PBML

logo

The Prague Bulletin of Mathematical Linguistics

NUMBER ??? JANUARY 2009 1–10

An Open Source Rule Induction Tool for Transfer-Based SMT

Yvette Graham, Josef van Genabith

Abstract

In this paper we describe an open source tool for automatic induction of transfer rules. Transfer rule induction is carried out on pairs of dependency structures and their node alignment to produce all rules consistent with the node alignment. We describe an efficient algorithm for rule induction and give a detailed description of how to use the tool.

1. Introduction

Statistical Machine Translation (SMT) using deep linguistic representations for transfer is a relatively new and growing research area (Bojar and Hajič, 2008, Graham, 2008, Bojar et al., 2007a, Bojar and Čmejrek, 2007b, Graham et al., 2007, Ding and Palmer, 2006, Riezler and Maxwell, 2006, Ding and Palmer, 2005, Ding and Palmer, 2004a, Ding and Palmer, 2004b, Cmejrek et al., 2003, Eisner, 2003, Ding and Palmer, 2003, Hajič et al., 2002, Alshawi et. al., 2000a, Alshawi et. al., 2000b, Alshawi et. al., 1998). Training requires highly efficient algorithms; the training data is hierarchical dependency graphs as opposed to surface form stings and is therefore more complex than training data used for other methods of SMT (Brown et al., 1993); large numbers of rules (that contain a lot of morphological information) are needed to achieve high coverage of unseen data and rich statistical information. We provide a tool that uses an efficient algorithm for rule induction and outputs the rules in efficient O(n) size data structures (Graham, 2008). The rule induction tool, the parser/generator engine XLE [1] and transfer decoder [2] constitute a complete Transfer-Based SMT system.

[1] "XLE is available to a limited number of researchers ... For more information about obtaining a license, please contact thking at parc.com."

[2] The transfer decoder is currently in development and will be released as an open source tool in the near future.

Please cite this article as: Yvette Graham, Josef van Genabith, An Open Source Rule Induction Tool for Transfer-Based SMT. The Prague Bulletin of Mathematical Linguistics No. ???, 2009, 1–10.

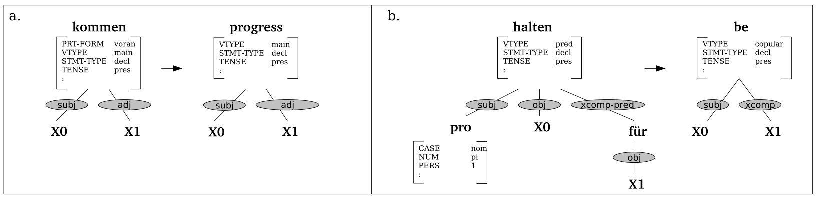


Figure 1. Example Transfer Rules

2. Transfer-Based SMT

Transfer-Based SMT is composed of three parts, i) parsing to linguistic structure, ii) transfer from SL linguistic structure to TL linguistic structure and iii) generation of TL sentence. Each step uses a statistical model to select the best output. For parsing, a disambiguation model is used to rank the parses and select the n-best output parses. Decoding (transfer) is then done on a parse structure (or n-best list of parse structures) via beam-search producing an n-best list of TL structures. For generation, the input is an n-best list of TL structures and all possible TL sentences are produced. The best TL sentence is then selected using an n-gram language model.

3. Training

The aim of rule induction is to acquire rules useful for transferring unseen SL structures by capturing correspondences between example training structure pairs of a parsed bilingual corpus. Figure 1(a) [3] shows a rule that captures an isomorphic German-English correspondence, while Figure 1(b) [4] captures a non-isomorphic correpsondence. Lexicalized nodes contain a vector of feature value pairs storing the morphological information belonging to that node.

3.1. Transfer Rule Induction

The transfer rule induction algorithm takes as input i) a dependency structure pair and ii) a one-to-one set of alignments between nodes of the dependency structure pair. Any alignment method can be used to align the nodes. For example, we currently

[3] The LHS contains a single lexicalized node with predicate value *(voran)kommen* with two arguments, a *subject* (X_0) and an *adjunct* (X_1) and the the RHS has the same structure but with *progress* as the root node.

