Tree-to-graph transductions with scope[⋆]

Johanna Björklund[1]

Umeå University, Sweden
johanna.bjorklund@umu.se

Abstract. High-level natural language processing requires formal languages to represent semantic information. A recent addition of this kind is abstract meaning representations. These are graphs in which nodes encode concepts and edges relations. Node-sharing is common, and cycles occur. We show that the required structures can be generated through the combination of (i) a regular tree grammar, (ii) a sequence of linear top-down tree transducers, and (iii) a fold operator that merges selected nodes. Delimiting the application of the fold operator to connected subgraphs gains expressive power, while keeping the complexity of the associated membership problem in polynomial time.

1 Introduction

Machine learning has been successfully applied to natural-language processing (NLP) tasks such as part-of-speech tagging [17], parsing [6] and machine translation [10]. In these works, features are predominately lexical and syntactic attributes, but higher-order tasks such as summarisation and topic-identification would benefit from the addition of semantic information. Banarecsu et al. [4] recently proposed abstract meaning representations (AMR), a semantic representation language that expresses logical meaning at the level of sentences. From a mathematical perspective, AMRs are directed graphs, in which nodes encode concepts, and edges relations. AMRs were presented by example with a bank of several thousand hand-annotated sentences. A technical specification is provided by Banarescu et al. [3], and a (simplified) sample is shown in Figure 1.

There is an ongoing evaluation of different devices to express the language of AMRs, and various types of grammars have been investigated [18,15]. In this work, we consider the possibility of modelling AMRs through a sequence of in themselves simple devices; a regular tree grammar [5] followed by a sequence of tree transducers [19], and a fold operator that turns the generated tree into a graph. Henceforth, we refer to this chain as a regular tree folding (RTF). The purpose of the grammar is to generate a tree t declares the main objects and relations of the meaning representation, but treats each object as a distinct entity and leaves the question of *co-referencing* open. In the following steps, a sequence of tree transducers identifies sets of nodes in t that represent the same concept and marks these for merging with an auxiliary alphabet Δ. In the final step, the fold operator actuates the merging and turns the tree into a graph.

[⋆] Supported by the Swedish Research Council, Grant No. 621-2012-4555

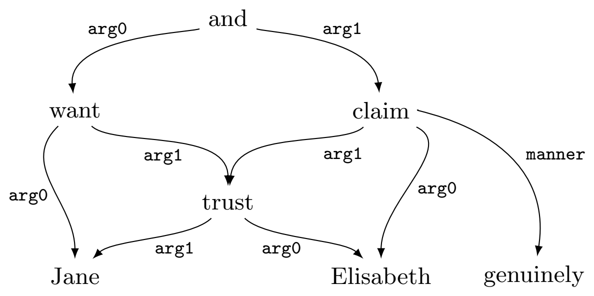


Fig. 1: An AMR for "Jane wants Elisabeth to trust her, and Elisabeth claims that she genuinely does. The outgoing edges of 'want', 'trust' and 'claim' labelled `arg0` represent the agent relation, whereas the edges labelled `arg1` represent the patient relation, and `manner` modifies the wanting.

The principal formalism for graph-based representation is hyper-edge replacement grammars (HRGs) [11]. HRGs are a natural generalisation of context-free grammars, in which nonterminals take the form of labelled hyperedges and provide restricted access to the intermediate graph. HRGs have been applied to AMR generation by Quernheim and Knight [18], and also by Chiang et al. [7]. An alternative formalism, equivalent in expressive power, is s-grammars [15]. Here, terms over a small set of operators and a finite set of elementary graphs are evaluated in the domain of node-labelled graphs. The operators are defined modulo an auxiliary alphabet, and can merge nodes with identical labels, injectively rename nodes, and clear node names.

The membership problem for HRGs and s-grammars require exponential time in general, and it is known that HRGs can generate languages for which the problem is NP-complete [1]. However, if the node degree is bounded, then solutions can be obtained in polynomial time [7,14]. Moreover, Lautemann [16] gave the following sufficient condition for an HRG G to allow polynomial-time parsing: Let k_G be the maximum number of source nodes of any nonterminal in G. For every graph g with node set V_G generated by G, and every choice $V \subseteq V_G$ of at most k_G nodes, the graph g separates into at most $O(\log |V_g|)$ connected components when the nodes in V are removed.

There is also a significant body of work on tree-to-graph transductions. Several of these are closely related to HRGs and have the same level of expressiveness. This is for instance the case for the transductions based on monadic second-order logic investigated by Courcelle [9], and for the transductions by [11] and [13] that take HRG derivation trees to their corresponding output graphs. As a result, the problem of deciding whether a graph is in the range of such a transduction is as hard as the membership problem for HRGs.

