itself is orchestrated by a number of `sh` and Perl scripts. The scripts allowed individual evaluations to be run as well as batch evaluations, and created a HTML webpage and GIF bar graph as output.

6.7 Results

Figures 6.1 and 6.2 display the final evaluation results in chart and graph form. In each of these, “Base N” is the baseline for Input N, and “Gen N” is the generation

	Base 1	Gen 1	Base 2	Gen 2	Upper
BLEU	0.04184	0.24416	0.16022	0.2365	0.53366
4-Gram	0.01146	0.10286	0.08038	0.09894	0.3385
3-Gram	0.02608	0.20238	0.17814	0.2016	0.46396
2-Gram	0.07564	0.41364	0.37008	0.41038	0.62766
1-Gram	0.76004	0.88762	0.7609	0.88116	0.8441
Brev	0.64902	0.82612	0.63484	0.8117	0.99396
	Base 3	**Gen 3**	**Base 4**	**Gen 4**	**GIZA++**
BLEU	0.04142	0.24324	0.05862	0.0479	0.18954
4-Gram	0.01146	0.10152	0.0181	0.00594	0.06884
3-Gram	0.026	0.20078	0.0408	0.02722	0.1365
2-Gram	0.07194	0.4132	0.12028	0.13192	0.28172
1-Gram	0.76194	0.8872	0.48054	0.6013	0.62328
Brev	0.65042	0.82758	0.7252	0.80064	0.9402

Figure 6.1: Final evaluation results for all systems.

system for Input N. The BLEU scores as well as the four modified n-gram precision scores and the brevity penalties are displayed.

The system as it currently stands can outperform the baselines for Inputs 1, 2, and 3, but it is still below the baseline for Input 4, the full MT. It is also well below the GIZA++ system's performance.

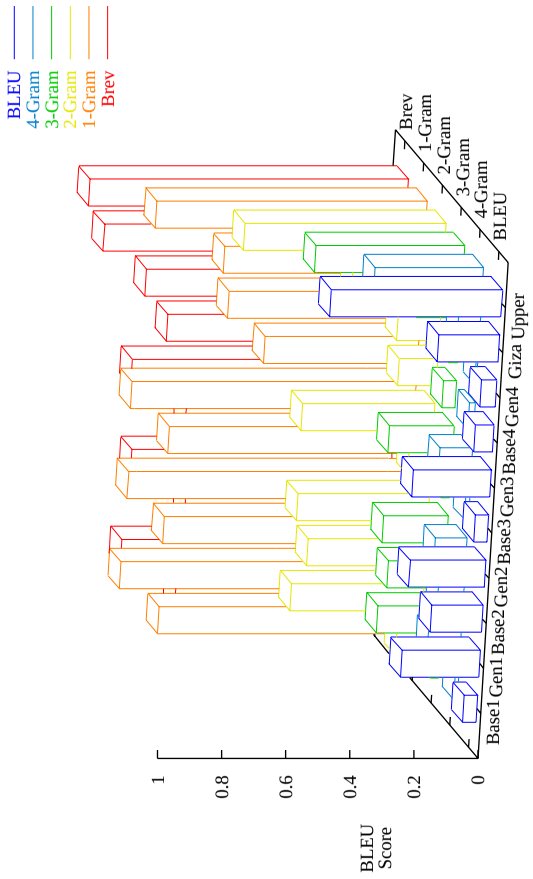


Figure 6.2: Graph of final evaluation results for all systems.

Chapter 7

Conclusions

Jan Hajič

As has already been said, the results as described in the previous section are worse than those obtained by a GIZA++ system based on lemmas. Base on the adequacy argument, I am still convinced, however, that deep linguistic analysis of the source language (and the subsequent relatively complex NL Generation it entails) is the right approach. During the 6-week workshop, we were unable to finish completely the implementation of the full system, and we had to use certain amount of "hacks" to get a fully automatic generation and thus translation, contributing to the overall low performance of the system.

Nevertheless, on top of bringing together a great team of people seeing the problem from different perspectives, there are already some tangible results of the Generation/MT team at this workshop:

- A methodology for automatically learnable nonisomorphic tree-to-tree transformations, the Synchronous Tree to Tree Transducers based on the STSG (Synchronous Tree Substitution Grammar);
- Resources (data) and tools for Czech/English Machine Translation (a CD with resources and tools rich enough to produce a fully trained statistical Czech/English MT system will be published soon by the LDC); it includes, e.g., over 25,000 manually aligned sentences of the WSJ part of the PTB and their translations into Czech;
- Deeply semantically annotated version of Penn Treebank (by an automatic rule-based system) using the so-called tectogrammatical representation of sentence meaning (will also be availabel on the above CD);
- A dataset for BLEU- or NIST-based evaluation of a Czech-to-English MT system (500 sentences in 5 reference versions), suitable also for an English NL generation system evaluation when using a deep semantic represenation of sentence meaning as a starting point;
- A suggestion to evaluate NL generation systems by the same methodology (automatically evaluatable metric such as the BLEU or NIST system);

We hope that these achievements will help to advance the field of Machine Translation and NL Generation, and we certianly hope that the methodology developed at the Workshop will eventually lead to results superior to the current state-of-the-art statistical MT systems.

