Towards Probabilistic Acceptors and Transducers for Feature Structures

Daniel Quernheim
Institute for Natural Language Processing
Universität Stuttgart, Germany
Pfaffenwaldring 5b, 70569 Stuttgart
daniel@ims.uni-stuttgart.de

Kevin Knight
University of Southern California
Information Sciences Institute
Marina del Rey, California 90292
knight@isi.edu

Abstract

Weighted finite-state acceptors and transducers (Pereira and Riley, 1997) are a critical technology for NLP and speech systems. They flexibly capture many kinds of stateful left-to-right substitution, simple transducers can be composed into more complex ones, and they are EM- trainable. They are unable to handle long-range syntactic movement, but tree acceptors and transducers address this weakness (Knight and Graehl, 2005). Tree automata have been profitably used in syntax-based MT systems. Still, strings and trees are both weak at representing linguistic structure involving semantics and reference ("who did what to whom"). Feature structures provide an attractive, well-studied, standard format (Shieber, 1986; Rounds and Kasper, 1986), which we can view computationally as directed acyclic graphs. In this paper, we develop probabilistic acceptors and transducers for feature structures, demonstrate them on linguistic problems, and lay down a foundation for semantics-based MT.

1 Introduction

Weighted finite-state acceptors and transducers (Pereira and Riley, 1997) provide a clean and practical knowledge representation for string-based speech and language problems. Complex problems can be broken down into cascades of simple transducers, and generic algorithms (best path, composition, EM, etc) can be re-used across problems.

String automata only have limited memory and cannot handle complex transformations needed in machine translation (MT). Weighted *tree* acceptors and transducers (Gécseg and Steinby, 1984; Knight and Graehl, 2005) have proven valuable in these scenarios. For example, systems that transduce source strings into target syntactic trees performed well in recent MT evaluations (NIST, 2009).

To build the next generation of language systems, we would like to represent and transform deeper linguistic structures, e.g., ones that explicitly capture semantic "who does what to whom" relationships, with syntactic sugar stripped away. *Feature structures* are a well-studied formalism for capturing natural language semantics; Shieber (1986) and Knight (1989) provide overviews. A feature structure is defined as a collection of unordered features, each of which has a value. The value may be an atomic symbol, or it may itself be another feature structure. Furthermore, structures may be re-entrant, which means that two feature paths may point to the same value.

Figure 1 shows a feature structure that captures the meaning of a sample sentence. This semantic structure provides much more information than a typical parse, including semantic roles on both nouns and verbs. Note how "Pascale" plays four different semantic roles, even though it appears only once overtly in the string. The feature structure also makes clear which roles are unfilled (such as the agent of the charging), by omitting them. For computational purposes, feature structures are often represented as rooted, directed acyclic *graphs* with edge and leaf labels.

While feature structures are widely used in hand-built grammars, there has been no compelling proposal for weighted acceptors and transducers for

Proceedings of SSST-6, Sixth Workshop on Syntax, Semantics and Structure in Statistical Translation, pages 76–85,
Jeju, Republic of Korea, 12 July 2012.