[4] The LHS has the German predicate *halten* as its root node with three arguments, a *subject* (the lexicalized node with predicate *pro*), an *object* (X_0) and an *xcomplement-predicate* (the lexicalized node with predicate *für* and *object* X_1) and the RHS of the rule has *be* as its root node with two arguments, a *subject* (X_0) and an *xcomplement* (X_1).

use Giza++ (Och et al., 1999) for node alignment by constructing a bilingual training corpus from the predicate lemmas of the dependency structure nodes of the parsed bitext. Word alignment can then be run, in both language directions, as in Phrase-Based SMT (Koehn et al., 2003), followed by symmetrisation of the word alignments. We currently use the intersection of the word alignments as this gives a reliable set of one-to-one alignments.

3.1.1. Consistent Transfer Rules

As in Phrase-Based SMT, where a word alignment for each example sentence pair is first established before phrases consistent with that word alignment are extracted (Och et al., 1999, Koehn et al., 2003), we induce transfer rules that are consistent with the node alignment. We define a consistent transfer rule using a simplification of the actual training dependency structures and temporarily consider them as tree structures by ignoring edges that cause cycles in the graph or edges that share an end node with another edge. [5] Definition 1 applied to a (simplified) dependency structure pair yields a set of rules containing no variables by constraining rule induction using both the alignments between nodes and the position of the nodes within the two structures:

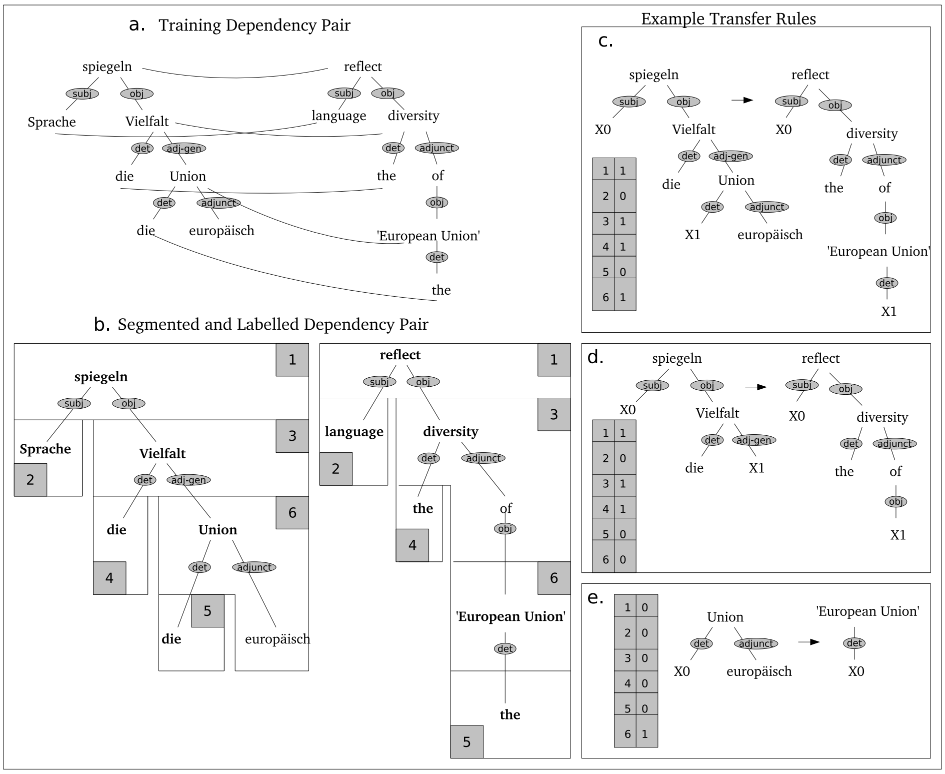
Definition 1.
Given a one-to-one set of alignments A between nodes in dependency pair (F, E), $(\overline{f}, \overline{e})$ is a rule consisting of nodes (N_f, N_e), rooted at (r_f, r_e), with descendents (D_f, D_e) of r_f and r_e in F and E respectively, if

$$\begin{array}{ll} & N_f = r_f \cup D_f \\ \bigwedge & N_e = r_e \cup D_e \\ \bigwedge & \forall f_i \in N_f : (f_i, e_j) \in A \rightarrow e_j \in N_e \\ \bigwedge & \forall e_j \in N_e : (f_i, e_j) \in A \rightarrow f_i \in N_f \\ \bigwedge & \exists e_j \in N_e : (r_f, e_j) \in A \\ \bigwedge & \exists f_i \in N_f : (f_i, r_e) \in A \end{array}$$

