14 Symbolic MT 4: Tree-based MT

In this chapter, we will cover one final symbolic method for translation: tree-based machine translation.

14.1 Motivation for Tree-based Methods

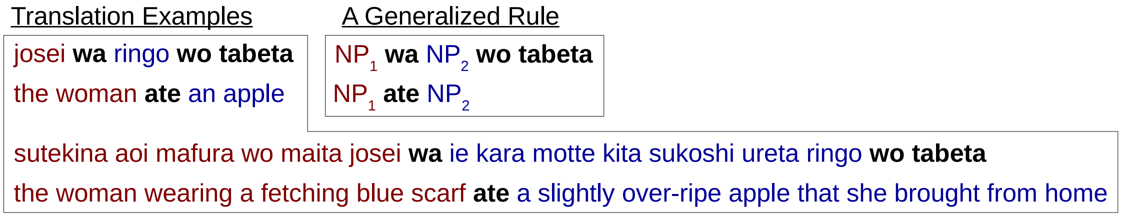
Translation Examples
josei wa ringo wo tabeta
the woman ate an apple
A Generalized Rule
NP1 wa NP2 wo tabeta
NP1 ate NP2
sutekina aoi mafura wo maita josei wa ie kara motte kita sukoshi ureta ringo wo tabeta
the woman wearing a fetching blue scarf ate a slightly over-ripe apple that she brought from home

Figure 46: An example of translations with varying complexity that can nonetheless be expressed using a general translation rule with embedded variables NP_1 and NP_2.

In the previous two chapters, we covered word-based methods and phrase-based methods, which memorize correspondences between words and contiguous phrases, respectively. However, there are still several very useful correspondences between the languages that are beyond the reach of these models.

Specifically, language has a hierarchical structure, where phrases can be arbitrarily simple, or complex. An example of this is shown in Figure 46. Here, we can see that there are two sentence pairs, the first one simple and the second one complex. However, both have the same underlying structure of "*noun phrase* ate *noun phrase*" in English, which is the translation equivalent of "*noun phrase* wa *noun phrase* wo tabeta" in Japanese.

If we can capture this structure in the rules that our model uses, then we can expect that it will be able to more accurately generalize to sentences with more complicated forms and improve the ability of the model to capture the reordering between phrases in complex sentences. One way to express this is through the generalized translation rule shown in the upper right corner of the Figure 46. This rule has words in each language, but also two variables NP_1 and NP_2, corresponding to the first and second verb phrase respectively, and we will discuss models that capture these types of rules throughout the remainder of this chapter. Before doing so, however, it is important to note that these rules cannot be captured in standard phrase-based translation models, as these models are restricted to handling contiguous phrases, with no capacity to model embedded variables.

14.2 Probabilistic Context-Free Grammars and Parsing

Before taking the step to using these types of rules in translation, we first need to take a step back and focus on the task of **syntactic parsing**, capturing the underlying syntactic structure of a sentence. While parsing is an extremely rich field in natural language processing, it is not the focus of this course, so we'll just cover a few points that are relevant to understanding tree-based models of translation here.

First, the goal of parsing is to go from a sentence, let's say an English sentence E, into a parse tree T. As explained in Section 7.3, parse trees describe the structure of the sentence,

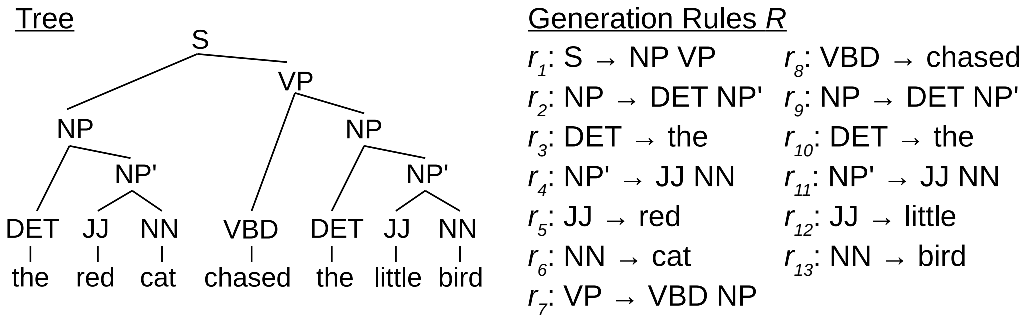

Figure 47: A parse tree and the rules used in its derivation.

as in the example which is reproduced here in Figure 47 for convenience. Like other tasks that we have handeled so far, we will define a probability $P(T \mid E)$, and calculate the tree that maximizes this probability

$$\hat{T} = \underset{T}{\operatorname{argmax}} \; P(T \mid E). \tag{139}$$

There are a number of models to do so, including both *discriminative* models that model the conditional probability $P(T \mid E)$ directly [2] and *generative* models that model the joint probability $P(T, E)$ [4]. Here, we will discuss a very simple type of generative model: probabilistic context free grammars (PCFGs), which will help build up to models of translation that use similar concepts.

