Capturing practical natural language transformations

Kevin Knight

Abstract   We study automata for capturing the transformations in practical natural language processing (NLP) systems, especially those that translate between human languages. For several variations of finite-state string and tree transducers, we survey answers to formal questions about their expressiveness, modularity, teachability, and generalization. We conclude that no formal device yet captures everything that is desirable, and we point to future research.

1 Introduction

Many problems in natural language processing (NLP) consist of transforming one string (or structure) into another. These include translation, summarization, question answering, speech recognition, speech synthesis, semantic interpretation, and language generation. Mapping inputs to their proper outputs amounts to capturing a mathematical relation, i.e., capturing a possibly infinite set of input/output pairs. Given such a relation, we can ask the question: for input x, what is the set Y of all possible outputs? Due to incomplete knowledge about these complex domains, we usually need to reason under uncertainty, so we often add numerical weights to the relation. We then ask: for input x, what is the highest-scoring output y?

In a case like machine translation, the relation is infinite. We cannot capture it by creating a finite list of sentence pairs ⟨x, y⟩ that are acceptable translations. Automata theory provides numerous frameworks and formalisms for concisely capturing such

K. Knight (✉)
Information Sciences Institute, University of Southern California, 4676 Admiralty Way,
Marina del Rey, CA 90292, USA
e-mail: knight@isi.edu

infinite relations. NLP practitioners are frequently interested in making use of standard automata, in order to exploit their formal properties and associated efficient algorithms. Of course, they only want to do this to the extent that the formalism is a good fit for the problem they are working on. In this paper we look at desirable properties of automata from an NLP perspective, and investigate whether the properties hold or not across a range of formalisms. In particular, we look at:

– Expressiveness: Can we can express the required linguistic knowledge in the formalism?
– Modularity: Can we break a complex problem down into pieces, model those pieces, and assemble them into a solution?
– Inclusiveness: In moving from simpler to more expressive formalisms, do we lose the ability to express the simpler things?
– Teachability: Can linguistic knowledge be obtained efficiently from sample input/output pairs?

For each of these broad topics, we select a single specific, provable formal property to investigate. Because we want to bridge between automata theory and NLP practice, we have written this paper in a style accessible to both. We conclude with some open issues to consider.

2 String transducers

A finite-state string transducer (FST) proceeds through its input string from left to right in discrete steps. At each step, some number of input-string symbols (possibly zero) are consumed, and some number of output-string symbols (possibly zero) are emitted. In addition, each step takes the machine from one state to another. A string pair $\langle x, y \rangle$ is considered an element of the FST's modeled relation if the machine (1) begins in some designated start state, (2) after a series of steps consumes all of input string x, (3) emits string y as a result of those same steps, and (4) ends in some designated final state. Because FSTs are non-deterministic, a given input string may map to many outputs.

An FST can be defined as a 6-tuple $\langle Q, \Sigma, \Delta, q_0, f_0, P \rangle$, where Q is a finite set of states, Σ is an alphabet of input symbols, Δ is an alphabet of output symbols, q_0 is a distinguished initial state, f_0 is a distinguished final state, and P is a set of transitions which are themselves 4-tuples. A transition like $\langle q, r, A, BC \rangle$ allows the FST, when in state q, to consume symbol A, emit symbols B and C, and move to state r.

There are several variations for the transition map. Drawing transitions from $Q \text{ x } Q \text{ x } \Sigma^* \text{ x } \Delta^*$ means that a single transition step can consume zero or more symbols and emit zero or more symbols. This choice provides flexibility, and in addition, it admits a useful normal form $Q \text{ x } Q \text{ x } (\Sigma \cup \epsilon) \text{ x } (\Delta \cup \epsilon)$. The generalized sequential machine (GSM) variation is restricted to $Q \text{ x } Q \text{ x } \Sigma \text{ x } \Delta^*$, requiring that each transition consume exactly one input symbol. A GSM cannot generate unbounded output—that is, given an input string of length n, any output string will have a maximum length kn, for some k dependent on the GSM's transition map.

Weighted FSTs add a numerical weight to each transition. From these transition weights, we can compute an overall weight for any string pair ⟨x, y⟩, allowing us to prefer one output over another.

