PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 102 OCTOBER 2014 27–36

Tree Transduction Tools for cdec

Austin Matthews[a], Paul Baltescu[b], Phil Blunsom[b], Alon Lavie[a], Chris Dyer[a]

[a] Carnegie Mellon University
[b] University of Oxford

Abstract

We describe a collection of open source tools for learning tree-to-string and tree-to-tree transducers and the extensions to the cdec decoder that enable translation with these. Our modular, easy-to-extend tools extract rules from trees or forests aligned to strings and trees subject to different structural constraints. A fast, multithreaded implementation of the Cohn and Blunsom (2009) model for extracting compact tree-to-string rules is also included. The implementation of the tree composition algorithm used by cdec is described, and translation quality and decoding time results are presented. Our experimental results add to the body of evidence suggesting that tree transducers are a compelling option for translation, particularly when decoding speed and translation model size are important.

1. Tree to String Transducers

Tree-to-string transducers that define relations on strings and trees are a popular formalism for capturing translational equivalence where syntactic tree structures are available in either the source or target language (Graehl et al., 2008; Galley et al., 2004; Rounds, 1970; Thatcher, 1970). The tools described in this paper are a restricted version of top-down tree transducers that support multi-level tree fragments on one side and strings on the other, with no copying or deletion (Huang et al., 2006; Cohn and Blunsom, 2009). Such transducers can elegantly capture syntactic regularities in translation. For example see Fig. 1, which gives the rules necessary to translate between English (an SVO language with ditransitive verbs) and Welsh (a VSO language with prepositional datives). In our notation, transducers consist of a set of rules

Corresponding author: cdyer@cs.cmu.edu
Cite as: Austin Matthews, Paul Baltescu, Phil Blunsom, Alon Lavie, Chris Dyer. Tree Transduction Tools for cdec. The Prague Bulletin of Mathematical Linguistics No. 102, 2014, pp. 27–36.
doi: 10.2478/pralin-2014-0011.

$$\text{(S}\ \boxed{\text{NP}}\ \text{(VP}\ \boxed{\text{VBD}}\ \boxed{\text{NP}}\ \boxed{\text{NP}}\text{))} \leftrightarrow \boxed{2}\ \boxed{1}\ \boxed{4}\ i\ \boxed{3}$$

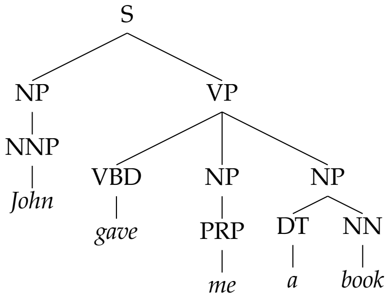
$$\text{(NP (DT } a\text{)}\ \boxed{\text{NN}}\ \text{)} \leftrightarrow \boxed{1}$$

$$\text{(VBD } \textit{gave}\text{)} \leftrightarrow \textit{rhoddodd}$$

$$\text{(NN } \textit{book}\text{))} \leftrightarrow \textit{lyfr}$$

$$\text{(NP (PRP } \textit{me}\text{))} \leftrightarrow \textit{mi}$$

$$\text{(NP (NNP John))} \leftrightarrow \textit{Ioan}$$



`cdec` text format of above transducer (with example features):

```
  (S [NP] (VP [VBD] [NP] [NP])) ||| [2] [1] [4] i [3] ||| logP(s|t)=-0.2471
  (NP (DT a) [NN]) ||| [1] ||| logP(s|t)=-0.6973 Delete_a=1
  (VBD gave) ||| rhoddodd ||| logP(s|t)=-2.3613
  (NN book) ||| lyfr ||| logP(s|t)=-0.971
  (NP (PRP me)) ||| mi ||| logP(s|t)=-1.3688
  (NP (NNP John)) ||| Ioan ||| logP(s|t)=0
```

`cdec` input text format of above tree:

```
  (S (NP (NNP John)) (VP (VBD gave) (NP (PRP me)) (NP (DT a) (NN book))))
```

Figure 1. Example single-state transducer that transduces between the SVIO English tree (upper right of figure) and its VSOP Welsh translation: rhoddodd Ioan lyfr i mi.

(also called edges) which pair a tree fragment in one language with a string of terminal symbols and variables in a second language. Frontier nonterminal nodes in the tree fragment are indicated with a box around the nonterminal symbol, and the corresponding substitution site in the string is indicated by a box around a number indexing the nonterminal variable in the tree fragment (counting in top-down, left to right, depth first order). Additionally, tree-to-string transducers can be further generalized so as to have multiple transducer states, shown in Fig. 2. The transducers in Fig. 1 can be understood to have a single state. For formal properties of tree-to-string transducers, we refer the reader to the above citations.