The way a PCFG works is by breaking down the generation of the tree into a step-by-step process consisting of applications of **production rules** R. These rules fully specify and are fully specified by T and E. As shown in the right-hand side of Figure 47, the rules used in this derivation take the form of:

$$s^{(l)} \rightarrow s_1^{(r)} s_2^{(r)} \ldots s_N^{(r)} \tag{140}$$

where $s^{(l)}$ is the label of the parent, while $s_n^{(r)}$ are the labels of the right-hand side nodes. More specifically, there are two types of labels: **non-terminals** and **terminals**. Non-terminals label internal nodes of the tree, usually represent phrase labels or part-of-speech tags, and are expressed with upper-case letters in diagrams (e.g. "NP" or "VP"). Terminals label leaf nodes of the tree, usually represent the words themselves, and are expressed with lowercase letters.

The probability is usually specified as the product of the conditional probabilities of the right hand side $\boldsymbol{s}^{(r)}$ given the left hand side symbol $s^{(l)}$:

$$P(T, E) = \prod_{i=1}^{|R|} P(\boldsymbol{s}_i^{(r)} | s_i^{(l)}). \tag{141}$$

The reason why we can use the conditional probability here is because at each time step, we know the identity of parent node of the next rule to be generated, because it has already been generated in one of the previous steps.[42] For example, taking a look at Figure 47, at

[42] At time step 1, we assume that the identity of the root node is the same for all sentences, for example, "S" in the example.

the second time step, in r_1 we just generated "S → NP VP", which indicates that at the next time step we should be generating a rule for which $s_2^{(l)} = \text{NP}$. In general, we follow a left-to-right, depth-first traversal of the tree when deciding the next node to expand, which allows us to uniquely identify the next non-terminal to expand.

The next thing we need to consider is how to calculate probabilities $P(\boldsymbol{s}_i^{(r)} | s_i^{(l)})$. In most cases, these probabilities are calculated from annotated data, where we already know the tree T. In this case, it is simple to calculate these probabilities using maximum likelihood estimation:[43]

$$P(\boldsymbol{s}_i^{(r)} | s_i^{(l)}) = \frac{c(s_i^{(l)}, \boldsymbol{s}_i^{(r)})}{c(s_i^{(l)})}. \tag{142}$$

14.3 Hypergraphs and Hypergraph Search

Now that we have a grammar that specifies rules and assigns each of them a probability, the next question is, given a sentence E, how can we obtain its parse tree? The answer is that the algorithms we use to search for these parse trees are very similar to the Viterbi algorithm, with one big change.

Specifically, the big change is that while the Viterbi algorithm as described before is an algorithm that attempts to find the shortest path through a *graph*, the algorithm for parsing has to find a path through a **hyper-graph**. If we recall from Section 12.3, a graph edge in a WFSA was defined as a previous node, next node, symbol, and score:

$$g = \langle g_\text{p}, g_\text{n}, g_\text{x}, g_\text{s} \rangle. \tag{143}$$

The only difference between this edge in a graph and a **hyper-edge** in a hyper-graph is that a hyper-edge allows each edge to have multiple "previous" nodes, re-defining $\boldsymbol{g}_\text{p}$ as a vector of these nodes:

$$g = \langle \boldsymbol{g}_\text{p}, g_\text{n}, g_\text{x}, g_\text{s} \rangle. \tag{144}$$

To make this example more concrete, Figure 48 shows an example of two parse trees for the famously ambiguous sentence "I saw a girl with a telescope".[44] In this example, each set of arrows joined together at the head is a hyper-edge. To give a few examples, the edge going into the root "$\text{S}_{1,8}$" node would be

$$g = \langle \{\text{NP}_{1,2}, \text{VP}_{2,8}\}, \text{S}_{1,8}, \text{"S} \ \rightarrow \ \text{NP VP"}, -\log P(\text{NP VP} \mid \text{S}) \rangle, \tag{145}$$

and the edge from "i" to "$\text{PRP}_{1,2}$" would be

$$g = \langle \{\text{i}_{1,2}\}, \text{PRP}_{1,2}, \text{"PRP} \ \rightarrow \ \text{i"}, -\log P(\text{i} \mid \text{PRP}) \rangle. \tag{146}$$

As can be seen in the bottom of Figure 48, hypergraphs can also be used to express uncertainty. In this case, there are two nodes going into $\text{VP}_{2,8}$, and depending on which one we choose, our interpretation of the sentence will change. Thus, like we performed search

[43] It is also possible to estimate these probabilities in a semi-supervised or unsupervised manner [14] using an algorithm called the "inside-outside algorithm", but this is beyond the scope of these materials.