FSTs have nice computational properties, one of which is closure under composition (Schutzenberger 1961; Mohri et al. 2000). This means that a pipeline of FSTs can always be rebuilt as a single FST, allowing a system designer to break a complex problem down into simple pieces, and to assemble those pieces automatically. Composition can happen off-line (e.g., D = A ∘ B ∘ C), and the resulting composed machine can be applied to the input I (e.g., best-path (I ∘ D)).[1] Alternatively, we can wait until we have the input, then perform a synchronized search using all of the FSTs in the pipeline simultaneously (e.g., best-path-synch(I ∘ A ∘ B ∘ C)). In this case, a node in the synchronized search space is taken to be an n-tuple of states drawn from the input and pipelined FSTs (e.g., ⟨i4, a1, b17, c3⟩). This lazy composition (Mohri et al. 2000) is practical in memory usage, and search beams can be applied to make for an efficient approximation to the best-path computation. The search is integrated, in that input x is processed simultaneously by all of the FSTs in the pipeline, rather than being passed from one to the next sequentially. Closure under composition allows all of these types of inference.

FSTs are also efficiently trainable. Exposed to a corpus of input and output string pairs of maximum length n, the forward–backward algorithm (Baum and Eagon 1967) can determine weights for the transitions that locally optimize the corpus probability in time $O(n^2)$.

Portable implementations of weighted FST composition, best path, and training can be found in software toolkits such as Graehl (1997) and Mohri et al. (2000). Mohri et al. (2000) provide an excellent overview of weighted FSTs for speech and NLP, and (Knight and Al-Onaizan 1998; Kumar and Byrne 2003) describe statistical machine translation systems based on weighted FSTs.

3 Tree transducers

String-based FSTs are a good fit for NLP problems that are characterized by stateful left-to-right substitution, e.g., acoustic modeling for speech recognition (Mohri et al. 2000) or transliteration of names across language pairs with different orthographies and sound systems (Knight and Graehl 1998). However, their expressiveness breaks down for more complex problems, such as machine translation, where there is a great deal of reordering, and where many operations are sensitive to syntactic and semantic structure.

Figure 1 shows an example of Arabic-to-English translation, in which the translation of the Arabic verb (at the beginning) must be moved to the middle of the English output sentence. Figure 2 shows the reverse.

---

[1] The "∘" operator indicates composition. A ∘ B is the FST that captures the set of all string pairs ⟨x, y⟩ where there exists a z such that ⟨x, z⟩ is captured by A and ⟨z, y⟩ is captured by B. The "best-path" operator finds the best-cost ⟨x, y⟩ in the FST it is applied to, then prints y. I is the input string x converted to an identity FST which captures the relation ⟨x, x⟩.
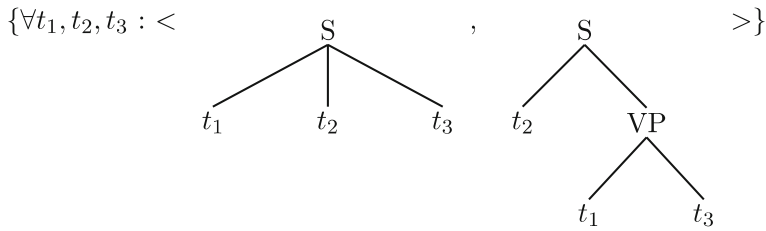
$$\{\forall t_1, t_2, t_3 : <$$

Fig. 1 Arabic-to-English translation example. We want to capture all input/output pairs of this form, where subtrees t_1, t_2, and t_3 are identical in the input and output
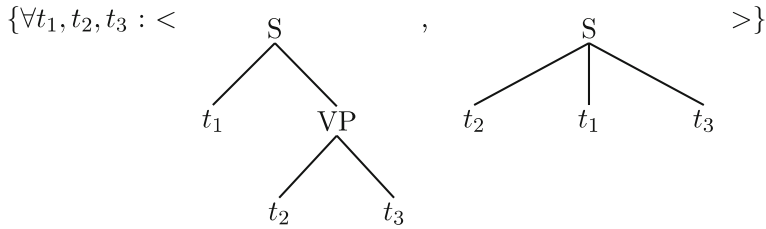
$$\{\forall t_1, t_2, t_3 : <$$

Fig. 2 English-to-Arabic translation example. This is the inverse of the relation in Fig. 1

The utility of hierarchical tree structure was noticed early by Chomsky, and as a result, automata theorists devised tree acceptors and transducers (Doner 1970; Rounds 1970; Thatcher 1970), whose mathematical aim was to generalize the previously-developed string automata. Recently, NLP practitioners have been constructing weighted syntax models for machine translation and other problems, so it has become important to understand the match between practical problems and automata formalisms.

