Bottom-Up Unranked Tree-to-Graph Transducers for Translation into Semantic Graphs

Johanna Björklund
Umeå University
Umeå, Sweden
johanna@cs.umu.se

Shay B. Cohen
University of Edinburgh
Edinburgh, UK
scohen@inf.ed.ac.uk

Frank Drewes
Umeå University
Umeå, Sweden
drewes@cs.umu.se

Giorgio Satta
University of Padova
Padova, Italy
satta@dei.unipd.it

Abstract

We propose a formal model for translating unranked syntactic trees, such as dependency trees, into semantic graphs. These tree-to-graph transducers can serve as a formal basis of transition systems for semantic parsing which recently have been shown to perform very well, yet hitherto lack formalization. Our model features "extended" rules and an arc-factored normal form, comes with an efficient translation algorithm, and can be equipped with weights in a straightforward manner.

1 Introduction

In dependency semantic parsing, one is given a natural language sentence and has to output a directed graph representing an associated, most-likely semantic analysis. Semantic parsing integrates tasks that have usually been addressed separately in statistical natural language processing, such as named entity recognition, word sense disambiguation, semantic role labeling, and coreference resolution. Semantic parsing is currently receiving considerable attention, as attested by the number of approaches being proposed for its solution (Oepen et al., 2014, 2015) and by the variety of existing semantic representations and available datasets (Kuhlmann and Oepen, 2016).

A successful approach to dependency semantic parsing by Wang et al. (2015b,a) first parses the input sentence into a dependency tree t, and then applies a transition-based algorithm that translates t into a dependency graph in Abstract Meaning Representation (AMR), a popular semantic representation developed by Banarescu et al. (2013). In this work, we present a finite-state transducer for tree-to-graph translation that can serve as a mathematical model for transition-based systems such as the one by Wang et al. (2015b) and, more in general, for work on the syntax-semantics interface.

Bottom-up tree transducers (Thatcher, 1973) have gained significant attention in the field of machine translation, where they are used to map syntactic phrase structure trees from source to target languages. This holds in particular for their "extended" version, which may process, in a single step, sections of the input consisting of several symbols; see (Maletti et al., 2009) and references therein. We propose a similar formalism for dependency semantic parsing, mapping syntactic dependency trees into directed graphs that represent the associated semantic interpretation.

When translating dependency trees into graphs in a bottom-up fashion, we face two problems. Firstly, bottom-up tree transducers process *ranked* trees, i.e., the number of children at each node is bounded by some constant. Thus, typically, these tree transducers use a single rule to process in one shot a node along with all of its (previously processed) children in the source tree. In contrast, in the case of dependency trees there is no global constant that limits the number of children a node may have, and processing all of the children by means of a single rule is problematic.

Secondly, in an output tree of a bottom-up tree transducer, nodes that are located near one another are translations of nodes in a source tree that are in close proximity as well. This condition is often referred to as *locality*. Locality does no longer hold true when translating trees into graphs. In fact, so-called reentrancy nodes in a graph have several parents, which are translations of nodes in the source tree whose distance from one another may not be bounded by a constant. Reentrancies thus require some form of nonlocal processing, generally not found in tree transducers.

The main contribution of this work is a finite-state tree-to-graph transducer that processes dependency trees in a bottom-up, left-to-right fashion. Our solution to the two problems mentioned above is rather simple. Each node is processed together with its children in several translation steps which consume the children left to right. Furthermore, in order to implement reentrancy, each translated subtree produces a graph annotated with a record of selected vertices, to be made accessible later in the translation process.

Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing, pages 7–17
Dresden, Germany, September 23-25, 2019. ©2019 Association for Computational Linguistics

While our transducers use extended translation rules in the sense of Maletti et al. (2009), they can be cast in a simple normal form, facilitating algorithmic processing. We provide a polynomial time algorithm for translating an input dependency tree into a packed graph forest, from which each translation graph can efficiently be recovered.

Related work. Bottom-up tree-to-graph transducers were introduced by Engelfriet and Vogler (1994, 1998) who based their work on hyperedge replacement. Since the graph construction mechanism we use is equivalent to hyperedge replacement, our notion of tree-to-graph transducers is essentially an unranked and extended generalization of theirs, except for the fact that ours cannot create multiple copies of unbounded material in the input. This ability seems inappropriate for modeling natural language semantics.

The system by Wang et al. (2015b) has inspired our work. A technical comparison between their formalism and ours is made in Remark 1. An alternative approach to the syntax-semantics interface exploits multi-component synchronous tree-adjoining grammars; see Nesson and Shieber (2006) and references therein. However, these formal models yield tree-like semantic representations, as opposed to general graphs.

A common approach in semantic parsing is to extend existing syntactic dependency parsers to produce graphs, realizing translation models from strings to graphs, as opposed to the tree-to-graph model investigated here. On this line, transition-based, greedy parsers have been adapted by Ballesteros and Al-Onaizan (2017), Damonte et al. (2017), Hershcovich et al. (2017), Peng et al. (2018) and Vilares and Gómez-Rodríguez (2018). Despite the fact that the input is a bare string, these systems exploit features obtained from a precomputed run of a dependency parser, thus committing to some best parse tree, similarly to the pipeline model of Wang et al. (2015b). Dynamic programming parsers have also been adapted to produce graphs by Kuhlmann and Jonsson (2015) and Schluter (2015). Semantic translation from strings to graphs is further investigated by Jones et al. (2012) and Peng et al. (2015) using synchronous hyperedge replacement grammars, who provide unsupervised learning algorithms for grammar extraction. Finally, Groschwitz et al. (2018) use a neural supertag parser to map a string into a dependency-style tree representation of the compositional structure of the corresponding AMR graph. More precisely, this tree is a term in a special algebra: its constants denote lexicalized AMR graph fragments, which are combined into larger and larger AMR graphs by two binary algebraic operations for graph combination. These operations supply a partial AMR graph either with an argument or with a modifier. The evaluation of the term then yields the output AMR for the input sentence. The tree-to-graph mapping is entirely deterministic, in contrast to our approach. Groschwitz et al. (2018) also provide an unsupervised alignment algorithm that extracts rules from semantic graph banks.

2 Preliminaries

In this section we introduce the notation and terminology that is used throughout this paper.