[44] In the first example, "with a telescope" is part of the verb phrase, indicating that I used a telescope to see a girl, while the second example has "a girl with a telescope" as a noun phrase, indicating that the girl is in possession of a telescope.

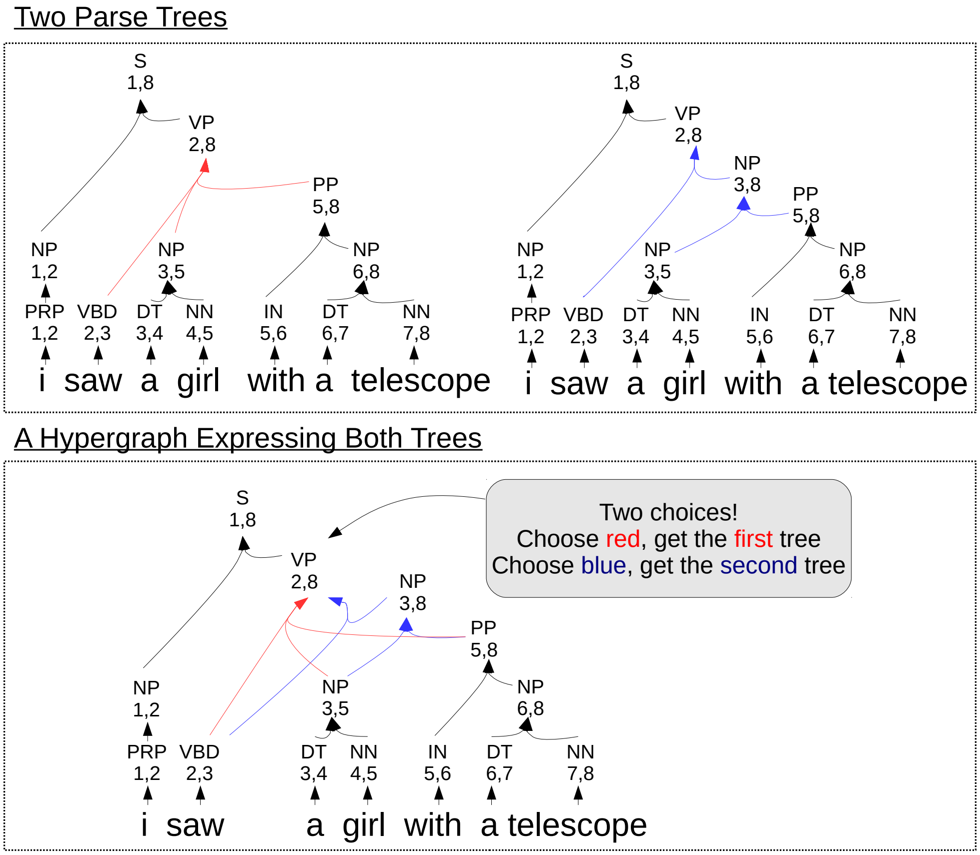

Figure 48: An example of two parse trees, and how they can be combined together into a single hypergraph.

over *graphs* with the Viterbi algorithm in Section 12, we would like to perform search over *hyper-graphs* to find the best scoring parse. Luckily, we can do so with only minimal changes to the overall algorithm. Recall that the forward score calculation in the Viterbi algorithm (Equation 116, reproduced here for convenience), calculated the score as follows:

$$a_i \leftarrow \min_{g \in \{\tilde{g}; \tilde{g}_{\mathrm{n}} = i\}} a_{g_{\mathrm{p}}} + g_{\mathrm{s}}. \tag{147}$$

The one change that we make to apply this to hyper-graphs that instead of adding the score for a single predecessor, we sum over all predecessors in the hyper-edge:

$$a_i \leftarrow \min_{g \in \{\tilde{g}; \tilde{g}_{\mathrm{n}} = i\}} \sum_{j \in \boldsymbol{g}_{\mathrm{p}}} a_j + g_{\mathrm{s}}. \tag{148}$$

Putting this in the context of the trees in the figure, this means that we will calculate a_i in a bottom-up fashion, and every time we calculate the score for a node in the hyper-graph, we will add its edge score to the sum of the scores for its child nodes. The following back-pointers step of the Viterbi algorithm is also similar. In the standard Viterbi algorithm over graphs, if we had a back-pointer $b_i = g$, we would follow its previous node g_{p} to read off the best-scoring string in reverse order. In hyper-graphs, we instead read off *all* the previous nodes in $\boldsymbol{g}_{\mathrm{p}}$, traversing in depth-first left-to-right order, which allows us to read off the rules used in the best-scoring tree in a similar fashion to the right side of Figure 47.