A top-down tree transducer can be defined as a tuple $\langle Q, \Sigma, \Delta, q_0, P \rangle$, where Q is a finite set of states, Σ is an alphabet of input symbols, Δ is an alphabet of output symbols, q_0 is a distinguished initial state, and P is a set of productions (or rules). Here is a sample transducer rule:
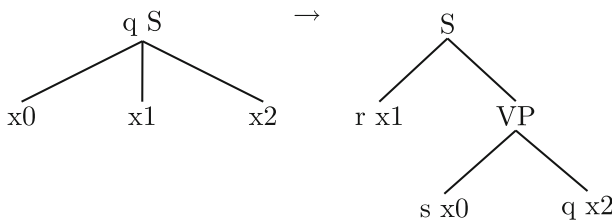
This rule is useful for capturing the relation in Fig. 1. In state q, it consumes an input tree node S, outputs a tree fragment with new S and VP nodes, and recursively processes the three children of the input S node. Note that this rule reorders the input children as it creates the output.[2] In computer-readable format, the same rule looks like:

---

[2] For a fuller overview of tree transducers for NLP, see Knight and Graehl (2005).

$$q\ S(x0, x1, x2) \rightarrow S(r\ x1, VP(s\ x0, q\ x2))$$

Tree transducer rules in the literature (Gécseg and Steinby 1984) have a one-level left-hand side (LHS) with a state, an input-tree symbol, and (optionally) a sequence of variables x0, x1, . . . ,xn. The right-hand side (RHS) shows what the rule emits. The RHS may be multi-leveled, containing both output-tree symbols and labels x0, x1, . . . ,xn, the latter of which are labeled with states for recursive top-down processing.

There are different classes of tree transducers based on the types of rules that are allowed. A rule is said to be *deleting* if its LHS contains a variable that does not appear on the RHS. The RHS in a *copying* rule will contain at least two instances of some LHS variable. A transducer is non-copying (linear) and non-deleting if all of its rules are likewise. The class of non-copying, non-deleting transducers is called LNT (L for linear, N for non-deleting, T for top-down). If we allow deleting, we wind up with the class LT, and if we allow both deleting and copying, we wind up with the class T of top-down transducers. T can express more relations than LT, which can express more relations than LNT (Gécseg and Steinby 1984).

LNT is described in the literature as a generalization of string transduction, in the following sense. If we write strings vertically as non-branching trees, then we can view string transduction as tree transduction, albeit on skinny trees. We can automatically convert any normal-form FST into an LNT transducer. For each transition in the FST, we construct a corresponding LNT rule. There are four cases of interest:

$$
\begin{array}{lll}
\langle q, r, A, B \rangle & q\ A(x0) \rightarrow B(r\ x0) & \\
\langle q, r, A, \epsilon \rangle & q\ A(x0) \rightarrow r\ x0 & \text{``output-}\epsilon\text{''} \\
\langle q, r, \epsilon, B \rangle & q\ x0 \rightarrow B(r\ x0) & \text{``input-}\epsilon\text{''} \\
\langle q, r, \epsilon, \epsilon \rangle & q\ x0 \rightarrow r\ x0 &
\end{array}
$$

In each case, we substitute the LNT rule on the right for the FST transition on the left. We must also apply a technical fix to account for the FST's final state by adding an END token to the bottom of the skinny trees that represent strings.

In this article, we refer to the second kind of rule above as an output-ϵ rule, and the third kind as an input-ϵ rule, in analogy to FSTs, even though there are no literal ϵ symbols in the LNT rules. Note that none of the four rules above are *deleting* rules—to be deleting, the rules would have to omit x0 from the RHS.

4 Properties

Now we revisit the four desirable properties from Sect. 1, assigning to each a particular formal property to investigate. Each topic is potentially quite broad, so we pick specific issues that arise frequently in practice:

– Expressiveness. Can the transducer class express the machine translation transformations in Figs. 1 and 2?

– Modularity. Is the transducer class closed under composition?
– Inclusiveness. Does the transducer class generalize FST?
– Teachability. Does the transducer class admit an efficient algorithm for optimizing rule weights based on a set of input/output training pairs?