General Notation. The set of natural numbers (including zero) is denoted by $\mathbb{N}$, and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. For $n \in \mathbb{N}$ the set $\{1, \ldots, n\}$ is abbreviated to $[n]$. In particular, $[0] = \emptyset$. The set of all finite sequences of elements of a set S is written S^*, ε is the empty sequence, $S^+ = S^* \setminus \{\varepsilon\}$, and 2^S is the powerset of S. Given a sequence w, we write $[w]$ for the set of its elements. Concatenation of sequences s, s' is denoted by juxtaposition or, if preferred for notational clarity, as $s \cdot s'$.

Trees. Let Σ be an alphabet. The set T_Σ of (unranked) trees over Σ is the smallest set such that, for all $f \in \Sigma$ and $t_1, \ldots, t_n \in \mathrm{T}_\Sigma$ ($n \in \mathbb{N}$), we have $f(t_1, \ldots, t_n) \in \mathrm{T}_\Sigma$. In particular $f()$, which we abbreviate by f, is in T_Σ.

The nodes of a tree are identified by their Gorn addresses, which are sequences in $\mathbb{N}_+^*$: the root has the address ε, and if α is the address of a node in t_i then $i\alpha$ is the address of that node in $f(t_1, \ldots, t_n)$. The set of all nodes of t is $N(t)$ and the size of t is $|t| = |N(t)|$.

The label of node α in t is $t(\alpha)$, and the subtree rooted at node α is t/α. For $\Sigma' \subseteq \Sigma$, the set of all nodes $\alpha \in N(t)$ with $t(\alpha) \in \Sigma'$ is denoted by $N_{\Sigma'}(t)$. Throughout the paper, a subset $\{\alpha_1, \ldots, \alpha_k\}$ of the set of nodes of a tree t is denoted as $(\alpha_1, \ldots, \alpha_k)$ to express that its nodes are listed in lexicographic order.

The following notion will play a crucial role in the definition of the translation step for our transducers in Section 3. Let $\square \notin \Sigma$ be a special symbol. A **context** is a tree $c \in \mathrm{T}_{\Sigma \cup \{\square\}}$ that contains exactly one occurrence of $\square$, and this occurrence is a leaf. Given such a context and a tree t, we let $c[t]$ denote the tree obtained from c by replacing $\square$ with t. Thus, $c[t] = t$ if $c = \square$, and otherwise $c[t] = f(s_1, \ldots, s_{i-1}, s_i[t], s_{i+1}, \ldots, s_n)$, where $c = f(s_1, \ldots, s_n)$ and $s_i \in \mathrm{T}_{\Sigma \cup \{\square\}}$ is the context among $s_1, \ldots, s_n$. For contexts $c \neq \square$, the notation $c[t]$ is straightforwardly extended to $c[t_1, \ldots, t_k]$ for trees $t_1, \ldots, t_k$ ($k \in \mathbb{N}$). It yields the tree obtained by inserting the sequence of subtrees $t_1, \ldots, t_k$ at the position marked by $\square$. (This yields a tree since we only use it if $c \neq \square$.) To be precise, if $c = f(s_1, \ldots, s_n)$ and $i \in [n]$ is the index such that $\square$ occurs in s_i, then $c[t_1, \ldots, t_k]$ is equal to $f(s_1, \ldots, s_{i-1}, t_1, \ldots, t_k, s_{i+1}, \ldots, s_n)$ if we have $s_i = \square$; otherwise, it is $f(s_1, \ldots, s_{i-1}, s_i[t_1, \ldots, t_n], s_{i+1}, \ldots, s_n)$.

Graphs. The translation process we propose assembles the output graph by combining smaller graphs into larger ones in a stepwise fashion. For this, every graph has a designated group of vertices, called ports. In the assembly step, ports from the graphs to be combined can be merged.

For a given alphabet Δ, the set $\mathcal{G}_\Delta$ of graphs with labels in Δ consists of all quintuples $G = (V, E, \mathit{lab}, \mathit{port})$ such that

1. V is a finite set of **vertices**,
2. $E \subseteq V \times \Delta \times V$ is the set of labeled **edges**,
3. $\mathit{lab} \colon V \to \Delta$ is a function labeling each vertex, and
4. $\mathit{port} \in V^*$ is a sequence of pairwise distinct vertices called **ports**.

The **size** of G is $|G| = |V| + |E|$. If $\mathit{port} = v_1 \cdots v_n$, then the p-th port v_p of G, $p \in [n]$, is denoted by $\mathit{port}(p)$ and $\mathit{type}(G) = |\mathit{port}|$ is the **type** of G. If the components of G are not explicitly named, they are denoted by V_G, E_G, lab_G, and port_G, respectively. To keep the notation simple, we do not use separate sets for the labels of vertices and edges. Such a distinction may of course be added by partitioning Δ into two sets, but for the present paper this is unnecessary.

3 Bottom-Up Unranked Tree-to-Graph Transducers

Informally, our transducers process the input tree in a locally bottom-up, left-to-right manner. To apply a translation rule with a left-hand side s at a given node α, s must cover α together with $k \geq 0$ of its leftmost subtrees. Hence, these subtrees must have been processed earlier, to the extent necessary to make the part to be processed identical to s. Applying the rule then removes the subtrees and turns α into a state (or turns it from one state into another, if it already was a state due to an earlier step). Disregarding for the moment the partial output graphs involved, this is depicted schematically in Figure 1.

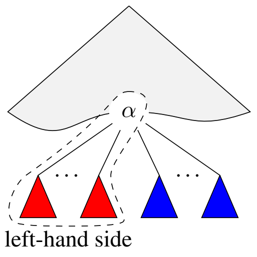

Figure 1: Rule application at node α is locally leftmost (any number, including zero, of the leftmost children of α are consumed) and bottom-up (the left-hand side covers those subtrees all the way down to the leaves). The result of applying a rule at α deletes the subtrees covered by the left-hand side and turns the label of α (a state or input symbol) into a state q.

Note that, in particular, the number k of processed children can be zero, which means that single nodes can initially be turned into states by translation rules whose left-hand sides consist of just one node. More generally, rules in which the root of the left-hand side is an input symbol (with or without children) can be viewed as initializing the processing of the remaining children of that node by turning their parent into an “initial” state.