Now that we have the algorithm to perform search over hyper-graphs, the only question is how to create a hyper-graph that allows us to perform the task that we want to do. One example of a parsing algorithm, which we can also frame as a hyper-graph creation algorithm, is the **CKY algorithm** [9, 15]. Interested readers can refer to [8] for precise details, but here we give a brief sketch to demonstrate the process of creating a hyper-graph from a probabilistic model and input sentence E.

The CKY algorithm can be used to parse grammars in **Chomsky normal form** (CNF), which means that the right-hand side of all non-terminals consists of either *a single non-terminal (word)*, or *two non-terminals.* [45] This indicates that all rules will either take the form $P(x \mid A)$ for the former type, or $P(BC \mid A)$ for the latter type. Algorithm 7 demonstrates how we use E and the probabilities of rules in the grammar to build hyper-graph that expresses the probabilities of all the possible trees. Briefly, Lines 2-6 step through all rules of the first type, and Lines 7-19 step through all rules of the second type, adding edges that result in non-zero paths through the hyper-graph, resulting in something like the bottom of Figure 48 (although the hyper-graph in the figure is not in Chomsky normal form). We could then run the Viterbi-like algorithm described above to find the highest-scoring path through this hyper-graph. This would tell us, for example, which of the two trees in the top of Figure 48 is the highest-scoring, and we could return this as our parse.

14.4 Synchronous Context-free Grammars

The PCFGs from the previous section allowed us to find a parse tree T from a sentence E, but they are not a sequence-to-sequence model that can transform one sentence F into another sentence E. However, one simple change to the PCFGs explained above can change them into a form that makes it possible to perform translation. This new, expanded version of PCFGs is called **synchronous context-free grammars** (SCFGs) [15, 3], which are very much like a standard PCFG, with the difference that they specify rules that simultaneously generate trees and sentences in two languages at a time.

The top section of Figure 49 shows an example of SCFG rules in Chomsky normal form. In contrast to the standard PCFG rules from Equation 140, we now have left- and right-hand side symbols for both the source and target languages:

$$\langle s^{(F,l)}, s^{(E,l)} \rangle \rightarrow \langle \boldsymbol{s}^{(F,r)}, \boldsymbol{s}^{(E,r)} \rangle \tag{149}$$

In addition, each symbol on the right-hand side is subscripted by a mapping between the symbols indicating which corresponds to which, as shown in the figure.

These rules can capture a number of interesting phenomena. For example, they can capture the fact that certain words translate into others (e.g. $\langle \text{this}, \text{kore} \rangle$ in the figure), as well as systematic grammatical similarities or differences between languages (e.g. $\langle \text{VP}, \text{VP} \rangle \rightarrow \langle \text{VBZ}_1\text{NP}_2, \text{NP}_2\text{VP}_1 \rangle$ in the figure indicates that the verb and object of a verb phrase are reordered between English and Japanese, capturing the difference between the SVO word order of English and SOV word order of Japanese).

[45] It can be shown that all context-free grammars can be converted into equivalent ones in CNF, so this is not a restriction on the expressivity of the model.

Algorithm 7 An example of how we can perform parsing using the CKY algorithm to create a hyper-graph

```
procedure CKYHyperGraph($E = \boldsymbol{e}_1^{|E|}$)
    for $i$ from 1 to $|E|$                                  ▷ Expand $P(e_i \mid A_{i,i+1})$ do
        for $A \in \{\tilde{A}; P(e_i \mid \tilde{A}) \neq 0\}$ do
            Add edge $\langle \{\}, A_{i,i+1}, \text{“}s \rightarrow e_i\text{”}, -\log P(e_i \mid A) \rangle$
        end for
    end for
    for $k$ from 3 to $|E| + 1$                              ▷ Expand $P(B_{i,j} C_{j,k} \mid A_{i,k})$ do
        for $i$ from $k - 2$ to 1 do
            for $j$ from $i + 1$ to $k - 1$ do
                for All $B_{i,j}$ existing in graph do
                    for All $C_{j,k}$ existing in graph do
                        for $A_{i,k} \in \{\tilde{A}; P(B_{i,j} C_{j,k} \mid \tilde{A}) \neq 0\}$ do
                            Add $\langle \{B_{i,j}, C_{j,k}\}, A_{i,k}, \text{“}A_{i,k} \rightarrow B_{i,j} C_{j,k}\text{”}, -\log P(B_{i,j} C_{j,k} \mid A_{i,k}) \rangle$
                        end for
                    end for
                end for
            end for
        end for
    end for
end procedure
```

In addition, generating translations with this grammar is an extremely straight-forward application of the CKY algorithm described in the previous section. Basically all we need to do is parse the input sentence F using only the source side of the SCFG rules

$$s^{(F,l)} \rightarrow \boldsymbol{s}^{(F,r)}, \tag{150}$$

but keep track of which target-side symbols participated in the parse. We can then re-assemble these target-side rules into a target-side tree, and read off the leaves of the tree, which gives us our translation.