Once these questions are considered in detail, we conclude with a diagram showing which automata classes possess which properties.
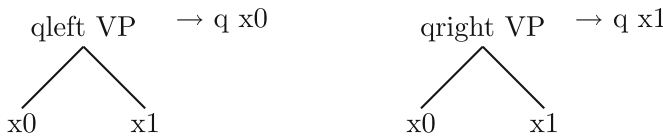
4.1 Basic and extended transducers

LNT is closed under composition (Gécseg and Steinby 1984), but it is not expressive (in the sense above), because it cannot encode the transformation in Fig. 2. An LNT rule matching Fig. 2 must have the form q S(x0, x1) → ???. There is no way for the RHS to insert x0 into the middle of x1.

By contrast, T is expressive, despite the fact that it also has a single-level LHS (Shieber 2004; Knight and Graehl 2005). We accomplish this with a copying rule:

q S   →   S
 /   \   / | \
x0   x1   qleft x1   q x0   qright x1
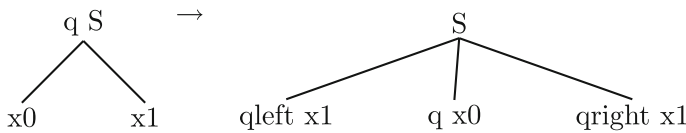
followed by two deleting rules:

qleft VP   → q x0   qright VP   → q x1
  /   \   /   \
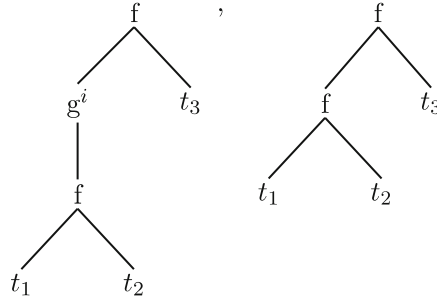 x0   x1   x0   x1

However, T is not closed under composition (Rounds 1970).

The fact that LNT can express the transformation in Fig. 1 but not Fig. 2 is unsatisfying. As a result, Graehl and Knight (2004) define the class xLNT, which allows rules with a multi-level LHS. xLNT is shown to be expressive by the simple rule:

q S   →   S
 /   \   / | \
x0   VP   q x1   q x0   q x2
      / \
     x1   x2
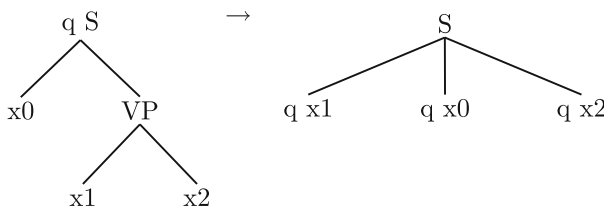
Maletti et al. (2008) show that xLNT, xLT, and xT are strictly more powerful than LNT, LT, and T, respectively. They also show that xLT and xT are not closed under composition. Interestingly, even xLNT is not closed under composition. This is illustrated by the following two relations, τ_1 and τ_2.
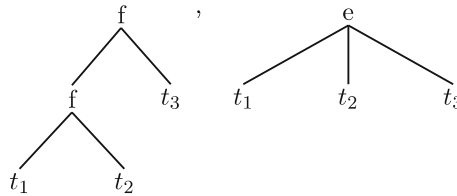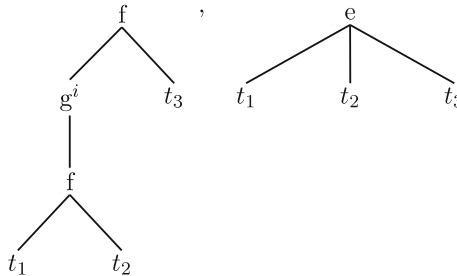
$$\tau_1 = \{\forall t_1, t_2, t_3, i : \;\; < \quad \text{tree}_1 \quad , \quad \text{tree}_2 \quad > \}$$

```
         f                                 f
        / \                               / \
      g^i  t_3                           f   t_3
       |                                / \
       f                              t_1  t_2
      / \
    t_1  t_2
```