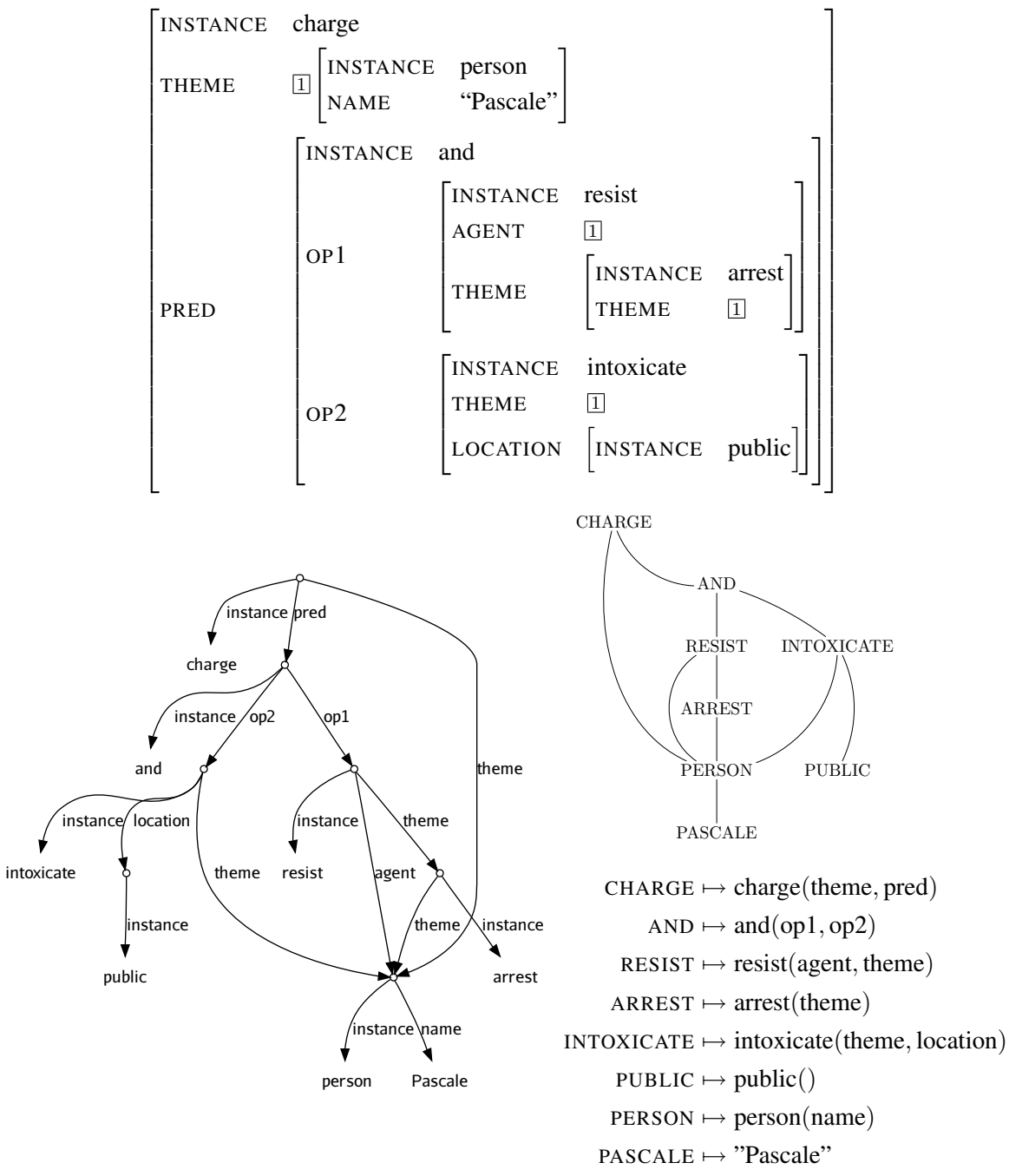

Figure 1: A feature structure representing the semantics of "Pascale was charged with resisting arrest and public intoxication," the corresponding dag, and the simplified dag with argument mapping. Dag edges always point downward.

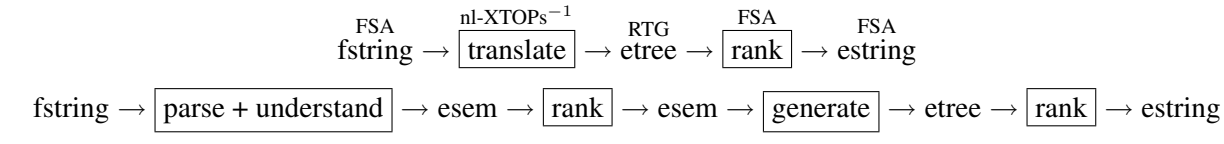

Figure 2: Pipelines for syntax-based and for semantics-based MT. Devices: FSA = finite string automaton; ln-XTOPs = linear non-deleting extended top-down tree-to-string transducer; RTG = regular tree grammar.

	string automata	tree automata	graph automata
k-best	...paths through a WFSA (Viterbi, 1967; Eppstein, 1998)	...trees in a weighted forest (Jiménez and Marzal, 2000; Huang and Chiang, 2005)	?
EM training	Forward-backward EM (Baum et al., 1970; Eisner, 2003)	Tree transducer EM training (Graehl et al., 2008)	?
Determinization	...of weighted string acceptors (Mohri, 1997)	...of weighted tree acceptors (Borchardt and Vogler, 2003; May and Knight, 2006a)	?
Transducer composition	WFST composition (Pereira and Riley, 1997)	Many transducers not closed under composition (Maletti et al., 2009)	?
General tools	AT&T FSM (Mohri et al., 2000), Carmel (Graehl, 1997), OpenFST (Riley et al., 2009)	Tiburon (May and Knight, 2006b)	?

Table 1: General-purpose algorithms for strings, trees and feature structures.

them. Such automata would be of great use. For example, a weighted graph acceptor could form the basis of a semantic language model, and a weighted graph-to-tree transducer could form the basis of a natural language understanding (NLU) or generation (NLG) system, depending on which direction it is employed. Putting NLU and NLG together, we can also envision semantics-based MT systems (Figure 2). A similar approach has been taken by Graham et al. (2009) who incorporate LFG f-structures, which are deep syntax feature structures, into their (automatically acquired) transfer rules. Feature structure graph acceptors and transducers could themselves be learned from semantically-annotated data, and their weights trained by EM.

However, there is some distance to be traveled. Table 1 gives a snapshot of some efficient, generic algorithms for string automata (mainly developed in the last century), plus algorithms for tree automata (mainly developed in the last ten years). These algorithms have been packaged in general-purpose software toolkits like AT&T FSM (Mohri et al., 2000), OpenFST (Riley et al., 2009), and Tiburon (May and Knight, 2006b). A research program for graphs should hold similar value.