However, the translations generated by this method are still stilted (“kore aru pen desu” feels about as natural as “this is it pen” in English). The reason for this, as noted in the previous section on phrase-based translation, is because there are differences between languages that cannot be easily fit into simple word-to-word translations, and trying to generate translations. In order to resolve this problem, in synchronous grammars, it is common to use rules that move beyond CNF by using multi-word strings, or by mixing non-terminal and terminal symbols, as shown in the bottom of Figure 49. This allows for much more natural handling of various phenomena such as the insertion of function words (the grammatical marker “wa” in Japanese, or determiner “a” in English) and the memorization of the types of phrases with empty variables, like the one mentioned at the beginning of this chapter in Figure Figure 46. While the addition of the rules that are not in CNF precludes the use of the standard CKY algorithm, there are other algorithms that can create the appropriate hyper-graphs with only slightly more complexity: the CKY+ algorithm [1].

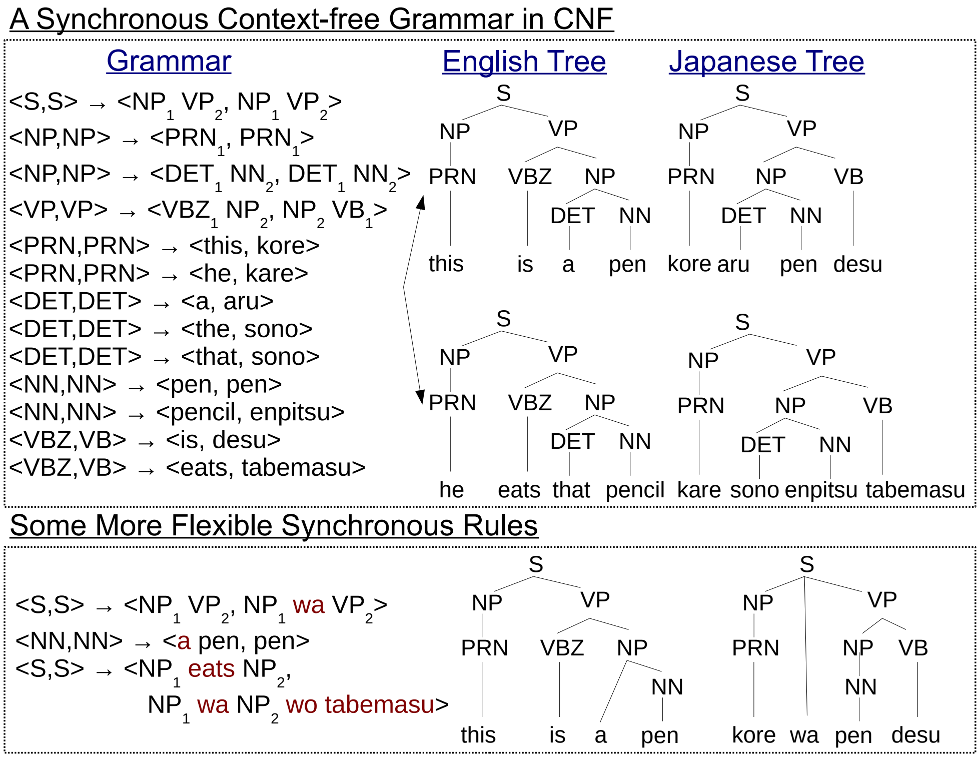
A Synchronous Context-free Grammar in CNF
Grammar
English Tree
Japanese Tree
<S,S> → <NP_1 VP_2, NP_1 VP_2>
<NP,NP> → <PRN_1, PRN_1>
<NP,NP> → <DET_1 NN_2, DET_1 NN_2>
<VP,VP> → <VBZ_1 NP_2, NP_2 VB_1>
<PRN,PRN> → <this, kore>
<PRN,PRN> → <he, kare>
<DET,DET> → <a, aru>
<DET,DET> → <the, sono>
<DET,DET> → <that, sono>
<NN,NN> → <pen, pen>
<NN,NN> → <pencil, enpitsu>
<VBZ,VB> → <is, desu>
<VBZ,VB> → <eats, tabemasu>
S NP VP PRN VBZ NP DET NN
this is a pen
S NP VP PRN NP VB DET NN
kore aru pen desu
S NP VP PRN VBZ NP DET NN
he eats that pencil
S NP VP PRN NP VB DET NN
kare sono enpitsu tabemasu
Some More Flexible Synchronous Rules
<S,S> → <NP_1 VP_2, NP_1 wa VP_2>
<NN,NN> → <a pen, pen>
<S,S> → <NP_1 eats NP_2,
NP_1 wa NP_2 wo tabemasu>
S NP VP PRN VBZ NP NN
this is a pen
S NP VP PRN NP VB NN
kore wa pen desu

Figure 49: An example of an SCFG in Chomsky normal form, which can generate translations in two languages (top), along with some additional SCFG rules that are not in Chomsky normal form, but are helpful in generating more fluent translations.