An (unranked, linear, nondeleting) **bottom-up tree-to-graph transducer** (briefly t2g transducer) is a tuple $\Theta = (\Sigma, \Delta, Q, R, \mu, F)$ consisting of

1. finite input and output alphabets Σ and Δ;
2. a finite set Q of **states** disjoint with Σ, where every state $q \in Q$ has a type $\mathit{type}(q) \in \mathbb{N}$;
3. a finite set R of translation rules defined below;
4. a merging function $\mu \colon 2^\Delta \setminus \{\emptyset\} \to \Delta$; and
5. a set $F \subseteq Q$ of **final states**.

Note that the merging function is finite (because Δ is). It allows us to determine the label of a vertex obtained by merging vertices with different labels. We do not place any restrictions on μ, but consider it as an unknown function that is to be learnt from data. However, it is reasonable to assume that in linguistic settings, μ will be generated by a binary function in the sense that $\mu(\{\delta\}) = \delta$ and $\mu(\Delta' \cup \{\delta\}) = \mu(\{\mu(\Delta'), \delta\})$ for all $\delta \in \Delta$

and $\Delta' \in 2^\Delta \setminus \{\emptyset\}$. Thus, in this case μ can be efficiently represented by a table of size $|\Delta|^2$.

As already mentioned, a translation rule reads a tree fragment of the input, say ϕ, replaces it with a single node labeled by a state, and produces as output some graph fragment ϕ'. The tree ϕ cannot be a single node labeled by a state: see Section 7 for discussion of this restriction. Some nodes α within ϕ may be labeled by a state. This means that the input tree has already undergone some partial processing, at the position corresponding to α, resulting in an output graph fragment which is "associated" with α. Finally, the graph fragment ϕ' produced by the translation rule is obtained by combining the graph fragments associated with the nodes within ϕ labeled by a state, in some way which is specified by the right-hand side of the rule itself. Formally, a **translation rule** $s \to \langle q, G\rangle$ consists of a left-hand side $s \in \mathrm{T}_{\Sigma\cup Q} \setminus Q$ and a right-hand side $\langle q, G\rangle$, where $q \in Q$ and G is a graph with $\mathit{type}(G) = \mathit{type}(q)$. Further, G must fulfill the following condition: if $P = \{\alpha{:}p \mid \alpha \in N_Q(s), p \in [\mathit{type}(s(\alpha))]\}$ then $G \in \mathcal{G}_{\Delta\cup(2^P\setminus\{\emptyset\})}$ and every $\alpha{:}p \in P$ occurs at most once in the labels of vertices in G. A vertex v carrying a label in $2^P \setminus \{\emptyset\}$ is called a **docking vertex**. Intuitively, each $\alpha{:}p \in \mathit{lab}_G(v)$ is a syntactic name (or formal parameter) referring to the p-th port of the graph G_α associated with the node matched by α. During the application of the rule, the p-th port of G_α will be merged with v. This is formalized next.

A **configuration** of Θ is a pair $\langle t, \Gamma\rangle$ with $t \in \mathrm{T}_{\Sigma\cup Q}$ such that $\Gamma\colon N_Q(t) \to \mathcal{G}_\Delta$, where $\mathit{type}(\Gamma(\alpha)) = \mathit{type}(t(\alpha))$ for every $\alpha \in N_Q(t)$. Given an input tree $t_0 \in \mathrm{T}_\Sigma$, the computation of a transducer starts with $\langle t_0, \Gamma_0\rangle$ where Γ_0 is the function with the domain $N_Q(t_0) = \emptyset$. Suppose inductively that, after some computation steps, a configuration $\langle t, \Gamma\rangle$ has been reached. A translation rule $s \to \langle q, G\rangle$ can be applied to this configuration if t can be written as $t = c[f(t_1, \dots, t_n)]$, such that $s = f(t_1, \dots, t_k)$ for some $k \le n$. If so, let α be the node in c such that $c(\alpha) = \square$. Then there is a computation step $\langle t, \Gamma\rangle \to_\Theta \langle \overline{t}, \overline{\Gamma}\rangle$ with $\overline{t} = c[q(t_{k+1}, \dots, t_n)]$, where $\overline{\Gamma}$ is as follows:

1. For every node $\overline{\beta} \in N_Q(\overline{t}) \setminus \{\alpha\}$, if β is the corresponding node in t, then $\overline{\Gamma}(\overline{\beta}) = \Gamma(\beta)$.[1]
2. $\overline{\Gamma}(\alpha)$ is obtained as follows:

   First, take the disjoint union of G and all graphs $\Gamma(\alpha\beta)$, $\beta \in N_Q(s)$, the ports of the resulting graph being those of G.

   Second, for every docking vertex $v \in V_G$, if $\mathit{lab}_G(v) = \{\beta_1{:}p_1, \dots, \beta_m{:}p_m\}$, then merge v with all $v_i = \mathit{port}_{\Gamma(\alpha\cdot\beta_i)}(p_i)$ for $i \in [m]$ and label the merged vertex by $\mu(\{\mathit{lab}_{\Gamma(\alpha\cdot\beta_1)}(v_1), \dots, \mathit{lab}_{\Gamma(\alpha\cdot\beta_m)}(v_m)\})$.

Example 1 Consider the sentence "The emperor loves, respects, and fears himself." A simplified Universal Dependencies parse tree of the sentence is shown leftmost in Figure 2. Here, we have removed the "and" node as well as the additional root node above the "loves" node. Further, the edge labels in the tree should be considered as intermediate nodes (since our trees, for simplicity, and in contrast to graphs, do not have edge labels). The figure shows how a t2g transducer may turn the tree into a semantic graph akin to AMR.

In Step 1 we assume for the sake of illustration that the learning algorithm has seen the leftmost path of the tree ("The emperor loves") often enough to construct an individual ("extended") translation rule for it, and that it has also learned that the emperor referred to is usually Julius. Thus, the translation rule

$$\text{loves}(\texttt{nsubj}(\text{emperor}(\texttt{det}(\text{The})))) \to \langle q_0, G_0\rangle$$

turns node "loves" into the state q_0 and its first dependent vanishes. The pair $\langle q_0, \Gamma(\varepsilon)\rangle = \langle q_0, G_0\rangle$ is illustrated by a dashed box with $\Gamma(\varepsilon)$ shown inside. The numbers next to the vertices indicate the ports. Thus, all three vertices are ports. Note that the rule, for illustration purposes, anticipates the existence of a direct object (or patient) of "love", but labels the corresponding node with a question mark because the processed part of the tree does not determine the argument.