The RTFs proposed here have less expressive power than HRGs and s-grammars: It is straight-forward to simulate an RTF with an HRG, but any language containing a disconnected graph is a witness to the infeasibility of the reverse construction. The advantage of RTFs is that their membership problem is polynomial-time solvable, also in case when the node degree is arbitrarily high, and when the graphs do not fulfill Lautemann's decomposition condition.

The main contributions of this paper are: (i) the introduction of fold operators and regular tree foldings; (ii) Theorem 1 which states that interconnectedness (or *treewidth*) of the graphs generated by RTFs is bounded by the size of Δ; and (iii) Theorem 2 which states that the non-uniform version of the membership problem for RTFs is solvable in polynomial time.

2 Preliminaries

Sets, matrices, and relations. The set of non-negative integers is denoted by $\mathbb{N}$. For $n \in \mathbb{N}$, $[n] = \{1, \dots, n\}$ and $[0] = \emptyset$. The *power set* of a set S is written 2^S. A *multiset* S' is a set in which elements can have multiple instances. For $s \in S'$, $|S'|_s = n$ denotes that S' contains n instances of s.

Let N and V be finite sets, and S be any set. A $N \times V$ matrix M over a set S is a mapping $M \colon N \times V \to S$, in which N indexes *rows*, and V *columns*. The set of all $N \times V$ matrices over S is denoted $\mathbb{M}_S^{N,V}$. If $|V| = 1$, then M is an N-*vector*. For $v \in V$, we denote by $M(\bullet, v)$ the vth *column vector* of M. This is the N-vector defined by $M(\bullet, v)(A) = M(A, v)$, for every $A \in N$. The size of a N-vector $\mathbb{v}$ over $\mathbb{N}$ is $|\mathbb{v}| = \sum_{A \in N} \mathbb{v}(A)$, and the size of a $N \times V$ matrix over $\mathbb{N}$ is $\sum_{v \in V} |M(\bullet, v)|$.

Let S be a set and $\mathcal{R}, \mathcal{P} \subseteq S \times S$ relations. The *composition* of $\mathcal{R}$ and $\mathcal{P}$ is $\mathcal{R} \circ \mathcal{P} = \{(s, s'') \mid \exists s' \in S \colon (s, s') \in \mathcal{P} \wedge (s', s'') \in \mathcal{R}\}$. Similarly, the *composition* of S and $\mathcal{R}$ is $S \circ \mathcal{R} = \{s' \mid \exists s \in S \colon (s, s') \in \mathcal{R}\}$. When there is no risk for confusion, we write the composition of $\mathcal{R} \circ \mathcal{P}$ as $\mathcal{P}\mathcal{R}$. The *inverse* of $\mathcal{R}$ is $\mathcal{R}^{-1} = \{(s', s) \mid (s, s') \in \mathcal{R}\}$. The relation $\mathcal{R}$ is: The *identity relation* $\mathcal{I}_S$ on S if $\mathcal{R} = \{(s, s) \mid s \in S\}$; *reflexive* if $\mathcal{I}_S \subseteq \mathcal{R}$; *transitive* if $\mathcal{R}\mathcal{R} \subseteq \mathcal{R}$; *symmetric* if $\mathcal{R} = \mathcal{R}^{-1}$; *antisymmetric* if $\mathcal{R} \cap \mathcal{R}^{-1} \subseteq \mathcal{I}_S$; *total* if $\mathcal{R} \cup \mathcal{R}^{-1} = S^2$; and a *total order* if it is antisymmetric, reflexive, transitive, and total. The *transitive closure* of $\mathcal{R}$ is the smallest superset $\mathcal{R}^+$ of $\mathcal{R}$ such that $\mathcal{R}^+\mathcal{R} \subseteq \mathcal{R}^+$. The *transitive and reflexive closure* of $\mathcal{R}$ is $\mathcal{R}^* = \mathcal{R}^+ \cup \mathcal{I}_S$.

Graphs. To allow parallel edges, which are common in AMR, we define our graphs in terms of source and target mappings. An *alphabet* is a finite nonempty set of symbols. Let Σ be an alphabet. A (directed and labelled) *graph* over Σ is a tuple $g = (V, E, \mathit{src}, \mathit{tar}, \mathit{lab})$ consisting of: Finite sets V and E of *nodes* and *edges*, respectively; *source* and *target mappings* $\mathit{src} \colon E \to V$ and $\mathit{tar} \colon E \to V$ assigning to each edge e its source $\mathit{src}(e)$ and target $\mathit{tar}(e)$; and a *labelling* $\mathit{lab} \colon V \to \Sigma$. The *in-degree* of $v \in V$ is $|\{e \in E \mid \mathit{tar}(e) = v\}|$, and the *out-degree* of v is $|\{e \in E \mid \mathit{src}(e) = v\}|$. A node with out-degree 0 is a *leaf*. The *size* of g is $|g| = |V| + |E|$. The set of graphs with node labels in Σ is denoted $\mathbb{G}_\Sigma$.