Tree-to-string transducers define a relation on strings and trees and, in translation applications, are capable of transforming either source trees (generated by a parser) into target language strings or source strings into target-language parse trees. Running the transducer in the tree-to-string direction can avail itself of specialized algorithms similar to finite state composition (§4); in the string-to-tree direction, they can be trivially converted to synchronous context free grammars and transduction can be carried out with standard CFG parsing algorithms (Galley et al., 2004).

$$q_0 : (\text{S}\ \boxed{\text{NP}}\ (\text{VP}\ \boxed{\text{VB}}\ \boxed{\text{NP}})) \leftrightarrow \boxed{1} : q_0\ \boxed{2} : q_0\ \boxed{3} : q_{acc}$$

$$q_0 : (\text{NP}\ (\text{DT}\ \textit{the})\ \boxed{\text{NN}}) \leftrightarrow \textit{der}\ \boxed{1} : q_0$$

$$q_{acc} : (\text{NP}\ (\text{DT}\ \textit{the})\ \boxed{\text{NN}}) \leftrightarrow \textit{den}\ \boxed{1} : q_0$$

$$q_0 : (\text{NN}\ \textit{dog}) \leftrightarrow \textit{Hund}$$

cdec text format of above transducer (with example features):

```
[Q0] ||| (S [NP] (VP [VB] [NP])) ||| [Q0,1] [Q0,2] [QACC,3] ||| lp=-2.9713
[Q0] ||| (NP (DT the) [NN]) ||| der [Q0,1] ||| lp=-1.3443
[QACC] ||| (NP (DT the) [NN]) ||| den [Q0,1] ||| lp=-2.9402
[Q0] ||| (NN dog) ||| Hund ||| lp=-0.3171
```

Figure 2. A tree-to-string transducer with multiple states encoding structural information for choosing the proper nominal inflection in English–German translation.

2. Heuristic Hypergraph-based Grammar Extraction

In this section we describe a general purpose tree-to-string and tree-to-tree rule learner. We will consider the tree-to-tree alignment problem in this case (the tree-to-string case is a straightforward simplification). Instead of extracting rules from a pair of aligned trees, rules from a pair of aligned *hypergraphs* (any tree can easily be transformed into an equivalent hypergraph; an example of such a hypergraph is shown in Fig. 3). By using hypergraphs, the rule extraction algorithm can use forest outputs from parsers to capture parse uncertainty; furthermore (as discussed below), it simplifies the rule extraction algorithm so that extraction events—even of so-called "composed rules" (Galley et al., 2006)—always apply locally to a single edge rather than considering larger structures. This yields a simpler implementation.

The rule extraction process finds pairs of aligned nodes in the hypergraphs based on the terminal symbol alignment. We will call a source node S and a target node T *node-aligned* if the following conditions hold. First, S and T must either both be non-terminals or both be terminals. Aligning a terminal to a non-terminal or vice-versa is disallowed. Second, there must be at least one alignment link from a terminal dominated by S to a terminal dominated by T. Third, there must be no alignment links from terminals dominated by S to terminals outside of T or vice-versa.

We define a "rule" to be pair of hyperedges whose heads are node-aligned *and* whose non-terminal children are node-aligned in a one-to-one (bijective) manner. For example, in the sample tree, we see that the source node $\texttt{PP}_{4,6}$ is node-aligned to the target node $\texttt{PP}_{5,7}$ *and* their children are node-aligned $\texttt{TO}_{4,5}$ to $\texttt{PREP}_{5,6}$ and $\texttt{NN}_{5,6}$ to $\texttt{NP}_{5,7}$. This edge pair corresponds to the rule `[PP::PP]` $\rightarrow$ `[TO,1] [NN,2] ||| [PREP,1] [NP,2]`. Note in this formalism, no edges headed by terminals, so we will not extract any rules with terminal "heads".

The above formulation allows the extraction of so-called “minimal” rules that do not contain internal structure, but it does not yet include any mechanism for extracting more complex rules. Rather than adding extra mechanisms to the rule extractor, we create extra edges in the hypergraph so that “composed” edges are available to the extractor. To do so, we recursively add edges from non-overlapping sets of descendent nodes. For example, one hyperedge added to the source side of the sample hypergraph pair is `VP`$_{3,6}$ $\rightarrow$ `walked TO`$_{4,5}$ NN$_{5,6}$. Independently, on the target side we add an edge `VP`$_{3,7}$ $\rightarrow$ `a marché PREP`$_{5,6}$ NP$_{5,7}$.