Definition 2.
For any rule $(\overline{f}, \overline{e})$ in dependency pair (F, E) rooted at (r_f, r_e) consisting of nodes N_f and N_e, where $(\overline{s}, \overline{t})$ is also a rule in (F, E) rooted at (r_s, r_t) consisting of nodes N_s and N_t where $r_s \neq r_f$, $r_t \neq r_e$, if $r_s \in N_f$ and $r_t \in N_e$ then there is a rule $(\overline{a}, \overline{b})$ rooted at (r_f, r_e) with nodes r_s and r_t replaced by variable x_k, where k is an index unique to the transfer rule, consisting of nodes:

$$\begin{array}{l} N_a : N_f \backslash N_s \cup x_k \\ N_b : N_e \backslash N_t \cup x_k \end{array}$$

[5] For example, if the subject of node A is also the subject of node B, one of these these edges is ignored temporarily. This is done by labelling the nodes using an increasing index in depth first order (only labelling each node once). Edges with an end node label less than their start node are ignored.



c.

1	1
2	0
3	1
4	1
5	0
6	1

d.

1	1
2	0
3	1
4	1
5	0
6	0

e.

1	0
2	0
3	0
4	0
5	0
6	1

Figure 2. Consistent Transfer Rules

To help visualize what is considered a consistent transfer rule, Figure 2(b) shows the example dependency structure of Figure 2(a) divided into parts by a number of boxes with corresponding parts of the dependency structure pair labelled with the numbers 1-6. Each consistent transfer rule can be realized by assigning a true or false value to each pair of boxes, so that boxes assigned true are included in the rule and boxes assigned false are left out. Combinations of true and false values are constrained and this can be visualized by only allowing adjoining boxes in Figure 2(b) to be labelled true for any one rule. Figures 2(c), 2(d) and 2(e) show consistent rules with truth value combinations. According to Definition 1, two nodes may form the root of a transfer rule if they both have the same non-empty set of aligned descendents (rule root nodes are in bold in Figure 2(b)). This ensures that the entire subtree rooted at the SL root of a transfer rule corresponds to the entire subtree rooted at the TL root. In addition, the root nodes of a rule must each be an aligned node, i.e. each must be aligned to some node in the other side of the rule. This ensures that unaligned nodes do not form the root of a transfer rule and are, therefore, only included in rules that also contain their head node, giving them a meaningful context. For example, the rule in Figure 2(e) is allowed by Definition 1, but a rule with this LHS and a RHS rooted at *of* is disallowed. The node *of* does not have a lexical correspondent in the German structure, but instead its head *diversity*, the dependency label *adjunct* and node *of* together correspond to *Vielfalt* and the dependency label *adjunct-gen* (genative adjunct). Definition 2 describes how rules with variables are produced by replacing both sides of a nested rule, i.e. a rule nested inside a larger rule, with a variable X_n. For example, in Figure 2, rule (d) was produced from rule (c) by replacing the nested rule (e) with variable X_1. Introducing variables to transfer rules in this way potentially produces a very large number of rules. In the worst case, when we have isomorphic structures with all nodes aligned, the number of rules is exponential. However, in reality, dependency structures of real world parallel corpora are very rarely entirely isomorphic with a very high number of aligned nodes. If the number of rules induced does indeed become unmanageable, however, rule induction can be constrained further by putting a limit on the size of rules. [6]