In Step 2, we apply a translation rule of the form $q_0(\texttt{conj}(\text{respects})) \to \langle q_{\text{conj}}, G\rangle$ to add two vertices and four edges to the graph. The graph G in the right-hand side is shown in Figure 3. The ports of G become the ports of $\Gamma(\varepsilon)$, and each of the vertices labeled $\varepsilon{:}p$ is merged with the p-th port of G_0 (i.e., of the $\Gamma(\varepsilon)$ of the previous step).

Step 3 processes $\text{fears}(\texttt{dobj}(\text{himself}))$, turning this subtree into a graph with two ports, with a

[1]Here, the node corresponding to $\overline{\beta}$ is defined in the obvious way, to take care of the change of Gorn addresses that results from the deletion of $t_1, \dots, t_k$: if $\overline{\beta} = \alpha i\gamma$ ($i \in \mathbb{N}_+$), then its corresponding node in t is $\alpha(k+i)\gamma$. If α is not a proper prefix of $\overline{\beta}$, then the corresponding node is $\overline{\beta}$ itself.

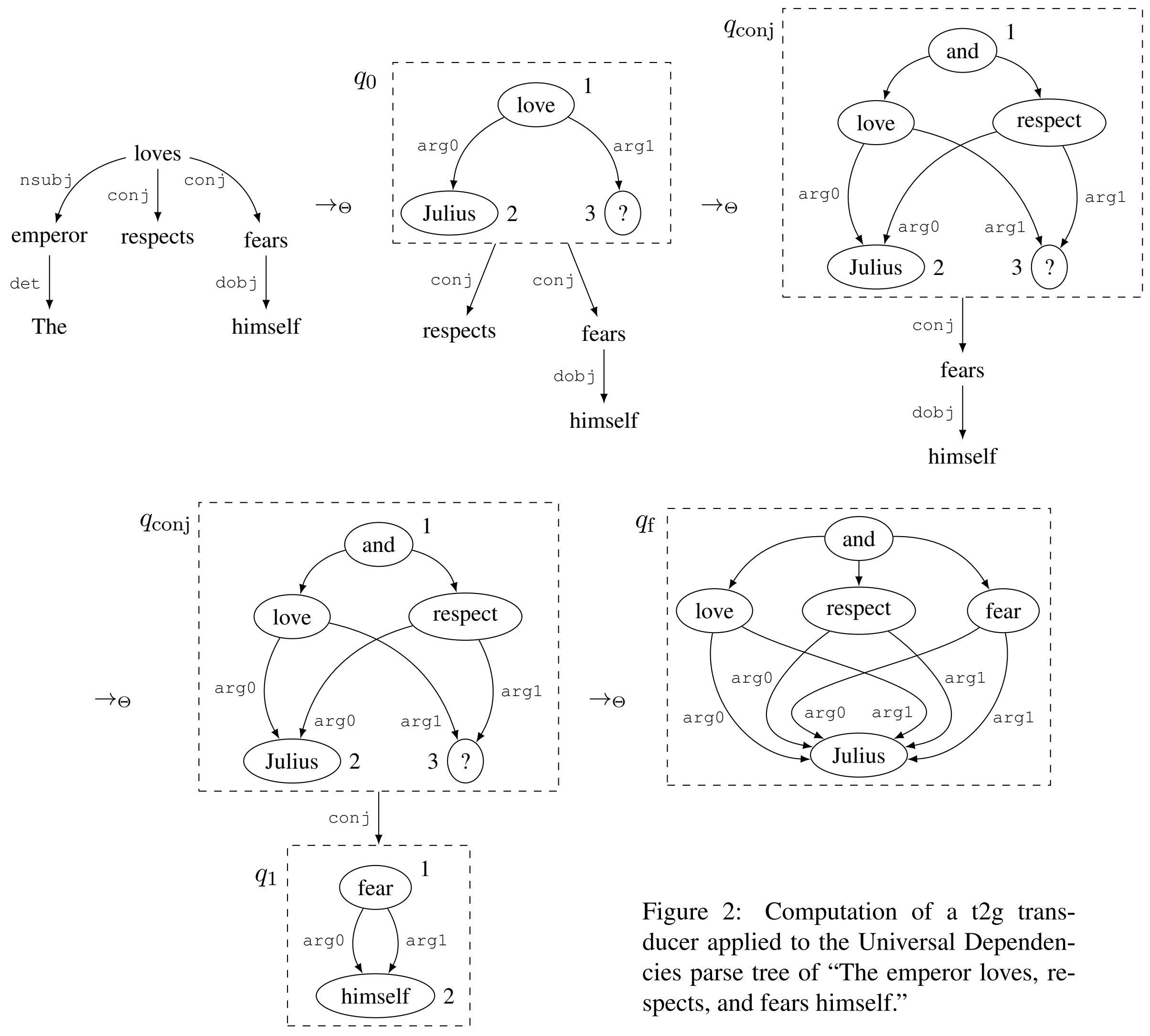

Figure 2: Computation of a t2g transducer applied to the Universal Dependencies parse tree of "The emperor loves, respects, and fears himself."

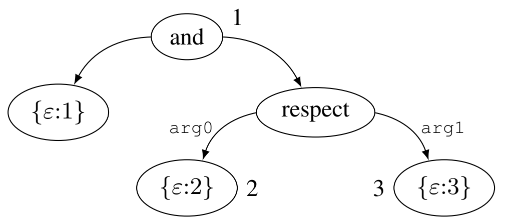

Figure 3: The graph G used in the translation rule $q_0(\texttt{conj}(\text{respects})) \rightarrow \langle q_{\text{conj}}, G\rangle$ of Step 2.

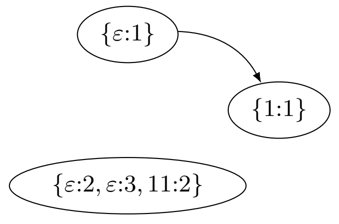

Figure 4: The graph H used in the translation rule $q_{\text{conj}}(\texttt{conj}(q_1)) \rightarrow \langle q_{\text{f}}, H\rangle$ of Step 4.

reentrancy caused by the semantics of "himself". Note that, similarly to Step 1, it is still unclear at this stage which entity "himself" refers to, so we assume that the rule just keeps it. It may be instructive to note that Step 3 is independent of Steps 1 and 2, hence it could just as well have been executed at the very beginning or in between these two.