Formal graph manipulation has, fortunately, received prior attention. A unification grammar can specify semantic mappings for strings (Moore, 1989), effectively capturing an infinite set of string/graph pairs. But unification grammars seem too powerful to admit the efficient algorithms we desire in Table 1, and weighted versions are not popular. Hyperedge replacement grammars (Drewes et al., 1997; Courcelle and Engelfriet, 1995) are another natural candidate for graph acceptors, and a synchronous hyperedge replacement grammar might serve as a graph transducer. Finally, Kamimura and Slutzki (1981, 1982) propose graph acceptor and graph-to-tree transducer formalisms for rooted directed acyclic graphs. Their model has been extended to multi-rooted dags (Bossut et al., 1988; Bossut and Warin, 1992; Bossut et al., 1995) and arbitrary hypergraphs (Bozapalidis and Kalampakas, 2006; Bozapalidis and Kalampakas, 2008); however, these extensions seem too powerful for NLP. Hence, we use the model of Kamimura and Slutzki (1981, 1982) as a starting point for our definition, then we give a natural language example, followed by an initial set of generic algorithms for graph automata.

2 Preliminaries

In this section, we will define directed acyclic graphs which are our model for semantic structures.

Let us just define some basic notions: We will write $\mathbb{R}$ for the real numbers. An alphabet is just a finite set of symbols.

Intuitively, a rooted ordered directed acyclic graph, or *dag* for short, can be seen as a tree that allows sharing of subtrees. However, it is not necessarily a *maximally* shared tree that has no isomorphic subtrees (consider the examples in Figure 3).

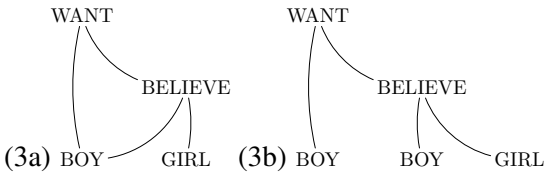

Figure 3: Maximally shared tree (a) and not maximally shared tree (b; note the two BOY nodes) can be distinct dags. The dag in (a) means "The boy wants to believe the girl," while the dag in (b) means "The boy wants some other boy to believe the girl."

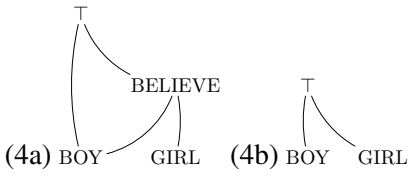

Figure 4: Subdag of dag (3a) and subdag of dag (3b) in Figure 3.

More formally, we define a directed graph over an alphabet Σ as a triple $G = (V, E, \ell)$ of a finite set of nodes V, a finite set of edges $E \subset V \times V$ connecting two nodes each and a labeling function $\ell : V \to \Sigma$. We say that (v, w) is an outgoing edge of v and an incoming edge of w, and we say that w is a child of v and v is a parent of w. A directed graph is a dag if it is

- acyclic: V is totally ordered such that there is no $(v, w) \in E$ with $v > w$;
- ordered: for each V, there is a total order both on the incoming edges and the outgoing edges;
- and rooted: $\min(V)$ is transitively connected by to all other nodes.

This is a simplified account of the dags presented in Section 1. Instead of edge-labels, we will assume that this information is encoded explicitely in the node-labels for the INSTANCE feature and implicitly in the node-labels and the order of the outgoing edges for the remaining features. Figure 1 shows a feature structure and its corresponding dag. Nodes with differently-labeled outgoing edges can thus be differentiated. Since the number of ingoing edges is not fixed, a node can have arbitrary many parents. For instance, the PERSON node in Figure 1 has four parents. We call the number of incoming edges of a given node its *head rank*, and the number of outgoing edges its *tail rank*.

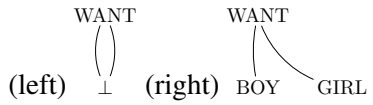

Figure 5: (left) Remainder of (3a) after removing (4a). (right) Dag resulting from replacing (4a) by (4b) in (3a).

We also need incomplete dags in order to compose larger dags from smaller ones. An incomplete dag is a dag in which some edges does not necessarily have to be connected to two nodes; they can be "dangling" from one node. We represent this by adding special nodes $\top$ and $\bot$ to the dag. If an incomplete dag has m edges $(\top, v)$ and n edges $(v, \bot)$, we call it an (m, n)-dag. An (m, n)-dag G can be composed with an (n, o)-dag G' by identifying the n downward-dangling edges of G with the n upward-dangling edges of G' in the right order; the result $G \circ G'$ is a (m, o)-dag. Furthermore, two dags H and H' of type (m, n) and (m', n') can be composed horizontally by putting their upward-dangling edges next to each other and their downward-dangling edges next to each other, resulting in a new $(m + m', n + n')$ dag $H \oplus H'$. If $G_1, \ldots, G_\ell$ can be composed (vertically and horizontally) in such a way that we obtain G, then G_i are called *subdags* of G.