Now when we extract rules, we will find these two edges will give rise to the rule `[VP::VP]` $\rightarrow$ `walked [TO,1] [NN,2] ||| a marché [PREP,1] [NP,2]`, a composed rule not extractable by the bald algorithm.

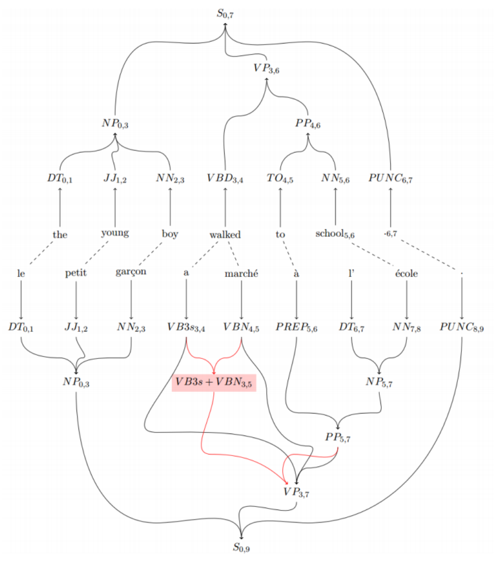


Figure 3. A pair of aligned hypergraphs. NP$_{0,3}$ represents an NP over the span $[0, 3)$. An example virtual node and its corresponding virtual edges is shown in red.

While using composed edges allows us to extract all permissible rules from a pair of aligned trees, to be consistent with previous work, we introduce one more type of hypergraph augmentation. Hanneman et al. (2011) allow for adjacent sibling non-terminal nodes to be merged into one *virtual node*, which may then be node-aligned to opposite nodes, be they “real” or virtual. To enable this, we explicitly add virtual nodes to our hypergraph and connect them to their children with a hyperedge. Furthermore, for every hyperedge that contained all of the sibling nodes as non-terminal tails, we add a duplicate hyperedge that uses the new virtual node instead.

For example, in Fig. 3, we have added a new non-terminal node labeled `VB3s+VBN`$_{3,5}$ to the hypergraph. This node represents the fusion of the `VB3s`$_{3,4}$ and `VBN`$_{4,5}$ nodes. We then add a hyperedge headed by the new `VB3s+VBN`$_{3,5}$ with tails to both `VB3s`$_{3,4}$ and `VBN`$_{4,5}$. Furthermore, we make a copy of the edge `VP`$_{3,7}$ $\rightarrow$ `VB3s`$_{3,4}$ `VBN`$_{4,5}$ `PP`$_{5,7}$, and replace the `VB3s`$_{3,4}$ and `VBN`$_{4,5}$ tail nodes with a single tail, `VB3s+VBN`$_{3,5}$, to form the new edge `VP`$_{3,7}$ $\rightarrow$ `VB3s+VBN`$_{3,5}$ `PP`$_{5,7}$. The addition of this new hyperedge allows the extraction of the rules `[VBD::VB3s+VBN]` $\rightarrow$ `walked ||| a marché` and `[VP::VP]` $\rightarrow$ `[VBD,1] [PP,2]`

`|||  [VB3s+VBN,1]  [PP,2]`, both of which were unextractable without the virtual node.

With the addition of virtual nodes, our work is directly comparable to Hanneman et al. (2011), while being more modular, extensible and provably correct. One particularly interesting extension our hypergraph formulation allows is the use of weighted parse *forests* rather than 1-best trees. This helps our rule extractor to overcome parser errors and allows us to easily handle cases of ambiguity, in which two or more trees may be equally likely for a given input sentence.

3. Bayesian Synchronous Tree to String Grammar Induction

Although HyperGrex, the tool described in the previous section, is flexible, it relies on heuristic word alignments that were generated without knowledge of the syntactic structure or the final translation formalism they will be used in. In this section, we present our open source implementation of the synchronous tree-to-string grammar induction algorithm proposed by Cohn and Blunsom (2009).[1] This model directly reasons about the most likely tree-to-string grammar that explains the parallel corpus. Tree-to-tree grammars are not currently supported.