Finally, Step 4 combines the two graphs by applying a rule of the form $q_{\text{conj}}(\texttt{conj}(q_1)) \rightarrow \langle q_{\text{f}}, H\rangle$. The graph H (shown in Figure 4) contains a vertex with label $\{\varepsilon{:}2, \varepsilon{:}3, 11{:}2\}$, causing the vertices labeled "Julius", "?", and "himself" to be merged. The function μ determines the label of the merged node. Here, we assume that μ gives proper names precedence over pronouns, which in turn take precedence over "?".

Remark 1 The transition-based system of Wang et al. (2015b) and subsequent versions translate dependency trees to AMR by visiting nodes and dependency arcs of the input tree bottom-up and left-to-right. At each node or arc, it greedily applies one out of eight alternative actions, turning the tree into a graph. Six actions are local, meaning that they involve nodes at a close distance in the input tree. These include node or arc relabelling, reversing arc directions, deleting a node, and deleting an arc by merging its two nodes. Each of these actions can easily be captured by some individual translation rule of a t2g transducer.

Two remaining actions are nonlocal: one reattaches a node and the other creates a new arc to form a reentrancy. These actions are restricted to local actions for efficiency reasons (Wang et al., 2015b, Section 3.2), so reattachment attaches to the grandparent or great grandparent, and reentrancy involves sibling nodes only. While t2g transducers can simulate reattachment, a major difference between the two models lies in the creation of reentrancies, as discussed below.

Wang's system can repeatedly apply the reentrancy action, turning for instance n sibling nodes into a clique, for any n. This is not possible in our model, since translation rules can create reentrancies only by accessing a fixed number of vertices "remembered" as ports. We believe this restriction to be linguistically adequate: while the repeated use of conjunctions and modifiers can yield AMR nodes with unbounded node degree, structures resembling cliques of unbounded size would correspond to an unbounded number of concepts, arguments or modifiers, pairwise dependent on each other. This does not appear to be a reasonable linguistic pattern.

However, note that t2g transducers *can* implement a weak form of nonlocality by percolating port nodes across any distance in the underlying derivation tree, though not in any number. This makes it possible to create reentrancies that extend further than to sibling nodes. In terms of translation power, the two formalisms seem close to each other in practice, but we conjecture that they are formally incomparable.

Readers who are familiar with the concept of hyperedge replacement may have noticed that, except for the role of the merging function, the process described in item 3 in the definition of $\to_\Theta$ is just hyperedge replacement (where the replaced hyperedges are kept implicit).

A configuration $\langle t, \Gamma\rangle$ is **final** if $t \in F$, i.e., if the first component has been reduced to a single state, which is final. For an input tree t_0, the set of all output graphs computed by Θ is denoted by $\Theta(t_0)$. It is the set of all graphs $\Gamma(\varepsilon)$ such that $\langle t_0, \Gamma_0\rangle \to^*_\Theta \langle t, \Gamma\rangle$ for some final configuration $\langle t, \Gamma\rangle$. The **transduction** computed by Θ is the set $\{(t, g) \in \mathrm{T}_\Sigma \times \mathcal{G}_\Delta \mid g \in \Theta(t)\}$. The **domain language** of Θ is $\{t \in \mathrm{T}_\Sigma \mid \Theta(t) \neq \emptyset\}$.

We define the **size** of Θ to be the sum of the sizes of its rules. The size of a rule is the size of the tree in the left-hand side plus the size of the graph in the right-hand side. This notion will be used in the next sections for the computational analysis of the algorithms we present.

4 Derivation Trees

In this section, we describe how a computation of a t2g transducer Θ can be represented by means of a tree over the alphabet R, the set of translation rules of Θ. We call these trees **derivation trees** of Θ. Derivation trees will be used in Section 6 to design efficient translation algorithms. We also show that the derivation trees of Θ form a regular tree language (and, in fact, even a local one).

Consider a computation γ of Θ that has the form $\langle t_0, \Gamma_0\rangle \to^+_\Theta \langle q, \Gamma\rangle$ with $t_0 \in \mathrm{T}_\Sigma$ and $q \in Q$. If γ consists of a single step, then a translation rule of the form $r_0\colon t_0 \to \langle q, G\rangle$ has been used. In this case the derivation tree associated with γ, written $d(\gamma)$, is simply r_0.

If γ consists of more than one step, assume that at the last step of γ we have used a translation rule r_0 of the form $s \to \langle q, G\rangle$. We can then write γ as $\langle t_0, \Gamma_0\rangle \to^+_\Theta \langle s, \Gamma'\rangle \to_{r_0} \langle q, \Gamma\rangle$. Let γ' denote the first part $\langle t_0, \Gamma_0\rangle \to^+_\Theta \langle s, \Gamma'\rangle$ of the computation.[2] We define $rk(r_0) = |N_Q(s)|$ to be the **rank** of r_0. In the derivation tree $d(\gamma)$, r_0 has $rk(r_0)$ direct subtrees. If $N_Q(s) = (\alpha_1, \ldots, \alpha_{rk(r_0)})$, then the i-th subtree of r_0 corresponds to the sub-derivation that ended in the state at α_i. Accordingly, we proceed to split the input tree t_0 into smaller pieces on the basis of the node addresses α_i.

In order to describe this thoroughly, we need to determine a correspondence between nodes in s and nodes in t_0. Intuitively, s is a segment at the top of t_0 that extends to the right. To see this, ob-

[2]Recall that $N_Q(s)$ is the set of all nodes in s labeled by states in Q, and that we write $N_Q(s) = (\alpha_1, \ldots, \alpha_{rk(r_0)})$ to indicate that $\alpha_1, \ldots, \alpha_{rk(r_0)}$ are listed in lexicographic order.

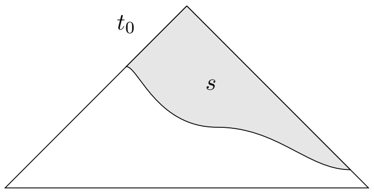

Figure 5: Schematic illustration of the part s of an input tree t_0 which is left after some computation steps. (Note that this is only a structural illustration; some of the node labels in s are not the same as in t_0 anymore, but have been replaced with states.)

serve that the root ε of s corresponds to the root of t_0. Some of the children of ε in t_0 may have been consumed by γ' and thus are no longer present in s. However, this happens strictly from left to right. Therefore, if a child of ε in t_0 is still present in s, then all of its siblings to the right are still present, too. The same pattern continues recursively at the children of these nodes in s. The situation is illustrated schematically in Figure 5.