Of course, like phrase-based translation, it is necessary to extract these phrases from data, which is possible to do with a minimal modification from the standard `phrase-extract` method (Algorithm 6).

1. First, we run `phrase-extract` to find all contiguous phrases in F and E given alignments A. In the case where we are using syntactic annotations such as NP or VP, these will be limited to only phrases that correspond to a syntactic phrase in the source and target sentences (more on relaxing this assumption next section).

2. Next, for each extracted phrase pair $\langle \boldsymbol{f}, \boldsymbol{e} \rangle$, we enumerate all sets of non-overlapping sub-phrases that are contained within the bounds of $\langle \boldsymbol{f}, \boldsymbol{e} \rangle$, replace these sub-phrases with placeholders such such as NP_1 or VP_2.

The exact algorithm can be found in more detail in [3].

14.5 Syntax or Not?

One important thing to note here is that up until now we have assumed that there is a constraint that non-terminals correspond to well-defined grammatical categories such as "noun phrase (NP)" or "verb phrase (VP)" in one or both of the languages. In the ideal case, this has advantages; it makes it possible to ensure that the syntactic structures in the source and target sentences are well-formed and align to each other. However, it also has several disadvantages:

- This constraint also has the side effect of making it more difficult to produce correct translations in the case when syntax diverges between the two languages. This is true in the case of non-literal translations.
- In many cases, it is hard to get a good syntactic parser in one or both of the languages under consideration, and syntactic parsing errors can cause failure in extraction of translation rules.
- Even if we are able to obtain appropriate trees for both languages, syntax-based methods still tend to be more brittle to alignment errors [12] due to the strict requirement that both the source and target side of the phrase pair must correspond exactly to a syntactic phrase.

Tree-to-tree:	<S,S> → <he VB$_1$ NP$_2$, kare wa NP$_2$ wo VB$_1$>
String-to-tree:	<X,S> → <he X$_1$ X$_2$, kare wa NP$_2$ wo VB$_1$>
Tree-to-string:	<S,X> → <he VB$_1$ NP$_2$, kare wa X$_2$ wo X$_1$>
String-to-string:	<X,X> → <he X$_1$ X$_2$, kare wa X$_2$ wo X$_1$>

Figure 50: Examples of various ways to use syntax in translation.

Thus, it is also common to remove this syntactic constraint to improve robustness of tree-based translation models. As shown in Figure 50, this is done by replacing "NP" or "VP" with a generic symbol "X" that can be any grammatical phrase, or can even correspond to a string that has no grammatical category at all. This leads to four combinations of source and target syntax:

Tree-to-tree translation uses syntax on both sides, providing the strongest syntactic constraint. While some works have found this method useful [16], they generally require more complicated methods of absorbing brittleness in training through the consideration of multiple parsing hypotheses.

String-to-tree translation uses syntax on only the target side [6], simultaneously performing translation and parsing on the target side. This has the advantage of relaxing the strong constraints imposed by tree-to-tree translation, while still helping to ensure that the generated translation has some degree of grammatical constraint to ensure that it is well-formed. The disadvantage of these methods is that because they consider a cubic combination of non-terminals (e.g. the loops in Lines 10-12 of Algorithm 7), translation time can be quite slow compared to other methods.

Tree-to-string translation uses syntax on only the source side. This is generally done in one of three ways:

Simultaneous parsing and translation , like string-to-tree translation, simultaneously generates a syntactic parse tree (on the source side) and a translation. However, this method has the disadvantages of string-to-tree (slow computation time) without the advantages (imposing a constraint on the generated hypotheses to ensure better grammaticality) and is not widely used.

Pre-parsing followed by translation , in contrast, first parses the input sentence with a high-accuracy syntactic parser, then performs translation using only translation rules that match with this tree [10]. This has a strong advantage in that it greatly restricts the space of hypotheses that the translator has to explore, greatly improving translation speed, and if the parsing result is correct, accuracy. However, it is quite common for syntactic parsers to make mistakes, degrading translation accuracy. As a remedy to this problem, **forest-to-string translation** encodes many parsing hypotheses in a hypergraph, and performs translation over these hypotheses [11].