The algorithm relies on a Bayesian model which incorporates a prior preference for learning small, generalizable STSG rules. The model is designed to jointly learn translation rules and word alignments. This is important for capturing long distance reordering phenomena, which might otherwise be poorly modeled if the rules are inferred using distance penalized alignments (e.g. as in the heuristic proposed by Galley et al. (2004) or the similar one used by HyperGrex).

The model represents the tree-to-string grammar as a set of distributions $\{G_c\}$ over the productions of each non-terminal c. Each distribution G_c is assumed to be generated by a Dirichlet Process with a concentration parameter α_c and a base distribution $P_0(\cdot \mid c)$, i.e. $G_c \sim DP(\alpha_c, P_0(\cdot \mid c))$. The concentration parameter α_c controls the model's tendency towards reusing rules or creating new ones according to the base distribution and has a direct influence on the size of the resulting grammar. The base distribution is defined to assign probabilities to an infinite set of rules. The probabilities decrease exponentially as the sizes of the rules increase, biasing the model towards learning smaller rules.

Instead of representing the distributions G_c explicitly, we integrate over all the possible values of G_c. We obtain the following formula for estimating the probability of a rule r with root c, given a fixed set of derivations $\mathbf{r}$ for the training corpus:

$$p(r \mid \mathbf{r}, c; \alpha_c, P_0) = \frac{n_r + \alpha_c P_0(r \mid c)}{n_c + \alpha_c}, \tag{1}$$

where n_r is the number of times r occurs in $\mathbf{r}$ and n_c is the number of rules with root c in $\mathbf{r}$.

[1]Our code is publicly available here: `https://github.com/pauldb89/worm`.

Cohn and Blunsom (2009) train their model using Gibbs sampling. To simplify the implementation, an alignment variable is defined for every internal node in the parsed corpus. An alignment variable specifies the interval of target words which are spanned by a source node. Alternatively, a node may not be aligned to any target words or may span a discontiguous group of words, in which case it is annotated with an empty interval. Non-empty alignment variables mark the substitution sites for the rules in a derivation of a parse tree. Overall, they are used to specify a set of sampled derivations $\mathbf{r}$ for the entire training data. Alignment spans are constrained to subsume the spans of their descendants and must be contained within the spans of their ancestors. In addition to this, sibling spans belonging to the frontier of the same rule must not overlap.

We implement Gibbs sampling with the help of two operators: `expand` and `swap`. The `expand` operator works by resampling a randomly selected alignment variable a, while keeping all the other alignment variables fixed. The set of possible outcomes consists of the empty interval and all the intervals assignable to a such that the previous conditions continue to hold. Each outcome is scored proportionally to the new rules it creates, using Equation 1, conditioned on all the rules in the training data that remain unaffected by the sampling operation. The `swap` operator randomly selects two frontier nodes labelled with non-terminals belonging to the same STSG rule and chooses to either swap their alignment variables or to leave them unchanged. The outcomes are weighted similarly to the previous case. The goal of the `swap` operator is to improve the sampler's ability to mix, especially in the context of improving word reordering, by providing a way to execute several low probability `expand` steps at once.

Our implementation of the grammar induction algorithm is written in `C++`. Compiling the code results in several binaries, including `sampler`, which implements our Gibbs sampler. Our tool takes as input a file containing the parse trees for the source side of the parallel corpus, the target side of the parallel corpus, the word alignments of the training data, and two translation tables giving $p(s \mid t)$ and $p(t \mid s)$ respectively. The word alignments are needed only to initialize the sampler with the first set of derivations (Galley et al., 2004). The remaining input arguments (hyperparameters, rule restrictions, etc.) are initialized with sensible default values. Running the binary with the `--help` option will produce the complete list of arguments and a brief explanation for each. The tool produces several files as output, including one containing the set of rules together with their probabilities, computed based on the last set of sampled derivations. The documentation released with our code shows how to prepare the training data, run the tool and convert the output to the `cdec` format.

Our tool leverages the benefits of a multithreaded environment to speed up grammar induction. At every sampler iteration, each training sentence is dynamically allocated to one of the available threads. In our implementation, we use a hash-based implementation of a Chinese Restaurant Process (CRP) (Teh, 2010) to efficiently compute the rule probabilities given by Equation 1. The data structure is updated whenever one of the expand or swap operators is applied. To lock this data structure with every update would completely cancel the effect of parallelization, as all the basic operations performed by the sampler are dependent on the CRP. Instead, we distribute a copy of the CRP on every thread and synchronize the data structures at the end of each iteration. Although the CRPs diverge during an iteration through the training data, no negative effects are observed when inferring STSGs in multithreaded mode.