An (m, n)-subdag H of a dag G can be replaced by an (m, n)-subdag H', resulting in the dag G', written $G[H \to H'] = G'$. An example is depicted in Figure 5, showing how a dag is split into two subdags, of which one is replaced by another incomplete dag. Our account of dag replacement is a simplified version of general hypergraph replacement that has been formulated by Engelfriet and Vereijken (1997) and axiomatized by Bozapalidis and Kalampakas (2004).

Trees are dags where every node has at most one incoming edge. Tree substitution is then just a special case of dag composition. We will write the set of dags over an alphabet Σ as D_Σ and the set of trees over Σ as T_Σ, and $T_\Sigma(V)$ is the set of trees with leaves labeled with variables from the set V.

3 Dag acceptors and transducers

The purpose of dag acceptors and dag transducers is to compactly represent (i) a possibly-infinite set of dags, (ii) a possibly-infinite set of (dag, tree)

pairs, and (iii) a possibly-infinite set of (graph, tree, weight) triples.

Dag acceptors and dag transducers are a generalization of tree acceptors and transducers (Comon et al., 2007). Our model is a variant of the dag acceptors defined by Kamimura and Slutzki (1981) and the dag-to-tree transducers by Kamimura and Slutzki (1982). The original definition imposed stricter constraints on the class of dags. Their devices operated on graphs called *derivation dags* (short: d-dags) which are always planar. In particular, the authors required all the parents and children of a given node to be adjacent, which was due to the fact that they were interested in derivation graphs of unrestricted phrase-structure grammar. (While the derivation structures of context-free grammar are trees, the derivation structures of type-0 grammars are d-dags.) We dropped this constraint since it would render the class of dags unsuitable for linguistic purposes. Also, we do not require planarity.

Kamimura and Slutzki (1981, 1982) defined three devices: (i) the bottom-up dag acceptor, (ii) the top-down dag acceptor (both accepting d-dags) and (iii) the bottom-up dag-to-tree transducer (transforming d-dags into trees). We demonstrate the application of a slightly extended version of (ii) to unrestricted dags (semantic dags) and describe a top-down dag-to-tree transducer model, which they did not investigate. Furthermore, we add weights to the models.

A (weighted) *finite dag acceptor* is a structure $M = (Q, q_0, \Sigma, R, w)$ where Q is a finite set of states and q_0 the start state, Σ is an alphabet of node labels, and R is a set of rules of the form $r : \alpha \to \beta$, where r is the (unique) rule identifier and (i) $\alpha \in Q^m(\sigma)$ and $\beta \in r(Q^n)$ for $m, n \in \mathbb{N}$ and some $\sigma \in \Sigma$ (an explicit rule of type (m, n)) or (ii) $\alpha \in Q^m$ and $\beta \in r(Q)$ (an implicit rule of type $(m, 1)$). The function $w : R \to \mathbb{R}$ assigns a weight to each rule.

Intuitively, explicit rules consume input, while implicit rules are used for state changes and joining edges only. The devices introduced by Kamimura and Slutzki (1981) only had explicit rules.

We define the derivation relation of M by rewriting of configurations. A *configuration* of M is a dag over $\Sigma \cup R \cup Q$ with the restriction that every state-labeled node has head and tail rank 1. Let c be a configuration of M and $r : \alpha \to \beta$ an explicit rule of type (m, n). Then $c \Longrightarrow_r c'$ if α matches a subdag of c, and $c' = c[\alpha \to \beta]$.

$$(q)\text{WANT} \to 1(r, q)\ \langle 0.3\rangle \quad (1)$$
$$(q)\text{BELIEVE} \to 2(r, q)\ \langle 0.2\rangle \quad (2)$$
$$(r)\text{BOY} \to 3\ \langle 0.3\rangle \quad (3)$$
$$(r)\text{GIRL} \to 4\ \langle 0.3\rangle \quad (4)$$
$$(r)\emptyset \to 5\ \langle 0.1\rangle \quad (5)$$
$$(q)\emptyset \to 6\ \langle 0.1\rangle \quad (6)$$
$$(q) \to 7(r)\ \langle 0.4\rangle \quad (7)$$

Figure 7: Ruleset of the dag acceptor in Example 1.

Now let c be a configuration of M and $r : \alpha \to \beta$ an implicit rule of type $(m, 1)$. If a configuration c' can be obtained by replacing m nodes labeled α such that all tails lead to the same node and are in the right order, by the single state-node β, then we say $c \Longrightarrow_r c'$. Example derivation steps are shown in Figure 6 (see Example 1). We denote the transitive and reflexive closure of $\Longrightarrow$ by $\Longrightarrow^*$.