3.2. Transfer Rule Induction Algorithm

The transfer rule induction algorithm works by encoding all consistent rules of the SL and TL dependency structure pair in a single structure. The most complex part of the algorithm decides which node pairs within the dependency structure pair form *rule roots*. Once the *rule roots* of the dependency structure pair have been decided, the entire set of SL and TL dependency triples are then simply recorded with each dependency triple slightly modified by labelling it with a *context variable* (A_i) (Maxwell and Kaplan, 1991) and by replacing the original node labels with variables (X_i). Labelling the dependency triples with *context variables* allows us to encode all rules consistent with the node alignment in a single structure. This method of encoding allows $O(2^n)$ rules to be encoded in an $O(n)$ size structure and is described in detail in Graham and van Genabith (2008). The algorithm for choosing the *rule root* nodes of the dependency structure pair is given in Figure 3. [7] The complexity of the algorithm is $O(a^2 \log a)$ in the worst case, where a is the number of aligned node

[6] We currently do not limit the size of rules as we train on a restricted sentence length of 5 to 15 words. This results in an average number of rules induced per sentence pair of 32.47, with average sentence length 9.89 for German and 10.48 for English.

[7] The program itself is written in Prolog, and uses some of Prolog's built-in features that are not available in other programming languages. To keep the pseudocode as implementation independent as possible, when we use a Prolog specific function or control structure we describe it in pseudocode using an equivalent function or control structure available in most programming languages. For example, Prolog has a built in indexing of terms, that uses the first argument of the term as a key to achieve an O(log n) return time when searching for that term in memory. We use this in our Prolog implementation but where we do so we described it in pseudocode as using a hash table.

pairs.

4. Using the Rule Induction Tool

The rule induction tool requires a Prolog engine to run. The system has been developed and tested with SICStus Prolog. Included in the download package are 4000 German-English sample dependency structure pairs from a portion of the Europarl Corpus (Koehn et al., 2005). [8] The sample dependency structures are divided into sets, each containing 1000 sentences.

1. Download the package from http://www.computing.dcu.ie/~ygraham/software.html
2. Unzip ria.tar.gz and extract the files. This should produce a folder called *ria*.
3. Create an environment variable called RIA and set it to the location of the tool, for example: RIA=/home/jsmith/ria; export RIA;
4. Add the location of the bin directory to your PATH environment variable, for example: PATH=$PATH:$RIA/bin; export PATH;
5. Test the tool by typing the following command to induce all rules from set 0 of the sample training sentences: ria 0

A file containing the rules of set 0 should now be written in the following directory: $RIA/output/rules.

4.1. Output Format

The rules induced from each training dependency pair are stored in a file containing the rules (in a single compact size structure), and a file with some additional information about the rules. For example, the following two files store rules for sentence 123 of set 0:

- $RIA/output/rules/sents_0000/R123.pl
- $RIA/output/rules/sents_0000/I123.pl

To retrieve the rules, both of these files should be loaded by SICStus before calling the following predicate:

- transfer_rule(+-S, +-T, +-Fs_id, +-Root_id, -LHS, -RHS, +-Options).

We include an example program [9] that retrieves all rules for a specified sentence pair and records them:

[8] The sample sentences were parsed with XLE parse engine (Kaplan et al., 2002) to Lexical Functional Grammar (Kaplan and Bresnan, 1982, Bresnan, 2001, Dalrymple, 2001) f-structures using German and English grammars (Riezler et al., 2002, Butt et al., 2002). For node alignment, Giza++ (Och et al., 1999) was run in both language directions and the intersection was gotten using moses (Koehn et al., 2007).

[9] The rules should be retrieved by loading the rule and information files and then calling findall on transfer_rules/7, as is done in the example program, as it is not necessary to enumerate all rules on disk to use them.

```
Algorithm RuleRoots( List sl_nodes, List tl_nodes,
                     HashTable <sl_node_id,alignment_id> sl_alignments,
                     HashTable <tl_node_id,alignment_id> tl_alignments):

  # For each aligned SL node create a list
  # containing the alignment ids of its aligned descendents
  # Put lists in a Hash Table
  sl_aligned_descs = new HashTable<list_of_aligned_descs,sl_node_id>
  foreach  s ∈ S
    if exists sl_alignments.get(s.node_id) then
      list = new empty list
      foreach d ∈ descendents(s)
        if exists sl_alignments.get( d.node_id) then
          a_id = sl_alignment.get( d.node_id)
          list.add( a_id)
      sl_aligned_descs.put( list, s)

  # Likewise for TL nodes
  tl_aligned_descs = new HashTable<list_of_aligned_descs,tl_node_id>
  foreach  t ∈ T
    if exists tl_alignments.get(t.node_id) then
      list = new empty list
      foreach d ∈ descendents(t)
        if exists tl_alignments.get( d.node_id) then
          a_id = tl_alignment.get( d.node_id)
          list.add( a_id)
      tl_aligned_descs.put( list, t)

  # Find node pairs with matching sets of aligned descendents
  roots = new empty List
  foreach key in keys( sl_aligned_desc)
    if exists tl_aligned_descs.get( key)
      # A pair has been found
      i = sl_aligned_descs.get( key)
      j = tl_aligned_descs.get( key)
      roots.add( i, j)
  return roots
```