Kevin Knight

$$\tau_2 = \{\forall t_1, t_2, t_3, i : \;\; < \quad \text{tree}_1 \quad , \quad \text{tree}_2 \quad > \}$$

```
         f                          e
        / \                       / | \
       f   t_3                  t_1 t_2 t_3
      / \
    t_1  t_2
```

g^i refers to a non-branching tree with i number of g symbols. It is easy to model each of τ_1 and τ_2 by xLNT transducers. (τ_1 requires the use of output-ϵ rules). However, no single xLNT transducer can capture the composition τ_3:

$$\tau_3 = \{\forall t_1, t_2, t_3, i : \;\; < \quad \text{tree}_1 \quad , \quad \text{tree}_2 \quad > \}$$

```
         f                          e
        / \                       / | \
      g^i  t_3                  t_1 t_2 t_3
       |
       f
      / \
    t_1  t_2
```

This is because t_1, t_2, and t_3 are separated by an unbounded number of g's, and no single rule can grab all three subtrees at once. This is interesting because xLNT *does* preserve regularity (Maletti et al. 2008). That is, we can send an input tree (or forest) through τ_1 and send the resulting tree (or forest) through τ_2, yielding another forest. However, it is not possible to do composition, which means that we cannot employ FST-like lazy algorithms for efficient inference.

Synchronous tree substitution grammar (STSG) (Eisner 2003) is slightly less powerful than xLNT, only because xLNT uses states that are separate from the input-symbol vocabulary.

4.2 ϵ-Rules

Now we ask which of the above formalisms is inclusive, i.e., which generalize FST. The answer is that none of them do. As defined in the literature, LNT allows output-ϵ rules, but not input-ϵ rules like:

q x0 $\rightarrow$ A
               |
            r x0

While FSTs can generate unbounded amounts of output given finite input, LNT does not allow this, so it is not a generalization of FST. Rather it is a generalization of string-based GSMs, which consume exactly one input symbol per transition. The same holds for the variation of xLNT as defined in Maletti et al. (2008). However, xLNT as originally defined in Graehl and Knight (2004) allows both output-ϵ and input-ϵ rules, and so generalizes FST.

How important are ϵ-rules in practice? We first consider examples from string transduction. One of the most widely adopted machine translation models is IBM Model 3 (Brown et al. 1993), which casts translation as a word substitution/permutation process. Knight and Al-Onaizan (1998) give a reconstruction of this model as a pipeline of FSTs, and both types of ϵ-transitions appear. Output-ϵ transitions eliminate "zero-fertility" input words that should not be translated, such as the word *do* in English/Spanish translation. Likewise, input-ϵ transitions generate target function words that have no corresponding source word, such as the Spanish object marker *a*. Interestingly, IBM Model 3 bounds the latter by the number of English words, so these input-epsilons could be eliminated in theory.

In many current phrase-based models of translation, by contrast, phrasal chunks are substituted one-for-one, with no deletion or spurious generation—thus, the 2-word phrase *sees Victoria* might be substituted by the 3-word phrase *ve a Victoria*. Kumar and Byrne (2003) present a practical phrase-based translation system built from generic FST tools. Because there is no unbounded generation of output (or unbounded consumption of input), this model can be encoded as an ϵ-free FST (though ϵ's are required for the normal form).

Similar variations exist in tree-based translation models. For example, the system of Galley et al. (2004) acquires xLNT rules from bilingual text corpora. These include rules of the form:

q A $\rightarrow$ q x0
  / \
B   x0

Such rules model the non-translation of words like B = *please* (in travel corpora) or B = *the* (in English/Chinese translation).

Likewise, Graehl and Knight (2004) employ ϵ-rules to better parameterize an English/Japanese translation model, e.g.:

$$q\ x0 \rightarrow q1\ x0$$

$$q\ x0 \rightarrow \text{INS}(i\ x0,\ q1\ x0)$$

$$q\ x0 \rightarrow \text{INS}(q1\ x0,\ i\ x0)$$

$$\text{q1 NP(x0, x1)} \rightarrow \text{NP(q x0, q x1)}$$

$$\text{q1 NP(x0, x1)} \rightarrow \text{NP(q x1, q x0)}$$

$$\text{i x0} \rightarrow wa$$

$$\text{i x0} \rightarrow ga$$

$$\dots$$

Before consuming an input tree node, the model makes a 3-way q-state decision about whether/where to generate target-language function words (such as *wa* or *ga*). The q-state rules decide whether to insert a target function word to the left of the node being processed, to the right of the node being processed, or not at all. The probabilities of the three q-state rules sum to one. The i-state rules decide which function word to insert. Both q-state and i-state rules are input-ϵ rules.