A dag G is accepted by M if there is a derivation $q_0(G) \Longrightarrow^* G'$, where G' is a dag over $\sigma(R)$. Note that the derivation steps of a given derivation are partially ordered; many derivations can share the same partial order. In order to avoid spurious derivations, recall that the nodes of G are ordered, and assume that nodes are rewritten according to this order: the resulting derivation is called a *canonical derivation*. The set of all canonical derivations for a given graph G is $D(G)$. The set of all dags accepted by M is the dag language $L(M)$. The weight $w(d)$ of a derivation dag (represented by its canonical derivation) $d = G \Longrightarrow_{r_1} G_1 \Longrightarrow_{r_2} \ldots \Longrightarrow_{r_n} G_n$ is $\prod_{i=1}^{n} w(r_n)$, and the weight of a dag G is $\sum_{d \in D(G)} w(d)$. The weighted language $L(N)$ is a function that maps every dag to its weight in N.

Example 1. *Let*

$$\Sigma = \{\text{GIRL}, \text{BOY}, \text{BELIEVE}, \text{WANT}, \emptyset\}$$

and consider the top-down dag acceptor $M = (\{q, r\}, q, \Sigma, R, w)$ *which has a ruleset containing the explicit and implicit* $(1, 1)$ *rules given in Figure 7. The weights defined by* w *have been written directly after the rules in angle brackets. This ac-*

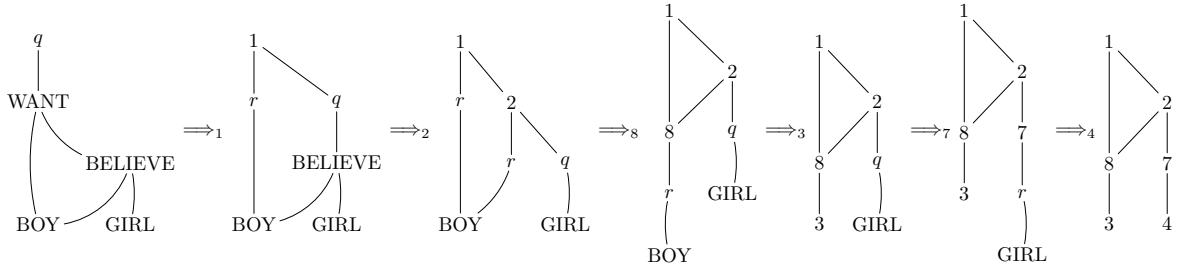

Figure 6: Derivation of a dag using the dag acceptor of Example 1. The weight of the derivation is $w(1) \cdot w(2) \cdot w(8) \cdot w(3) \cdot w(7) \cdot w(4) = 0.3 \cdot 0.2 \cdot 0.2 \cdot 0.3 \cdot 0.4 \cdot 0.3 = 0.000432$.

ceptor can accept dags that involve boys and girls believing and wanting. One of them is given in Figure 3b. To obtain dags that are not trees, let us add the following implicit $(2, 1)$ *and* $(3, 1)$ *rules:*

$$(r, r) \to 8(r)\ \langle 0.2 \rangle \tag{8}$$

$$(r, r, r) \to 9(r)\ \langle 0.1 \rangle \tag{9}$$

A non-treelike dag is given in Figure 3a, while its derivation is given in Figure 6. Note that the effect of rule (8) could be simulated by rule (9).

Let us now define dag-to-tree transducers. Contrarily to Kamimura and Slutzki (1982), who defined only the bottom-up case and were skeptical of an elegant top-down formulation, we only consider top-down devices.

A (weighted) *top-down dag-to-tree transducer* is a machine $T = (Q, q_0, \Sigma, \Delta, R, w)$ which is defined in the same way as a finite dag acceptor, except for the additional output alphabet Δ and the rules' right-hand side. A dag-to-tree transducer explicit rule has the form $r : \alpha \to \beta$ where $\alpha \in Q^m(\Sigma)$ and $\beta \in (T_\Delta(Q(X_n)))^m$ for $m, n \in \mathbb{N}$. Intuitively, this means that the left-hand side still consists of a symbol and m "incoming states", while the right-hand side now are m trees over Δ with states and n variables used to process the n child subdags. Implicit $(m, 1)$ rules are defined in the same way, having m output trees over one variable. The dag-to-tree transducer T defines a relation $L(T) \subseteq D_\Sigma \times T_\Delta \times \mathbb{R}$.

A derivation step of T is defined analogously to the acceptor case by replacement of α by β. However, copying rules (those that use a variable more than once in a right-hand side) and deleting rules (those that do not use a rule at all) are problematic in the dag case. In the tree world, every tree can be broken up into a root symbol and independent subtrees. This is not true in the dag world, where there is sharing between subdags. Therefore, if an edge reaching a given symbol σ is not followed at all (deleting rule), the transducer is going to choke if not every edge entering σ is ignored. In the case of copying rules, the part of the input dag that has not yet been processed must be copied, and the configuration is split into two sub-configurations which must both be derived in parallel. We will therefore restrict ourselves to linear (non-copying) non-deleting rules in this paper.