4. Tree-to-string translation with cdec

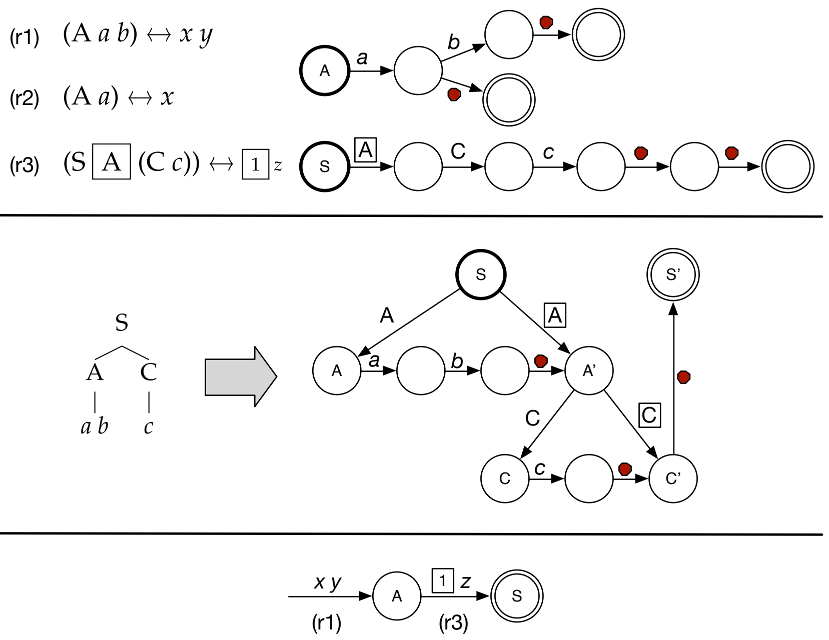


Figure 4. DFA representation of a tree transducer (above) and an input tree (middle). This transducer will transduce the input tree to the hypergraph (below) yielding a single string x y z*, using rules (r1) and (r3). Red octagons are closing parentheses.*

The cdec decoder (Dyer et al., 2010) has a modular decoder architecture that factors the decoding problem into multiple stages: first, a hypergraph is generated that represents the translation search space produced by composing the input (a string, lattice, or tree) with the relevant transducer (a synchronous context-free grammar, a finite state transducer, etc.); second, the hypergraph is rescored—and possibly restructured (in the case of adding an n-gram language model)—with generic feature extractors; finally, various outputs of interest are extracted (the best translation, the k-best translations, alignments to a target string, statistics required for parameter optimization, etc.).

The original cdec implementation contained hypergraph generators based on a variety of translation backends, although SCFG translation is the most widely used (Chiang, 2007). In this section, we describe the tree-to-string algorithm that generates a translation forest given a source language parse tree and a tree-to-string transducer.

The construction of the hypergraph takes place by composing the input tree with the tree-to-string transducer using a top down, recursive algorithm. Intuitively, the algorithm matches all rules in the transducer that start at a given node in the input tree and match at least one complete rewrite in the source tree. Any variables that were used in the match are then recursively processed until the entire input tree is completed. To make this process efficient, the tree side of the input transducer is determinized by depth-first, left-to-right factoring—this process is analogous to left factoring a context-free grammar (Klein and Manning, 2001). By representing the tree using the same depth-first, left-to-right representation, standard DFA intersection algorithms can be used to compute each step of the recursion. The DFA representation of a transducer (tree side) and an input tree (starting at nonterminal S) is shown in Fig. 4.

To understand how this algorithm proceeds on this particular input, the input tree DFA is matched against the 'S' DFA. The output transductions are stored in the final states of the transducer DFA, and for all final states in the transducer DFA that are reached in a final state of the input DFA, an edge is added to the output hypergraph, one per translation option. Variables that were used in the input DFA are then recursively processed, starting from the relevant transducer DFA (in this case since first (r3) will be used which has an A variable, then 'A' DFA will then be invoked recursively).

Extractor	k_s	k_t	Instances	Types
grex	1	1	24.9M	11.8M
HyperGrex	1	1	25.9M	12.7M
HyperGrex	1	10	33.3M	17.7M
HyperGrex	10	1	33.7M	17.9M
HyperGrex	10	10	48.7M	24.3M

Figure 5. Grammar sizes using different grammar extraction set ups. k_s (k_t) represents the number of source (target) trees used.