Hence, ϵ-transitions are used frequently in practice, though it is not obvious that system designers really need generation of unbounded output, or consumption of unbounded input. Unbounded output does appear in n-best lists, where a translation like *please X* is accompanied by lower-scoring alternatives *please please X*, *please please please X*, and so on.

4.3 Generalizing FST

To make LNT a generalization of FST, we need to add input-ϵ rules like q x0 → A(r x0). Unfortunately, this destroys closure under composition. A relevant example is:

$$\tau_3 = \{\forall i : <$$
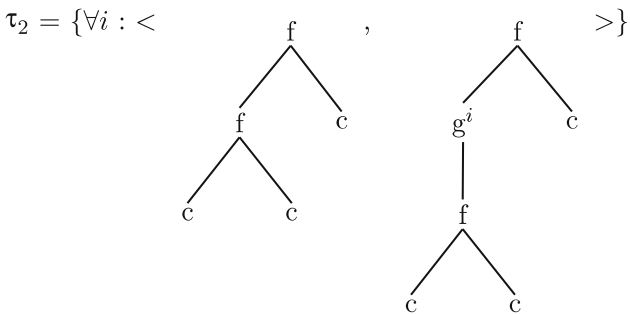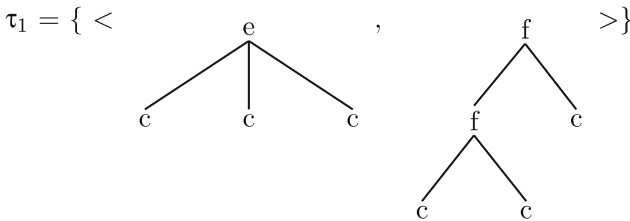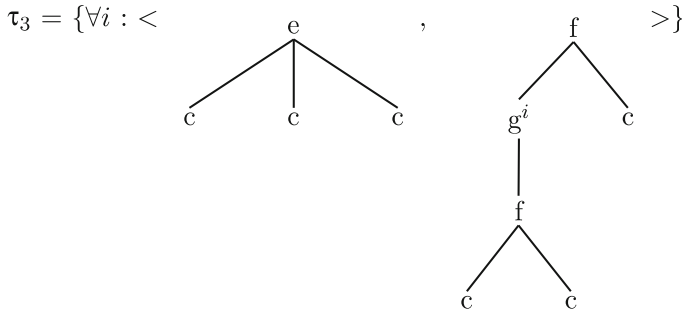
τ_1 and τ_2 can both be captured; τ_2 uses input-ϵ rules to generate an unbounded number of g's. However, τ_3 cannot be captured. This example is simpler than the previous example for xLNT, as the 'c' symbols are atomic and do not stand for whole subtrees. The practical significance is that general LNT composition is impossible, so a general FST composition algorithm may still be needed for the string case.

The example above also covers xLNT. Therefore, while xLNT has expressiveness that seems to be a good match for NLP problems, both input-ϵ and output-ϵ rules independently cause non-closure under composition. Because practitioners may be able to rework their models into ϵ-free versions, it is worth asking whether ϵ-free xLNT is closed under composition. The answer is shown to be no in Arnold and Dauchet (1982), with the following example:

$$h(t_1, t_2, h(t_3, t_4, h(\ldots g(t_{n-1}, t_n)))) \leadsto^{\tau_1}$$
$$g(t_1, g(t_2, g(\ldots g(t_{n-1}, t_n)))) \leadsto^{\tau_2}$$
$$g(t_1, h(t_2, t_3, h(t_4, t_5, h(\ldots h(t_{n-2}, t_{n-1}, t_n)))))$$

Here, τ_2 is any relation that maps its above-specified input to a set that includes its above-specified output; it may non-deterministically produce other outputs as well. While both relations can be modeled individually with ϵ-free xLNT, it is impossible for one xLNT to make the entire leap.