4 NLP example

Recall the example dag acceptor from Example 1. This acceptor generates an sentences about boys and girls wanting and believing. Figure 3 shows some sample graphs from this language.

Next, we build a transducer that relates these graphs to corresponding English. This is quite challenging, as BOY may be referred to in many ways ("the boy", "he", "him", "himself", "his", or zero), and of course, there are many syntactic devices for representing semantic role clusters. Because of ambiguity, the mapping between graphs and English is many-to-many. Figure 8 is a fragment of our transducer, and Figure 9 shows a sample derivation.

Passives are useful for realizing graphs with empty roles ("the girl is wanted" or "the girl wants to be believed"). Note that we can remove syntactic 0 (zero) elements with a standard tree-to-tree transducer, should we desire.

$$(q_s)\text{WANT}(x, y) \to \text{S}(q_{nomg}(x), \text{is wanted}, q_{zero}(y))$$

$$(q_{infg})\text{BELIEVE}(x, y) \to \text{INF}(q_{zero}(y),\ \text{to be believed}, q_{zerog}(y))$$

$$(q_{zero})\emptyset \to 0$$

$$(q_s)\text{WANT}(x, y) \rightarrow \text{S}(q_{nomb}(x), \text{wants}, q_{infb}(y)) \quad (10)$$

$$(q_{infb})\text{BELIEVE}(x, y) \rightarrow \text{INF}(q_{accg}(x), \text{to believe}, q_{accb}(y)) \quad (11)$$

$$(q_{accg})\text{GIRL} \rightarrow \text{NP(the girl)} \quad (12)$$

$$(q_{nomb}, q_{accb})\text{BOY} \rightarrow \text{NP(the boy), NP(him)} \quad (13)$$

Figure 8: Transducer rules mapping semantic graphs to syntactic trees.

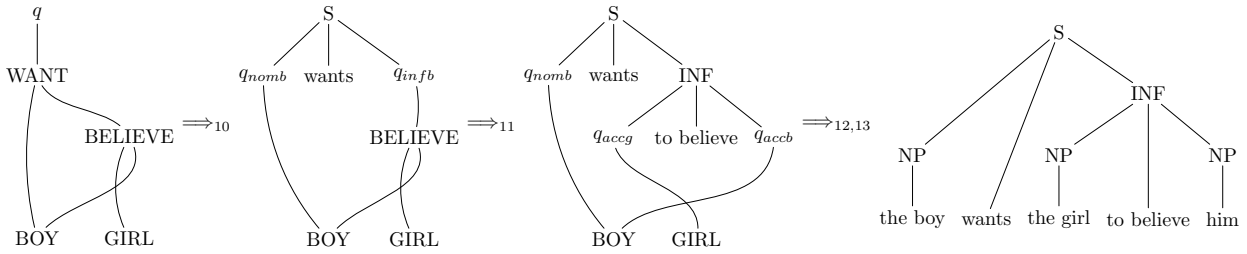

Figure 9: Derivation from graph to tree "the boy wants the girl to believe him".

Events can be realized with nouns as well as verbs ("his desire for her, to believe, him"):

$$(q_{np})\text{WANT}(x, y) \rightarrow \text{NP}(q_{possb}(x), \text{'s desire}, q_{infb}(y))$$

We note that transducer rules can be applied in either direction, semantics-to-English or English-to-semantics. Though this microworld is small, it certainly presents interesting challenges for any graph transduction framework. For example, given "the boy's desire is to be believed by the girl," the transducer's graph must make BOY the theme of BELIEVE.

5 Generic dag acceptor and transducer algorithms

In this section we give algorithms for standard tasks.

5.1 Membership checking

Membership checking is the task of determining, for a given finite dag acceptor M and an input dag G, whether $G \in L(M)$, or in the weighted case, compute the weight of G. Recall that the set of nodes of G is ordered. We can therefore walk through G according to this order and process each node on its own. A very simple algorithm can be given in the framework of "parsing as deduction" (Shieber et al., 1995):

Items: configurations, i.e. dags over $\Sigma \cup Q \cup R$

Axiom: G, a dag over Σ

Goal: dag over R

Inference rule: if an item has only ancestors from Q, apply a matching rule from R to obtain a new item

This algorithm is correct and complete and can be implemented in time $O(2^{|G|})$ since there are exponentially many configurations. Moreover, the set of derivation dags is the result of this parser, and a finite dag acceptor representing the derivation dags can be constructed on the fly. It can be easily extended to check membership of (dag, tree) pairs in a dag-to-tree transducer and to generate all the trees that are obtained from a given dag ("forward application"). In order to compute weights, the techniques by Goodman (1999) can be used.