Pre-ordering is one final method that is not exactly tree-based translation but shares its motivation of incorporating syntax into translation. Specifically, this method works by first parsing the input sentence, re-ordering the constituents into an order closer to that of the target language, then translating these re-ordered constituents using a standard phrase-based MT system [5]. The advantage of this method is that it can help phrase-based systems overcome their problems with handling long-distance reordering while making minimal changes to the translation engine itself. Disadvantages of error propagation are similar to those experienced in tree-to-string translation, and it is possible to overcome these to some extent by considering multiple pre-ordering candidates when performing translation [13].

String-to-string tree-based translation, also often called **hierarchical phrase-based translation** or **Hiero**, takes the removal of syntax to the extreme: it uses no formal syntax at all [3]. As a result, creation of Hiero models is as simple as that of phrase-based, being possible in cases where no syntactic parser is available at all. However, it does lose the benefit of constraining the hypothesis space using syntax, and the accuracy often lags behind well-trained systems that use some sort of syntax on the source or target side.

14.6 Integration with Language Models

One detail that was glossed over up until this point is how we incorporate a language model into translation. While the exact details are beyond the scope of this chapter (see [3] for details), the basic idea is that the language model probabilities are added to the graph as soon as we know which words will be combined together consecutively in the output $e_1^{|E|}$.

An example of this is shown in Figure 51, where we calculate both the PCFG probability similar to the hypergraph in Section 14.3, but also additionally calculate the probability of a 2-gram language model between the words. At each edge, we have both the PCFG probability for the edge, and the probabilities for the boundaries between the children of each node For example, at $\text{VP}_{2,8}$, we will have the PCFG probability $P(\text{VBD NP PP} \mid \text{VP})$, as well as $P(\text{a} \mid \text{saw})$ which calculates the probability of the words at the boundary between the first two child nodes, and $P(\text{with} \mid \text{girl})$ which calculates the probability of the words at the boundary between the second two child nodes. The reader can confirm that by combining together the probabilities assigned to each edge, we can calculate the full PCFG and 2-gram LM probability for the entire sentence accurately. While this is not exactly a translation

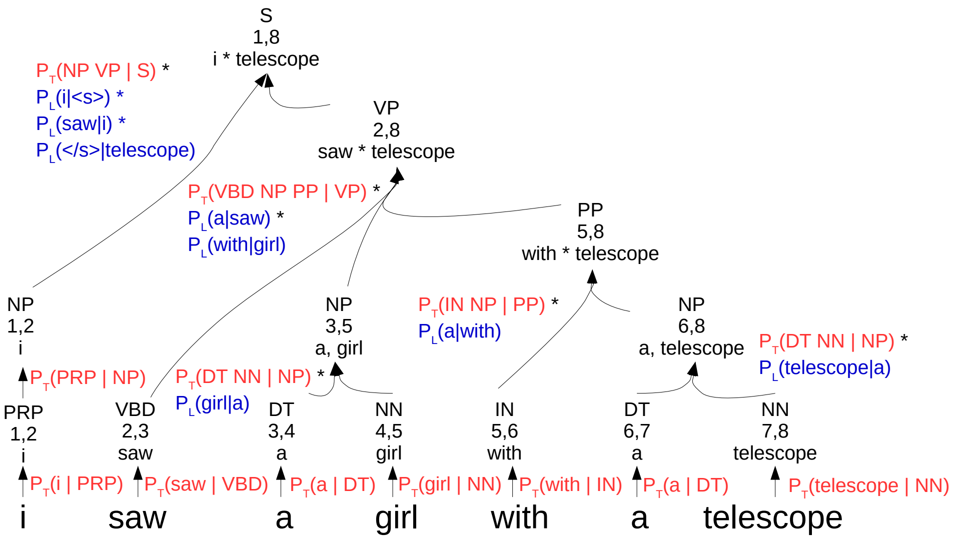

Figure 51: An example of calculating PCFG probabilities (red) and 2-gram language model probabilities (blue) in the same hyper-graph.

model, the general principle would be the same in an SCFG, or any tree-based generation model with the additional n-gram language model integrated.

One thing to note here is that in order to calculate the 2-gram model accurately, we need to keep track of the words on the furthest-most left and furthest-most right side of each subtree as part of the information available to the node. This is because a 2-gram language model needs one word of context e_{t-1} (from the left node), to calculate the probability of the next word e_t (from the right node). While incorporating a language model is essential for improving translation accuracy, this has the unfortunate side effect of expanding the search space for the translation system, making translation less efficient. This is exacerbated even more when using higher order n-gram language models, when we must keep track of $n-1$ words of context on either side of each node. Luckily, there are relatively efficient approximate search algorithms such as **cube pruning** [3] and hierarchical LM search [7], which allow us to search even this complex search space more effectively.