Formally, for a node $\alpha \in N(s)$ we define the pre-image $\overline{\alpha} \in N(t_0)$ of α inductively over the structure of s, as follows:

1. $\overline{\varepsilon} = \varepsilon$.
2. Assume that $\alpha \cdot 1, \ldots, \alpha \cdot k \in N(s)$ are the children of a node α in s. Then it should be clear that the children of $\overline{\alpha}$ in t_0 are $\overline{\alpha} \cdot 1, \ldots, \overline{\alpha} \cdot n$ for some $n \geq k$. We would like to identify the k-th last children of the preimage with the k children of the postimage, and so we let $\overline{\alpha \cdot i} = \overline{\alpha} \cdot (i+n-k)$ for all $i \in [k]$.

For a set $N \subseteq N(s)$ of node addresses, we let $\overline{N} = \{\overline{\alpha} \mid \alpha \in N\}$.

We are now ready to split t_0 into the subtrees that, via the computation γ', gave rise to the states at $\alpha_1, \ldots, \alpha_{rk(r_0)}$. We do this by defining the sets $N_i \subseteq N(t_0)$ of their nodes, $i \in [rk(r_0)]$. For each $i \in [rk(r_0)]$, N_i is the set of all nodes $\beta \in N(t_0)$ such that $\overline{\alpha_i}$ is the first node in $\overline{N(s)}$ that appears on the path from β to the root of t_0.

Thus, N_i consists of $\overline{\alpha_i}$ and those of its descendants which are not in s anymore, i.e., which have already been "consumed" by the computation γ' in the process of producing node α_i in s. For each $i \in [rk(r_0)]$, define tree t_i as the portion of tree t_0 that is induced by the nodes in N_i.

For each $i \in [rk(r_0)]$, consider the translation rules of γ' that are applied to nodes in N_i. Clearly, the restriction of γ' to them yields a computation γ_i of the form $\langle t_i, \Gamma_0\rangle \to^+_\Theta \langle s(\alpha_i), \Gamma_i\rangle$ whose length is at most the length of γ' and thus less than the length of γ. Let $d(\gamma_i)$ be the derivation tree associated with γ_i. Then we define the derivation tree $d(\gamma)$ to be $r_0(d(\gamma_1), \ldots, d(\gamma_{rk(r_0)}))$.

The inductive procedure above associates a unique derivation tree $d(\gamma)$ with each computation γ of Θ. Observe that each node of $d(\gamma)$ has a label $r \in R$ and a number of children $rk(r)$. This means that the set of derivation trees of Θ, written $D(\Theta)$, is defined over a finite, ranked alphabet. The set $D(\Theta)$ can be recognized by a bottom-up finite-state tree automaton M, as follows.[3] The set of states of M is Q, with F being the subset of accepting states. Its set of rules consists of all $r(q_1, \ldots, q_k) \to q$ such that:

1. $r\colon t \to \langle q, G\rangle$ is a translation rule of Θ,
2. $N_Q(t) = (\alpha_1, \ldots, \alpha_k)$, and
3. $t(\alpha_i) = q_i$ for all $i \in [k]$.

Recall from Section 3 that the size of Θ is the sum of the sizes of its translation rules. We can easily construct rule $r(q_1, \ldots, q_k) \to q$ of M in time linear in the size of r. Hence, M can be constructed in time (and space) linear in the size of Θ.

Given a derivation tree $dt \in D(\Theta)$, such that $dt = r(dt_1, \ldots, dt_k)$, we can compute its input tree $in(dt)$ and its output graph $out(dt)$ recursively, as follows. Suppose the root r of dt is the translation rule $r\colon t \to \langle q, G\rangle$ with $N_Q(t) = (\alpha_1, \ldots, \alpha_{rk(r)})$.

1. If $t_i = in(dt_i)$ for all $i \in [k]$, then $in(dt)$ is obtained from t and $t_1, \ldots, t_k$ by fusing each node α_i with the root of t_i and making $t_i(\varepsilon)$ the label of the fused node. The subtrees of t_i are added to the left of the leftmost subtree of α_i in t. (If α_i is a leaf, t_i just replaces α_i.)
2. If $G_i = out(dt_i)$ for all $i \in [k]$, then the graph $out(dt)$ is obtained from the disjoint union of G and $G_1, \ldots, G_k$ by merging each docking vertex $v \in V_G$ with ports in $G_1, \ldots, G_k$, as follows: if $lab_G(v) = \{\alpha_{i_1}{:}p_1, \ldots, \alpha_{i_m}{:}p_m\}$, then v is merged with all $v_j = port_{G_{i_j}}(p_j)$, $j \in [m]$, and the resulting vertex is labeled by $\mu(\{lab_{G_{i_1}}(v_1), \ldots, lab_{G_{i_m}}(v_m)\})$.

Note that the definition of $out(dt)$ simply reiterates the way in which computations are defined to construct output graphs. As a consequence, it is a straightforward task to show that $dt = d(\gamma)$ for a computation γ that consumes $in(dt)$ and yields the output graph $out(dt)$.

[3]For bottom-up tree automata, see e.g. (Comon et al., 2002, Chapter 1).

Finally, let ρ be the size of the largest translation rule used in dt. It is not difficult to see that, following the recursive procedure above, both $in(dt)$ and $out(dt)$ can be constructed in time $\mathcal{O}(\rho \cdot |dt|)$.

5 Arc-Factored Normal Form

Extended left-hand sides are convenient for specifying transducers, but in order to prove formal properties about transducers or implement algorithms based on them, it is useful to express the transition rules in a more restricted normal form. The normal form introduced in this section processes a tree by first turning every node into a state, and then by visiting the individual arcs of the tree one at each step. A translation rule is in **arc-factored normal form** if its left-hand side is in Σ or has the form $q(q')$ where $q, q' \in Q$. A t2g transducer is in arc-factored normal form if each of its translation rules is in arc-factored normal form.

We now show that every t2g transducer Θ can effectively be transformed into a t2g transducer in arc-factored normal form which computes the same transduction. First, introduce a new state q_f of type 0 for every $f \in \Sigma$ that occurs in the left-hand side of some translation rule, add the rule $f \rightarrow \langle q_f, \emptyset \rangle$ (where $\emptyset$ denotes the empty graph), and replace f by q_f in the left-hand sides of all original rules. Clearly, the computed transduction remains the same and all rules which violate the condition of the arc-factored normal form have left-hand sides in T_Q.