5.2 1-best and k-best generation

The k-best algorithm finds the highest-weighted k derivations (not dags) in a given (weighted) dag acceptor. If no weights are available, other measures can be used (e.g. the number of derivation steps or symbol frequencies). We can implement the k-best algorithm (of which 1-best is a special case) by generating graphs and putting incomplete graphs on a priority queue sorted by weight. If rule weights are probabilities between 0 and 1, monotonicity ensures that the k-best graphs are found, as the weights of incomplete hypotheses never increase.

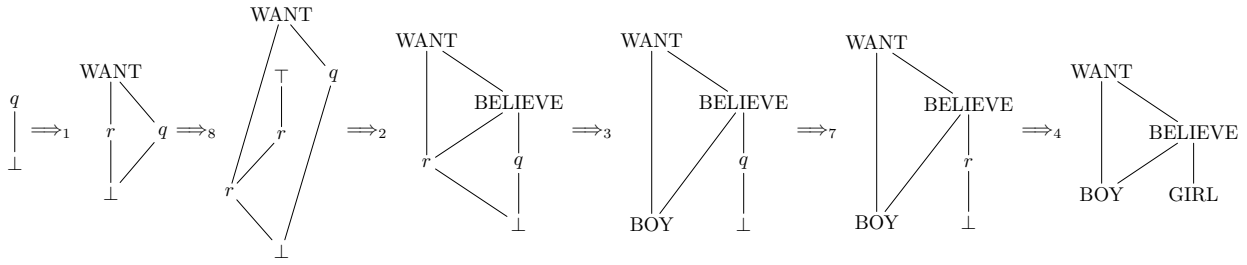

Figure 10: Example derivation in "generation mode".

Dags are generated by taking the basic incomplete dags (rule dags) defined by each rule and concatenating them using the dangling edges. Every dangling edge of the rule dag can be identified with a dangling edge of the current hypothesis (if the orientation matches) or be left unconnected for later connection. In that way, all children and parents for a given node are eventually created. Strictly speaking, the resulting structures are not dags anymore as they can contain multiple $\top$ and $\bot$ symbols. A sample generation is shown in Figure 10. Note how the order of rules applied is different from the example in Figure 6.

Using the dag acceptor as a generating device in this way is unproblematic, but poses two challenges. First, we have to avoid cyclicity, which is easily confirmed by keeping nodes topologically sorted.

Second, to avoid spurious ambiguity (where derivations describe the same derivation dag, but only differ by the order of rule application), special care is needed. A simple solution is to sort the edges in each incomplete dag to obtain a canonical ("leftmost") derivation. We start with the start state (which has head rank 0). This is the first incomplete dag that is pushed on the dag queue. Then we repeatedly pop an incomplete dag G from the dag queue. The first unused edge e of G is then attached to a new node v by identifying e with one of v's edges if the states are compatible. Remaining edges of the new node (incoming or outgoing) can be identified with other unused edges of G or left for later attachment. The resulting dags are pushed onto the queue.

Whenever a dag has no unused edges, it is complete and the corresponding derivation can be returned. The generation process stops when k complete derivations have been produced. This k-best algorithm can also be used to generate tree output for a dag-to-tree transducer, and by restricting the shape of the output tree, for "backward application" (given a tree, which dags map to it?).

6 Future work

The work presented in this paper is being implemented in a toolkit that will be made publicly available. Of course, there is a lot of room for improvement, both from the theoretical and the practical viewpoint. This is a brief list of items for future research:

- Complexity analysis of the algorithms.
- Closure properties of dag acceptors and dag-to-tree transducers as well as composition with tree transducers.
- Investigate a reasonable probabilistic model and training procedures.
- Extended left-hand sides to condition on a larger semantic context, just like extended top-down tree transducers (Maletti et al., 2009).
- Handling flat, unordered, sparse sets of relations that are typical of feature structures. Currently, rules are very specific to the number of children and parents. A first step in this direction is given by implicit rules that can handle a potentially arbitrary number of parents.
- Hand-annotated resources such as (dag, tree) pairs, similar to treebanks for syntactic representations.

Acknowledgements

This research was supported in part by ARO grant W911NF-10-1-0533. The first author was supported by the German Research Foundation (DFG) grant MA 4959/1–1.

References

L. E. Baum, T. Petrie, G. Soules, and N. Weiss. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Statist.*, 41(1):164171.

Björn Borchardt and Heiko Vogler. 2003. Determinization of finite state weighted tree automata. *J. Autom. Lang. Comb.*, 8(3):417–463.

Francis Bossut and Bruno Warin. 1992. Automata and pattern matching in planar directed acyclic graphs. In Imre Simon, editor, *Proc. LATIN*, volume 583 of *LNCS*, pages 76–86. Springer.

Francis Bossut, Max Dauchet, and Bruno Warin. 1988. Automata and rational expressions on planar graphs. In Michal Chytil, Ladislav Janiga, and Václav Koubek, editors, *Proc. MFCS*, volume 324 of *LNCS*, pages 190–200. Springer.