Figure 3. Algorithm to choose the rule roots *in the SL and TL dependency structures*

- write_rules 123

The enumerated rules for sentence id 123 should now be written in:

- $RIA/output/example/R123

4.2. Running the Tool on New Training Data

Start by converting the dependency structures into the same Prolog format of the sample structures.[10] The training structures should be divided into sets of 1000 and put in directories in the following locations:

- $RIA/data/sl_train
- $RIA/data/tl_train
- $RIA/data/alignments

For example, SL and TL dependency structures for sentence id 134067 should be put in two separate files;

- $RIA/data/sl_train/sents_0134/S067.pl and
- $RIA/data/tl_train/sents_0134/S067.pl,

and node alignments in a file called

- $RIA/data/alignments/sents_0134/A067.pl.

These files can then be archived and zipped. Name them as follows:

- $RIA/data/sl_train/sents_0134.tar.gz
- $RIA/data/tl_train/sents_0134.tar.gz
- $RIA/data/alignments/sents_0134.tar.gz

Rule induction can then be run for this set of structures using the command:

- ria 134

5. Conclusion

We presented an open-source tool for automatic transfer rule induction from parsed bilingual corpora. We described an efficient algorithm that induces transfer rules from dependency structure pairs encoding rules in an efficient O(n) size data structure (Graham, 2008). We hope that this tool is used to produce interesting research.

Acknowledgments

Many thanks to John Maxwell, Joachim Wagner, Marcus Furlong, Mary Hearne, Philipp Koehn and our reviewers.

Bibliography

Hiyan Alshawi, Srinivas Bangalore and Shona Douglas. 1998. Automatic Acquisition of Hierarchical Transduction Models of Machine Translation. In *Proceedings of COLING-*

[10] The sample structures are in the output format of the XLE parse engine (for further details see http://www2.parc.com/isl/groups/nltt/xle/doc/xle.html#Prolog_Output).

ACL 1998.

Hiyan Alshawi, Srinivas Bangalore and Shona Douglas. 2000. Learing Dependency Translation Models as Collections of Finite State Head Transducers. In *Computational Linguistics*, 26(1):45-60.

Hiyan Alshawi and Shona Douglas. 2000. Learning Dependency Transduction Models from Unannotated Examples. In *Philisophical Transactions A*, The Royal Society 358:1357-1372.

Ondřej Bojar, Silvie Cinkova and Jan Ptaček. Towards English-to-Czech MT via Tectogrammatical Layer. In *Proceedings of the Sixth International Workshop on Treebanks and Linguistic Theories (TLT 2007)*, Bergen, Norway, December. 2007

Ondřej Bojar and Martin Čmejrek. 2007. Mathematical Model of Tree Transformations. *Project Euromatrix Deliverable 3.2*, Ufal, Charles University, Prague.

Ondřej Bojar and Jan Hajič. 2008. Phrase-Based and Deep Syntactic English-to-Czech Statistical Machine Translation. In *Proceedings of the third Workshop on Statistical Machine Translation*, Columbus, Ohio, June 2008.

Joan Bresnan. 2001. *Lexical-Functional Syntax.*, Blackweelm Oxford, 2001.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra and Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2): 263-311.

Miriam Butt, Helge Dyvik, Tracy H. King, Hiroshi Masuichi and Christian Rohrer. 2002. The Parallel Grammar Project. (grammar version 2005) In *Proceedings of the 19th International Conference on Computational Linguistics (COLING'02), Workshop on Grammar Engineering and Evaluation*, pages 1-7. Tapei, ROC.