Chapter 8

References

Al-Onaizan, Yaser, Jan Cuřín, Michael Jahr, Kevin Knight, John Lafferty, Dan Melamed, Franz-Josef Och, David Purdy, Noah A. Smith, and David Yarowsky. 1999. The statistical machine translation. Technical report, Johns Hopkins University. NLP WS'99 Final Report.

Alshawi, H., S. Bangalore, and S. Douglas. 2000. Learning dependency translation models as collections of finite state head transducers. In *Computational Linguistics*, volume 26(1), pages 45–60.

Baker, Collin F., J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet project. pages 86–90, Montréal.

Böhmová, Alena. 2001. Automatic procedures in tectogrammatical tagging. *The Prague Bulletin of Mathematical Linguistics*, 76.

Böhmová, Alena, Jan Hajič, Eva Hajičová, and Barbora Hladká. 2001. The Prague Dependency Treebank: Three-Level Annotation Scenario. In Anne Abeillé, editor, *Treebanks: Building and Using Syntactically Annotated Corpora*. Kluwer Academic Publishers.

Cohen, William. 1996. Learning trees and rules with set-valued features. In *Fourteenth Conference of the American Association of Artificial Intelligence*. AAAI.

Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the ACL*, pages 16–23, Madrid, Spain.

Collins, Michael John. 1999. *Head-driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

Cuřín, Jan and Martin Čmejrek. 1999. Automatic extraction of terminological translation lexicon from czech-english parallel texts. In *Proceedings of 4th TELRI-II Seminar*, Bratislava, Slovakia, November.

Cuřín, Jan and Martin Čmejrek. 2001. Automatic extraction of terminological translation lexicon from czech-english parallel texts. *International Journal of Corpus Linguistics*, 6(Special Issue):1–12, December.

Dorr, Bonnie and Mari Broman Olsen. 1997. Deriving verbal and compositional lexical aspect for NLP applications. In *"Proceedings of the 35th Annual Meeting of the ACL"*, pages 151–158, Madrid, Spain.

Eisner, Jason. 2001. *Smoothing a Probabilistic Lexicon via Syntactic Transformations*. Ph.D. thesis, University of Pennsylvania, July.

Eisner, Jason. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (Companion Volume)*, Sapporo, July.

Elhadad, Michael. 1991. FUF: The universal unifier - user manual, version 5.0. Technical Report CUCS-038-91, Columbia University.

Elhadad, Michael. 1993. *Using argumentation to control lexical choice: a unification-based implementation*. Ph.D. thesis, Computer Science Department, Columbia University.

Feiner, Steven and Kathleen McKeown. 1991. Automating the generation of coordinated multimedia explanations. *IEEE Computer*, 24(10):33–41, October.

Gildea, Daniel and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. 28(3):245–288.

Goodman, Joshua. 1999. Semiring parsing. In *Computational Linguistics*, volume 25(4), pages 573–605.

Goodman, Joshua. 2002. Efficient parsing of dop with pcfg-reductions. In *Rens Bod, Khalil Sima'an, and Remko Scha, editors: Data Oriented Parsing. CSLI*.

Hajič, Jan, Eva Hajičová, Martin Holub, Petr Pajas, Petr Sgall, Barbora Vidová-Hladká, and Veronika Řezníčková. 2001. The current status of the prague dependency treebank. In *LNAI 2166*, LNAI 2166. Berlin, Heidelberg, New York, pages 11–20.

Hajičová, Eva and Ivona Kučerová. 2002. Argument/valency structure in PropBank, LCS database and Prague Dependency Treebank: A comparative study. In *Proceedings of LREC*.

Hajič, Jan, Eric Brill, Michael Collins, Barbora Hladká, Douglas Jones, Cynthia Kuo, Lance Ramshaw, Oren Schwartz, Christopher Tillmann, and Daniel Zeman. 1998. Core Natural Language Processing Technology Applicable to Multiple Languages. Technical Report Research Note 37, Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD.