Now, we split rules with large left-hand sides into smaller ones. As long as the transducer is not in arc-factored normal form, select any translation rule $s \rightarrow \langle q, G \rangle$ such that $|s| > 2$. Then s has the form $c[q_1(q_2, t_1, \ldots, t_n)]$ for some context c, states q_1, q_2, and trees $t_1, \ldots, t_n$ ($n \geq 0$). If $k = \mathit{type}(q_1)$ and $\ell = \mathit{type}(q_2)$, we decompose the translation rule into two rules, namely $q_1(q_2) \rightarrow \langle q_{1;2}, H \rangle$ and $c[q_{1;2}(t_1, \ldots, t_n)] \rightarrow \langle q, G' \rangle$, where $q_{1;2}$ is a fresh state with $\mathit{type}(q_{1;2}) = k + \ell$.

The intermediate graph H consists of $k + \ell$ isolated vertices $u_1, \ldots, u_k, v_1, \ldots, v_\ell$ with $\mathit{port}_H = u_1 \cdots u_k v_1 \cdots v_\ell$ and, for all $i \in [k]$ and $j \in [\ell]$, $\mathit{lab}_H(u_i) = \varepsilon{:}i$ and $\mathit{lab}_H(v_j) = 1{:}j$. The effect of this translation rule is to take the disjoint union of the graphs associated with the two nodes, concatenating the port sequences.

The graph G' is obtained from G by appropriately renaming the references of the form $\alpha{\cdot}1{:}p$ where α is the address of $\square$ in c: for every $p \in [\ell]$, if $\alpha{\cdot}1{:}p$ occurs in a label of a vertex in G, then it is replaced by $\alpha{:}(\ell + p)$. Moreover, in every label each port reference of the form $\alpha{\cdot}i{:}p$ for $i > 1$ is replaced by $\alpha{\cdot}(i-1){:}p$.

It should be clear that the two translation rules, executed one after the other, have precisely the same effect as the original one. This completes the proof of the arc-factored normal form.

Note that the size increase implied by the preceding construction is modest. More precisely, each rule will be decomposed into as many rules as there are arcs in the original left-hand side, and the size of graphs in the right-hand sides of intermediate translation rules is at most twice the largest type τ of states in Q. Hence, the total size of the new rules replacing $s \rightarrow \langle q, G \rangle$ is $\mathcal{O}(|s| \cdot \tau + |G|)$. More sophisticated constructions can result in a smaller transducer. A rather simple optimization is to drop all ports from the discrete graph H which do not occur in G, and to identify those referenced in the label of the same docking vertex. We do not further pursue this here.

6 Translation into a Packed Forest

Given a transducer $\Theta = (\Sigma, \Delta, Q, R, \mu, F)$ and an unranked tree t, we construct a suitable representation for the set of all graphs that are translations of t under Θ. We solve the problem in two steps, specified below. To simplify the presentation, we assume Θ is in arc-factored normal form.

6.1 Grounding

The first step annotates every occurrence of a symbol in t with its address, yielding $\hat{t}$, and constructs a new t2g transducer $\Theta_t = (\Sigma', \Delta, Q', R', \mu, F')$ with domain language $\{\hat{t}\}$ and output graphs that are the translations of t by Θ. Let $N(\hat{t}) = N(t)$ and $\hat{t}(\alpha) = t(\alpha)^\alpha$ for all $\alpha \in N(t)$. We restrict the domain language of Θ to the set $\{\hat{t}\}$, in such a way that the translation process of Θ is "preserved". We call this construction the **grounding** of Θ to t. For this, let $k_\alpha = \min\{i \in \mathbb{N}_+ \mid \alpha i \notin N(t)\}$ for every $\alpha \in N(t)$, i.e., k_α is the number of children of α plus one.

1. The input alphabet Σ' consists of all symbols appearing in $\hat{t}$.
2. The set Q' consists of all $\langle q, \alpha, i \rangle$ such that $q \in Q$, $\alpha \in N(t)$, and $i \in [k_\alpha]$. Intuitively, α records the position in the tree and i is the number of the next child to be consumed.
3. The set F' is $\{\langle q, \varepsilon, k_\varepsilon \rangle \mid q \in F\}$.

4. For every translation rule $f \to \langle q, G \rangle$ of Θ and every $\alpha \in N(t)$ with $t(\alpha) = f$, we include $f^\alpha \to \langle \langle q, \alpha, 1 \rangle, G \rangle$ in R'.
5. For every translation rule $q_1(q_2) \to \langle q, G \rangle$ of Θ and every $\alpha i \in N(t)$ ($i \in \mathbb{N}_+$), we let $\langle q_1, \alpha, i \rangle (\langle q_2, \alpha i, k_{\alpha i} \rangle) \to \langle \langle q, \alpha, i+1 \rangle, G \rangle$ be a translation rule in R'.

Note that the grounding algorithm above bears close similarity with the notion of parsing by intersection, which makes use of the construction proposed by Bar-Hillel et al. (1964) for producing a context-free grammar that generates the intersection of the languages of a context-free grammar and a finite-state (string) automaton. It should thus be clear that $\Theta_t(\hat{s}) = \emptyset$ for all $s \in \mathrm{T}_\Sigma \setminus \{t\}$, and $\Theta_t(\hat{t}) = \Theta(t)$.

The construction of Θ_t can be carried out in time proportional to the product of the sizes of Θ and t. In practice, many of the translation rules of Θ_t may be useless. It is possible to avoid this by interleaving the construction of the translation rules of Θ_t with a simulation of the process of parsing by Θ on input t. This has the advantage of pruning the construction, so that useless translation rules are filtered out.

6.2 Graph Forest

In the second step of our translation algorithm, we construct a suitable representation of all the graphs that are obtained in any translation of t based on Θ. Using the t2g transducer Θ_t from the previous step, we can apply the construction outlined in Section 4 and produce a bottom-up finite-state tree automaton M_t whose language is the set $D(\Theta_t)$ of all derivation trees of Θ_t. Together with the interpretation of generated derivation trees dt as $out(dt)$ this yields the desired compact representation of the set $\Theta(t)$ of graphs t translates into. We therefore call M_t a **graph forest** for the translation of t under Θ.