5. Experiments

We tested our tree-to-tree rule learner on the FBIS Chinese–English corpus (LDC2003E14), which consists of 302,966 sentence pairs or 9,350,506 words on the English side. We first obtain k-best parses for both sides of FBIS using the Berkeley Parser[2] and align the corpus using fastalign (Dyer et al., 2013). We use a 5-gram language model built

[2]https://code.google.com/p/berkeleyparser/

with KenLM on version four of the English GigaWord corpus plus the target side of FBIS, smoothed with improved Kneser-Ney smoothing. For each set up we extract rules using `grex` (Hanneman et al., 2011) or our new tool. When using our tool we have the option of simply using 1-best trees to compare directly to `grex`, or using the weighted forests consisting of all of Berkeley's k-best parses on the source side, the target side, or both. For these experiments we use $k = 10$. Each system is tuned on mt06 using Hypergraph MERT. We then test each system on both mt03 and mt06.

Details concerning the size of the extracted grammars can be found in Table 5.[3] Translation quality results are shown in Table 6.

5.1. Baysiean Grammar Experiments

Extractor	k_s	k_t	mt06	mt03	mt08
`grex`	1	1	29.6	31.8	23.4
`HyperGrex`	1	1	30.1	32.4	24.0
`HyperGrex`	1	10	**30.4**	**32.9**	**24.3**
`HyperGrex`	10	1	29.5	32.0	23.1
`HyperGrex`	10	10	30.0	32.7	23.7

Figure 6. BLEU results on mt06 (tuning set), mt03, and mt08 using various grammar extraction configurations.

The Bayesian grammar extractor we describe is constructed to find compact grammars that explain a parallel corpus. We briefly discuss the performance of these grammars in a tree-to-string translation task relative to a standard Hiero baseline. Each of these systems was tuned on mt03 and tested on mt08. Table. 7 summarizes the findings. Although tree-to-string system with minimal rules underperforms Hiero slightly, it uses orders of magnitude fewer rules—in fact the number of rules in the Hiero grammar *filtered* for the 691-sentence test set is twice as large as the Bayesian grammar. The unfiltered Hiero grammar is 2 orders of magnitude larger than the Bayesian grammar.

Extractor	iterations	rule count	mt08
Hiero	–	36.6M	27.9
HyperGrex minimal (§2)	–	1.4M	26.5
Bayes (§3)	100	0.77M	26.5
Bayes (§3)	1,000	0.74M	26.9

Figure 7. Comparing HyperGrex (minimal rules), the Bayesian extractor after different numbers of iterations, and Hiero.

[3] This indicates that `grex` failed to extract certain valid rules. This conclusion was validated by our team, and confirmed with the authors of (Hanneman et al., 2011).

Acknowledgements

This work was supported by the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant number W911NF-10-1-0533, the Qatar National Research Fund (a member of the Qatar Foundation) under grant NPRP 09-1140-1-177, by a Google Faculty Research Grant (2012_R2_10), and by the NSF-sponsored XSEDE program under grant TG-CCR110017.

Bibliography

Chiang, David. Hierarchical phrase-based translation. *Computational Linguistics*, 2007.

Cohn, Trevor and Phil Blunsom. A bayesian model of syntax-directed tree to string grammar induction. In *Proc. of EMNLP*, 2009.

Dyer, Chris, Adam Lopez, Juri Ganitkevitch, Johnathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proc. of ACL*, 2010.

Dyer, Chris, Victor Chahuneau, and Noah A. Smith. A simple, fast, and effective reparameterization of IBM model 2. In *Proc. of NAACL*, 2013.

Galley, Michel, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? In *HLT-NAACL*, 2004.

Galley, Michel, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. Scalable inference and training of context-rich syntactic translation models. In *Proc. of NAACL*, 2006.

Graehl, Jonathan, Kevin Knight, and Jonathan May. Training tree transducers. *Computational Linguistics*, 34(3), 2008.

Hanneman, Greg, Michelle Burroughs, and Alon Lavie. A general-purpose rule extractor for SCFG-based machine translation. In *Proc. of SSST*, 2011.

Huang, Liang, Kevin Knight, and Aravind Joshi. Statistical syntax-directed translation with extended domain of locality. In *Proc. of AMTA*, 2006.

Klein, Dan and Christopher D. Manning. Parsing and hypergraphs. In *Proc. of IWPT*, 2001.

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

Teh, Yee Whye. Dirichlet process. In *Encyclopedia of Machine Learning*, pages 280–287. 2010.

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

Address for correspondence:
Chris Dyer
cdyer@cs.cmu.edu
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, United States