Hajič, Jan and Barbora Hladká. 1998. Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset. In *Proceedings of COLING-ACL Conference*, pages 483–490, Montreal, Canada.

Hajič, Jan, Jarmila Panevová, Eva Buráňová, Zdeňka Urešová, Alla Bémová, Jan Štěpánek, Petr Pajas, and Jiří Kárník. 2001. A Manual for Analytic Layer Tagging of the Prague Dependency Treebank. Technical Report TR-2001-, ÚFAL MFF UK, Prague, Czech Republic. English translation of the original Czech version.

Hajičová, Eva, Jarmila Panevová, and Petr Sgall. 2000. A Manual for Tectogrammatic Tagging of the Prague Dependency Treebank. Technical Report TR-2000-09, ÚFAL MFF UK, Prague, Czech Republic. in Czech.

Jackendoff, Ray. 1983. *Semantics and Cognition*. MIT Press, Cambridge, MA.

Kingsbury, Paul, Martha Palmer, and Mitch Marcus. 2002a. Adding semantic annotation to the penn treebank. In *In Proceedings of the Human Language Technology Conference*, San Diego, California.

Kingsbury, Paul, Martha Palmer, and Mitch Marcus. 2002b. Adding semantic annotation to the Penn TreeBank. In *Proceedings of the Human Language Technology Conference*, San Diego, CA.

Kipper, Karin, Hoa Trang Dang, and Martha Palmer. 2000. Class-based construction of a verb lexicon. In *"Proceedings of the 17th AAAI"*, pages 151–158, Austin, TX, July 30 - August 3 2000.

Knight, Kevin. 1999. Decoding complexity in word-replacement translation models. In *Computational Linguistics*, volume 25(4).

Knight, Kevin and Daniel Marcu. 2000. Statistics-based summarization—step 1: Sentence compression. In *Proc. AAAI*.

Langkilde, I. 2000. Forest-based statistical sentence generation. In *Proceedings of NAACL'00*, Seattle, WA.

Langkilde, Irene and Kevin Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Conference of the Association for Computational Linguistics (ACL-98)*, pages 704–710.

Levin, Beth. 1993. *English Verb Classes and Alternations: A Preliminary Investigation*. The University of Chicago Press.

Marcus, M., G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 1994. The Penn Treebank: Annotating predicate argument structure. In *Proceedings of the ARPA Human Language Technology Workshop*.

Marcus, Mitchell M., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19.2:313–330, June.

Minnen, G, John Caroll, and D Pearce. 2001. Applied morphological processing of English. *Natural Language Engineering*, 7.3:207–223.

Minnen, G., J. Carroll, and D. Pearce. 2001. Applied morphological processing of english. *Natural Language Engineering*, 7(3):207–223.

Och, F., C. Tillmann, and H. Ney. 1999. Improved alignment models for statistical machine translation. In *Proceedings of EMNLP*.

Och, F. J. and H. Ney. 2000. Improved statistical alignment models. In *Proc. of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 440–447, Hongkong, China, October.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2001. Ibm research report: Bleu: a method for automatic evaluation of machine translation. In *Technical Report RC22176 (W0109-022)*, IBM Research Division, Yorktown Heights, New York.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318.

Poutsma, A. 2000. Data-oriented translation. In *Proceedings of COLING'00*.

Radev, Dragomir R. and Kathleen R. McKeown. 1998. Generating Natural Language Summaries from Multiple On-Line Sources. *Computational Linguistics*, 4:469–500, September.

Ratnaparkhi, Adwait. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142, University of Pennsylvania, May. ACL.

Robin, Jacques. 1994. *Revision-Based Generation of Natural Language Summaries Providing Historical Background*. Ph.D. thesis, Computer Science Department, Columbia University.

Sgall, P., E. Hajičová, and J. Panevová. 1986. *The meaning of the sentence and its semantic and pragmatic aspects*. Reidel, Dordrecht.

Shieber, Stuart and Yves Schabes. 1990. Synchronous tree adjoining grammars. In *Proceedings of COLING 1990*.

Žabokrtský, Zdeněk and Ivona Kučerová. 2002. Transforming penn treebank phrase trees into (praguian) tectogrammatical dependency trees. *The Prague Bulletin of Mathematical Linguistics*, 78.

Žabokrtský, Zdeněk, Petr Sgall, and Sašo Džeroski. 2002. Machine learning approach to automatic functor assignment in the prague dependency treebank. In *Proceedings of LREC 2002 (Third International Conference on Language Resources and Evaluation)*, volume V, pages 1513–1520, Las Palmas de Gran Canaria, Spain.

Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. In *Computational Linguistics*, volume 23(3).

Yamada, Kenji and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of ACL'01*.