Francis Bossut, Max Dauchet, and Bruno Warin. 1995. A kleene theorem for a class of planar acyclic graphs. *Inf. Comput.*, 117(2):251–265.

Symeon Bozapalidis and Antonios Kalampakas. 2004. An axiomatization of graphs. *Acta Inf.*, 41(1):19–61.

Symeon Bozapalidis and Antonios Kalampakas. 2006. Recognizability of graph and pattern languages. *Acta Inf.*, 42(8-9):553–581.

Symeon Bozapalidis and Antonios Kalampakas. 2008. Graph automata. *Theor. Comput. Sci.*, 393(1-3):147–165.

H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 2007. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`. release October, 12th 2007.

Bruno Courcelle and Joost Engelfriet. 1995. A logical characterization of the sets of hypergraphs defined by hyperedge replacement grammars. *Math. Syst. Theory*, 28(6):515–552.

Frank Drewes, Hans-Jörg Kreowski, and Annegret Habel. 1997. Hyperedge replacement, graph grammars. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars*, pages 95–162. World Scientific.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proc. ACL*, pages 205–208. ACL.

Joost Engelfriet and Jan Joris Vereijken. 1997. Context-free graph grammars and concatenation of graphs. *Acta Inf.*, 34(10):773–803.

David Eppstein. 1998. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673.

Ferenc Gécseg and Magnus Steinby. 1984. *Tree Automata*. Akadémiai Kiadó, Budapest, Hungary.

Joshua Goodman. 1999. Semiring parsing. *Comput. Linguist.*, 25:573–605.

Jonathan Graehl, Kevin Knight, and Jonathan May. 2008. Training tree transducers. *Comput. Linguist.*, 34(3):391–427.

Jonathan Graehl. 1997. Carmel finite-state toolkit. `http://www.isi.edu/licensed-sw/carmel`.

Yvette Graham, Josef van Genabith, and Anton Bryl. 2009. F-structure transfer-based statistical machine translation. In *Proc. LFG*.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proc. IWPT*.

Víctor M. Jiménez and Andrés Marzal. 2000. Computation of the n best parse trees for weighted and stochastic context-free grammars. In *Proc. SSPR/SPR*, volume 1876 of *LNCS*, pages 183–192. Springer.

Tsutomu Kamimura and Giora Slutzki. 1981. Parallel and two-way automata on directed ordered acyclic graphs. *Inf. Control*, 49(1):10–51.

Tsutomu Kamimura and Giora Slutzki. 1982. Transductions of dags and trees. *Math. Syst. Theory*, 15(3):225–249.

Kevin Knight and Jonathan Graehl. 2005. An overview of probabilistic tree transducers for natural language processing. In *Proc. CICLing*, volume 3406 of *LNCS*, pages 1–24. Springer.

Kevin Knight. 1989. Unification: A multidisciplinary survey. *ACM Comput. Surv.*, 21(1):93–124.

Andreas Maletti, Jonathan Graehl, Mark Hopkins, and Kevin Knight. 2009. The power of extended top-down tree transducers. *SIAM J. Comput.*, 39(2):410–430.

Jonathan May and Kevin Knight. 2006a. A better n-best list: Practical determinization of weighted finite tree automata. In *Proc. HLT-NAACL*. ACL.

Jonathan May and Kevin Knight. 2006b. Tiburon: A weighted tree automata toolkit. In Oscar H. Ibarra and Hsu-Chun Yen, editors, *Proc. CIAA*, volume 4094 of *LNCS*, pages 102–113. Springer.

Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. 2000. The design principles of a weighted finite-state transducer library. *Theor. Comput. Sci.*, 231(1):17–32.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Robert C. Moore. 1989. Unification-based semantic interpretation. In *Proc. ACL*, pages 33–41. ACL.

NIST. 2009. NIST Open Machine Translation 2009 Evaluation (MT09). `http://www.itl.nist.gov/iad/mig/tests/mt/2009/`.

Fernando Pereira and Michael Riley. 1997. Speech recognition by composition of weighted finite automata. In *Finite-State Language Processing*, pages 431–453. MIT Press.

Michael Riley, Cyril Allauzen, and Martin Jansche. 2009. OpenFST: An open-source, weighted finite-state transducer library and its applications to speech and language. In Ciprian Chelba, Paul B. Kantor, and Brian Roark, editors, *Proc. HLT-NAACL (Tutorial Abstracts)*, pages 9–10. ACL.

William C. Rounds and Robert T. Kasper. 1986. A complete logical calculus for record structures representing linguistic information. In *Proc. LICS*, pages 38–43. IEEE Computer Society.

Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementation of deductive parsing. *J. Log. Program.*, 24(1&2):3–36.

Stuart M. Shieber. 1986. *An Introduction to Unification-Based Approaches to Grammar*, volume 4 of *CSLI Lecture Notes*. CSLI Publications, Stanford, CA.

Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.