We can summarize the effects on top-down tree transducers of all combinations of: (1) extended LHS, (2) input-ϵ rules, and (3) output-ϵ rules:

x-LHS	Input-ϵ	Output-ϵ	Expressive	Composable	Inclusive
No	No	No	No	Yes	No
No	No	Yes	No	Yes	No
No	Yes	No	No	No	No
No	Yes	Yes	No	No	Yes
Yes	No	No	Yes	No	No
Yes	No	Yes	Yes	No	No
Yes	Yes	No	Yes	No	No
Yes	Yes	Yes	Yes	No	Yes

4.4 Teachability

Finally, we look at whether efficient parameter training procedures exist for various classes. Given input/output trees of maximum size n, Graehl and Knight (2004) present an expectation-maximization algorithm for xT transducers with ϵ rules, which covers all of the top-down classes in this paper. This algorithm runs in $O(n^2)$ time, which is the same asymptotic behavior as the forward–backward algorithm for FSTs (Baum and Eagon 1967). Like forward–backward, it guarantees a set of parameter values that locally optimize the probability of the training corpus.

5 Conclusion

Figure 3 summarizes the top-down transducer classes analyzed in this paper, plus some of the bottom-up transducer classes (suffixed with B), together with their properties.
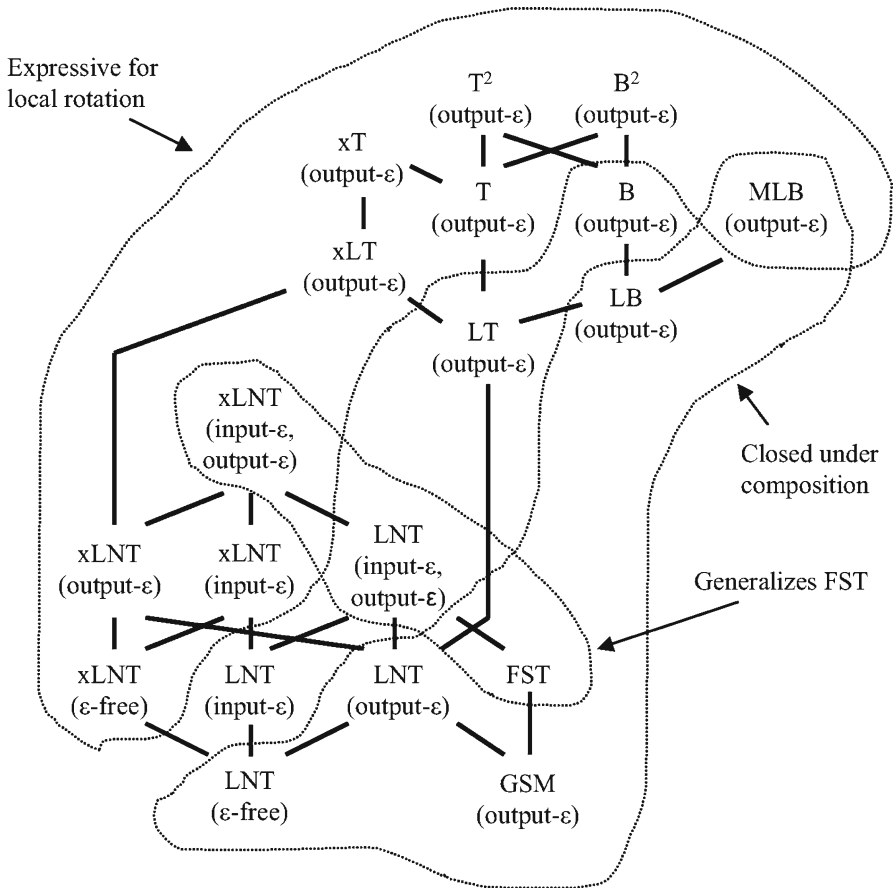

Fig. 3 Classes of tree transducers and their properties

Immediately, we can see that no transducer class has all of the desirable properties we laid out. Classes of interest include LNT (which offers closure under composition), xLNT (which offers expressiveness and generalizes FST), and xT (which offers copying, deleting, and teachability). Due to LNT not generalizing FST, it is still the case that string software toolkits (Mohri et al. 2000) and tree software toolkits (May and Knight 2006) offer overlapping capabilities.

Future problems include exploring more automata frameworks. For example, it appears that bottom-up transducers are not expressive, even with copying and deleting power. However, within the bottom-up family, Maletti (2007) has recently analyzed non-deterministic *multi-state* transducers (MLB in Fig. 3), which can remember multiple output tree fragments as they crawl up the input tree. These machines can carry out the transformation of Fig. 2, and their non-copying version is closed under composition (though, like LNT, they do not generate unbounded output and do not generalize FST).