14.7 Tree Substitution Grammars

Finally, it is worth mentioning one more variety of grammar that is often used in translation in addition to SCFGs. This grammar, called a tree substitution grammar, focuses on the replacement of full sub-trees that remember the inner structure of the tree being replaced, as opposed to just flat SCFG rules. An example, contrasting to standard SCFG rules, can be found in Figure 52. As shown in this example, there are some cases in which remembering the whole sub-tree can be useful. In the case in the figure, the PCFG rule "VBD$_1$ NP$_2$ with NP$_3$" does not allow us to disambiguate between the two translations corresponding to the ambiguous "saw a girl with a telescope" in Figure 48. However, the TSG rule that explicitly makes the distinction between the inner structure does allow us to make this distinction, and will rule out one of the two translations based on the source-side parsing result.

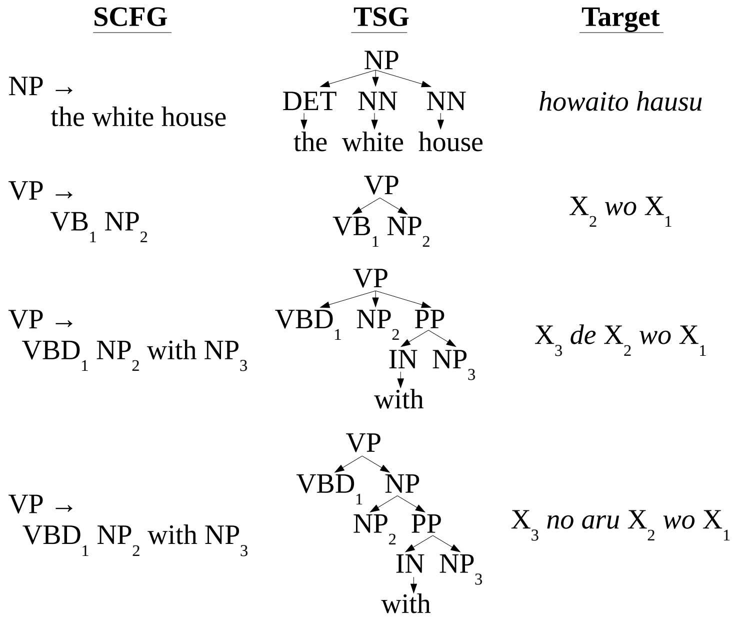

Figure 52: A contrast between SCFG and tree substitution grammar rules.

14.8 Exercise

The exercise this time will be to create a translation system based on Hiero. This will include:

- Modifying phrase extraction code from the previous chapter to make it possible to add open spaces for variables.
- Write code to match these phrases to the input sentence.
- Write the CKY algorithm to find the best result.

Expansions to the model could include integrating an n-gram language model into search. It might also be useful to incorporate syntax in some form, either through tree-to-string, string-to-tree, or tree substitution grammar models.

References

[1] Jean-Cédric Chappelier, Martin Rajman, et al. A generalized CYK algorithm for parsing stochastic CFG. In *TAPD*, pages 133–137, 1998.

[2] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 173–180, 2005.

[3] David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228, 2007.

[4] Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 16–23. Association for Computational Linguistics, 1997.

[5] Michael Collins, Philipp Koehn, and Ivona Kucerova. Clause restructuring for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 531–540, 2005.

[6] Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? In *Proceedings of the 2004 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 273–280, 2004.

[7] Kenneth Heafield, Philipp Koehn, and Alon Lavie. Grouping language model boundary words to speed k–best extraction from hypergraphs. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 958–968, 2013.

[8] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson Education, 2 edition, 2008.

[9] Tadao Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical report, DTIC Document, 1965.

[10] Yang Liu, Qun Liu, and Shouxun Lin. Tree-to-string alignment template for statistical machine translation. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 609–616, 2006.

[11] Haitao Mi, Liang Huang, and Qun Liu. Forest-based translation. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 192–199, 2008.

[12] Graham Neubig and Kevin Duh. On the elements of an accurate tree-to-string machine translation system. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 143–149, 2014.

[13] Yusuke Oda, Taku Kudo, Tetsuji Nakagawa, and Taro Watanabe. Phrase-based machine translation using multiple preordering candidates. In *Proceedings of the 26th International Conference on Computational Linguistics (COLING)*, pages 1419–1428, 2016.

[14] Fernando Pereira and Yves Schabes. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 128–135, 1992.

[15] Daniel H Younger. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208, 1967.

[16] Min Zhang, Hongfei Jiang, Aiti Aw, Haizhou Li, Chew Lim Tan, and Sheng Li. A tree sequence alignment-based tree-to-tree translation model. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 559–567, 2008.