Martin Cmejrek, Jan CurÃŋn and JirÃŋ Havelka 2003 Czech-English Dependency Tree-based Machine Translation In *Proceedings of EACL 2003*, pages 83-90.

Mary Dalrymple. *Lexical Functional Grammar*, Academic Press, San Diego, CA; London. 2001.

Yuan Ding, Daniel Gildea and Martha Palmer. 2003. An Algorithm for Word-Level Alignment of Parallel Dependency Trees. In *Proceedings of the Machine Translation Summit 2003.*

Yuan Ding and Martha Palmer. 2004. Automatic Learning of Parallel Dependency Treelet Pairs. In *Proceedings of the 1st International Joint Conference on Natural Language Processing.*

Yuan Ding and Martha Palmer. 2004. Synchronous Dependency Insertion Grammars A Grammar Formalism for Syntax-Based Statistical MT. In *Proceedings of COLING 2004.*

Yuan Ding and Martha Palmer. 2005. Machine Translation Using Probabilistic Synchronous Dependency Insertion Grammars. In *Proceedings of the 43rd Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 541-548, Ann Arbor, June 2005.

Yuan Ding and Martha Palmer. 2006. Better Learning and Decoding for Syntax Based SMt Using PSDIG. In *Proceedings of AMTA 2006*

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 205-208, Sappora, July 2003.

Yvette Graham, Deirdre Hogan and Josef van Genabith. 2007. Automatic Evaluation of Generation and Parsing for Machine Translation with Automatically Acquired Transfer Rules. In *Proceedings of the 2007 Workshop on Using Corpora for NLG: Language Generation and Machine Translation*, at MT Summit XI, Copenhagen, September 2007.

Yvette Graham and Josef van Genabith. 2008. Packed Rules for Automatic Transfer Rule Induction. In *Proceedings of the European Association of Machine Translation Conference 2008*, Hamburg, Germany.

Jan Hajič, Martin Čmejrek, Bonnie Dorr, Yuan Ding, Jason Eisner, Daniel Gildea, Terry Koo, Kristin Parton, Gerald Penn, Dragomir Radev and Owen Rambow. 2002. Natural Language Generation in the Context of Machine Translation. Technical Report. *NLP WS'02*, final report.

Ronald M. Kaplan, Tracy H. King and John T. Maxwell. 2002. Adapting existing grammars: the XLE experience. In *Proceedings of COLING 2002*, Taipei, Taiwan.

Ronald Kaplan and Joan Bresnan. 1982. Lexical Functional Grammar, a Formal System for Grammatical Represenation. In Bresnan, J. editor, *The Mental Representation of Grammatical Relations*, pages 173-281, MIT Press, Cambridge, MA.

Philipp Koehn, Franz Josef Och and Daniel Marcu. 2003. Statistical Phrase-based Translation. In *Proceedings of the HLT-NAACL 2003*, pages 48-54, Edmonton, May/June 2003.

Philipp Koehn. 2005. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Proceedings of MT Summit 2005*

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison Burch, Richard Zens, Alexandra Constantin, Marcello Federico, Nicola Bertoldi, Chris Dyer, Evan Herbst, Brooke Cowen, Wade Shen, Christine Moran and Ondřej Bojar. 2007. Moses: Open Source Toolkit for Statistical Machine Translation In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*

John T. Maxwell III and Ronald M. Kaplan. 1991. A Method for Disjunctive Constraint Satisfaction. In *Current Issues in Parsing Technology*, Masaru Tomita editor, pages 173-190, Kluwer Aca demic Publishers.

Franz Josef Och, Christoph Tillmann Hermann and Ney. 2000. Improved Alignment Models for Statistical Machine Translation. In *Proceedings of the 1999 Conference on Empirical Methods in Natural Language Processing* (EMNLP'99). College Park, MD, pages 20-28.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using Lexical Functional Grammar and discriminitive estimation techniques . (grammar version 2005) In *Proceedings of the 40th Annual Meeting of the Association of Computational Linguistics (ACL)*, Philadelphia, July 2002.

Stefan Riezler and John T. Maxwell III. 2006. Grammatical Machine Translation. In *Proceedings of HLT-ACL*, pages 248-255, New York.