One can now use standard algorithms to, e.g., generate the graphs of the form $out(dt)$. Further, if the rules of Θ are equipped with weights from some weight structure, these weights carry over to the rules of M_t in the obvious way. If we now turn every such weighted rule $r(q_1, \ldots, q_k) \to^w q$ of M_t into the weighted context-free string production $q \to^w r(q_1, \ldots, q_k)$, where states become nonterminal symbols, and rule names, parentheses and commas are viewed as terminal symbols, we get an equivalent weighted context-free grammar generating $D(\Theta_t)$ (where trees are viewed as strings over the mentioned alphabet). One can now apply Knuth's generalization of Dijkstra's shortest paths algorithm (Knuth, 1977) to find the best-scoring derivation tree in $D(\Theta_t)$, and thus the "best" translation of t. As Knuth's algorithm runs in time $O(n \log n)$ in the size of the grammar, the total time required by this process is $O(m \log m)$, where m is the product of the sizes of Θ and t.

7 Discussion

We have developed a novel finite-state transducer that implements nonlocal processing to translate unranked dependency trees into general graphs for semantic representation of natural language.

Our formalism is essentially a finite-state device processing unranked trees in a bottom-up fashion, following a well-assessed tradition in natural language processing. We remark that tree preprocessing to convert unranked trees into binary trees, in the style of the stepwise tree automata of Martens and Niehren (2005), is not attractive from a linguistic standpoint since it might destroy the linguistic intuition underlying translation rules. Therefore, our solution to the problem of processing unranked trees uses on-the-fly binarization at each node. This solution was previously adopted by the Z-automata of Björklund et al. (2019), recognizing dependency trees. In fact, if we remove the graph component from the right-hand side of translation rules in t2g transucers, we obtain Z-automata.

Our definition of translation rules forbids the rewriting of single nodes labeled by a state. This is done to avoid cycling on the same input node for an unbounded number of steps. This ability would make it possible to turn a single input tree into infinitely many semantic graphs that can be arbitrarily larger than the input syntactic tree. Such a model would not be linguistically adequate.

As already remarked, there is a deep similarity between the definition of computation step in t2g transducers and hyperedge replacement. In fact a synchronous hyperedge replacement grammar could easily simulate a t2g transducer.

The next step in this project is the development of algorithms for unsupervised extraction of t2g translation rules from semantic graph corpora.

References

Miguel Ballesteros and Yaser Al-Onaizan. 2017. AMR parsing using stack-LSTMs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1269–1275. Association for Computational Linguistics.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proc. 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186.

Yehoshua Bar-Hillel, Micha Perles, and Eliyahu Shamir. 1964. On formal properties of simple phrase structure grammars. In Y. Bar-Hillel, editor, *Language and Information: Selected Essays on their Theory and Application*, chapter 9, pages 116–150. Addison-Wesley.

Johanna Björklund, Frank Drewes, and Giorgio Satta. 2019. Z-automata for compact and direct representation of unranked tree languages. In *Implementation and Application of Automata - 24th International Conference, CIAA 2019, Košice, Slovakia, July 22-25, 2019, Proceedings*, volume 11601 of *Lecture Notes in Computer Science*, pages 83–94. Springer.

Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. 2002. *Tree Automata Techniques and Applications*. Online publication available at http://www.grappa.univ-lille3.fr/tata.

Marco Damonte, B. Shay Cohen, and Giorgio Satta. 2017. An incremental parser for abstract meaning representation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 536–546. Association for Computational Linguistics.

Joost Engelfriet and Heiko Vogler. 1994. The translation power of top-down tree-to-graph transducers. *Journal of Computer and System Sciences*, 49:258–305.

Joost Engelfriet and Heiko Vogler. 1998. The equivalence of bottom-up and top-down tree-to-graph transducers. *Journal of Computer and System Sciences*, 56(3):332–356.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. AMR dependency parsing with a typed semantic algebra. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1831–1841, Melbourne, Australia. Association for Computational Linguistics.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for UCCA. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1127–1138, Vancouver, Canada. Association for Computational Linguistics.

Bevan Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, and Kevin Knight. 2012. Semantics-based machine translation with hyperedge replacement grammars. In *Proceedings of COLING 2012*, pages 1359–1376. The COLING 2012 Organizing Committee.

Donald E. Knuth. 1977. A generalization of dijkstra's algorithm. *Information Processing Letters*, 6:1–5.

Marco Kuhlmann and Peter Jonsson. 2015. Parsing to noncrossing dependency graphs. *Transactions of the Association for Computational Linguistics*, 3:559–570.

Marco Kuhlmann and Stephan Oepen. 2016. Towards a catalogue of linguistic graph banks. *Computational Linguistics*, 42(4):819–827.

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.

Wim Martens and Joachim Niehren. 2005. Minimizing tree automata for unranked trees. In *Database Programming Languages*, pages 232–246, Berlin, Heidelberg. Springer Berlin Heidelberg.

Rebecca Nesson and Stuart M. Shieber. 2006. Simpler TAG semantics through synchronization. In *Proceedings of the 11th Conference on Formal Grammar*, Malaga, Spain.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, and Zdenka Uresova. 2015. Semeval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th Intl. Workshop on Semantic Evaluation (SemEval 2015)*, pages 915–926.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. 2014. Semeval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proc. 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 63–72.

Xiaochang Peng, Daniel Gildea, and Giorgio Satta. 2018. AMR parsing with cache transition systems. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 4897–4904. AAAI Press.

Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A synchronous hyperedge replacement grammar based approach for amr parsing. In *Proceedings of the Nineteenth Conference on Computational*

Natural Language Learning, pages 32–41. Association for Computational Linguistics.

Natalie Schluter. 2015. The complexity of finding the maximum spanning DAG and other restrictions for DAG parsing of natural language. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics, *SEM 2015, June 4-5, 2015, Denver, Colorado, USA.*, pages 259–268.

James W. Thatcher. 1973. Tree automata: An informal survey. In A. V. Aho, editor, *Currents in the Theory of Computing*, pages 143–172. Prentice Hall, New York.

David Vilares and Carlos Gómez-Rodríguez. 2018. A transition-based algorithm for unrestricted AMR parsing. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 142–149. Association for Computational Linguistics.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015a. Boosting transition-based AMR parsing with refined actions and auxiliary analyzers. In *Proc. 53rd Annual Meeting of the Association for Computational Linguistics and the 7th Intl. Joint Conf. on Natural Language Processing (Short Papers)*, pages 857–862.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015b. A transition-based algorithm for AMR parsing. In *Proc. 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, (NAACL HLT 2015), Denver, Colorado, USA, 2015*, pages 366–375.