Another future direction is to propose other desirable formal properties from a machine translation perspective, and to see whether more powerful, non-finite-state formalisms (e.g., Shieber and Schabes 1990) have those properties. Translation models based on dependency grammars have also been proposed (e.g., Shen et al. 2008), and these may also be formalized.

Finally, it would be useful to be able to test, for two tree transducers (both in some class), whether the composition of their transformations can be captured by a third transducer that lies within the same class. There may be no algorithm for this test—for example, there can be no algorithm to tell whether the intersection of two context-free languages (represented by two context-free grammars) is itself context-free. If there were such a composability test, however, most pairs of NLP transducers would be detected as composable, and when applied to string FSTs, tree-based composition would work appropriately.

References

Arnold A, Dauchet M (1982) Morphismes et Bimorphismes d'Arbres. Theor Comput Sci 20:33–93

Baum LE, Eagon JA (1967) An inequality with application to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. Bull Am Math Soc 73:360–363

Brown P, Della Pietra S, Della Pietra V, Mercer R (1993) The mathematics of statistical machine translation: parameter estimation. Comput Linguist 19(2):263–311

Doner J (1970) Tree acceptors and some of their applications. J Comput Syst Sci 4:406–451

Eisner J (2003) Learning non-isomorphic tree mappings for machine translation. In: 41st annual meeting of the Association for Computational Linguistics. Sapporo, Japan, pp 205–208

Galley M, Hopkins M, Knight K, Marcu D (2004) What's in a translation rule? In: Proceedings of the human language technology conference of the North American chapter of the association for computational linguistics: HLT-NAACL 2004. Boston, MA, pp 273–280

Gécseg F, Steinby M (1984) Tree Automata. Akadémiai Kiadó, Budapest

Graehl J (1997) Carmel finite-state toolkit. http://www.isi.edu/licensed-sw/carmel

Graehl J, Knight K (2004) Training tree transducers. In: Proceedings of the human language technology conference of the North American chapter of the association for computational linguistics: HLT-NAACL 2004. Boston, MA, pp 105–112

Knight K, Al-Onaizan Y (1998) Translation with finite-state devices. In: Machine translation and the information soup: third conference of the association for machine translation in the Americas, AMTA'98. Springer, Berlin, pp 421–437

Knight K, Graehl J (1998) Machine transliteration. Comput Linguist 24(4):599–612

Knight K, Graehl J (2005) An overview of probabilistic tree transducers for natural language processing. In: Computational linguistics and intelligent text processing, 6th international conference, CICLing 2005, proceedings. Lecture notes in computer science, vol 3406. Springer, Berlin, pp 1–24

Kumar S, Byrne W (2003). A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In: HLT-NAACL: human language technology conference of the North American chapter of the association for computational linguistics. Edmonton, AL, Canada, pp 63–70

Maletti A (2007) Compositions of extended top-down tree transducers. In: Proceedings of the 1st international conference on language and automata theory and applications. Springer, Berlin, pp 379–390

Maletti A, Graehl J, Hopkins M, Knight K (2008) The power of extended top-down tree transducers. SIAM J Comput (to appear)

May J, Knight K (2006) Tiburon: a weighted tree automata toolkit. In: Proceedings of the international conference on implementation and application of automata (CIAA). Lecture notes in computer science, vol 4094. Springer, Berlin, pp 102–113

Mohri M, Pereira F, Riley M (2000) The design principles of a weighted finite-state transducer library. Theor Comput Sci 231(1):17–32

Rounds WC (1970) Mappings and grammars on trees. Math Syst Theory 4(3):257–287

Schutzenberger MP (1961) A remark on finite transducers. Inf Control 4:185–196

Shen L, Xu J, Weischedel R (2008) A new string-to-dependency machine translation algorithm with a target dependency language model. In: 46th annual meeting of the association for computational linguistics: human language technologies, proceedings of the conference. Columbus, OH, pp 577–585

Shieber S (2004) Synchronous grammars as tree transducers. In: 7th international workshop of TAG and related formalisms (TAG+7). Vancouver, BC, Canada, pp 88–95

Shieber SM, Schabes Y (1990) Synchronous tree-adjoining grammars. In: COLING-90. Papers presented to the 13th international conference on computational linguistics, vol 3. Helsinki, Finland, pp 253–258

Thatcher JW (1970) Generalized sequential machine maps. J Comput System Sci 4:339–367