A *path* in the graph $g = (V, E, \mathit{src}, \mathit{tar}, \mathit{lab})$ is a finite and possibly empty sequence $p = e_0, e_1, \dots, e_k$ of edges such that for each $i \in [k]$ the target of e_{i-1} is the source of e_i. Here, we say that p is a path from $\mathit{src}(e_0)$ to $\mathit{tar}(e_k)$. The path p is a *cycle* if $\mathit{src}(e_0) = \mathit{tar}(e_k)$. The graph g is *connected* if for every pair of nodes

$v, v' \in V$, there is a path from v to v' in the graph $(V, E \cup E', \mathit{src}', \mathit{tar}', \mathit{lab}')$, where E' is the smallest set that contains, for every $e \in E$, an edge e' with $\mathit{src}'(e') = \mathit{tar}(e)$, $\mathit{tar}'(e') = \mathit{src}(e)$ and $\mathit{lab}'(e') = \mathit{lab}(e)$.

The subgraph of g *induced* by the node set $V' \subseteq V$ is the graph $g[V'] = (V', E', \mathit{src}', \mathit{tar}', \mathit{lab}')$, where the edge $e \in E'$ if $\{\mathit{src}(e), \mathit{tar}(e)\} \subseteq V'$; $\mathit{src}'(e) = \mathit{src}(e)$ and $\mathit{tar}'(e) = \mathit{tar}(e)$ for every $e \in E'$, and $\mathit{lab}'(v) = \mathit{lab}(v)$ for every $v \in V'$. The *ports* of the graph $g[V']$, written $\mathit{ports}(g[V'])$, is the set of nodes $\{v \in V \setminus V' \mid \exists e \in E : (\mathit{src}(e) = v \wedge \mathit{tar}(e) \in V') \vee (\mathit{tar}(e) = v \wedge \mathit{src}(e) \in V')\}$.

A *single-rooted* graph g is a tuple $g = (V, E, \mathit{src}, \mathit{tar}, \mathit{lab}, v)$, where $v \in V$ and $(V, E, \mathit{src}, \mathit{tar}, \mathit{lab})$ is a graph. We refer to v as $\mathit{root}(g)$. The notions of paths, cycles, connectedness, etc., transfer to single-rooted graphs in the natural way.

For simplicity, our definition of graphs does not include edge labels. However, AMRs such as that in Figure 1 have unordered edges, and therefore need edge labels to distinguish between arguments. We note that such edge labels can be modelled with auxiliary nodes, as is done by Chiang et al. [8].

Trees. A *ranked alphabet* is an alphabet $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \ldots$. For every $i \in \mathbb{N}$, the symbols in Σ_i are said to have *rank* i, and we write $\sigma^{(i)}$ to indicate this. The alphabet Σ is said to be *unary* if $\Sigma_i = \emptyset$ for every $i \in \mathbb{N} \setminus \{1\}$. An *(ordered ranked) tree over* Σ is tuple $t = (g, \leq)$, where $g = (V, E, \mathit{src}, \mathit{tar}, \mathit{lab}, v)$ is a connected single-rooted acyclic graph with labels in Σ, and $\leq$ is a binary relation on E, such that (i) the out-degree of every node $v \in V$ is in $\{n \in \mathbb{N} \mid \mathit{lab}(v) \in \Sigma_n\}$, (ii) for every $v \in V$, $\leq$ is a total order on the set $\{e \in E \mid \mathit{src}(e) = v\}$, and (iii) $\mathit{src}(e) \neq \mathit{src}(e')$ implies $e \not\leq e'$. A node $u \in V$ is an *ancestor* of $v \in V$ if there is a path in t from u to v, and a *descendant* of v if v is an ancestor of u. We write the sets of ancestors and descendants of v as $\mathit{anc}(v)$ and $\mathit{desc}(v)$, respectively.

Throughout the remainder of this paper, Σ is a ranked alphabet, and $X = \{x_1, x_2, \ldots\}$ is a set of *variables*. When in the following sections a finite subset of X is used as a ranked alphabet, the variables involved are taken to have rank zero. In other words, they only label leaves.

Let $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k$ be trees over Σ with $t_i = (g_i, \leq_i)$ and $g_i = (V_i, E_i, \mathit{src}_i, \mathit{tar}_i, \mathit{lab}_i, v_i)$, such that the node sets V_i, $i \in [k]$, are mutually disjoint, and similarly for the edge sets E_i, $i \in [k]$. Intuitively, the *top-concatenation* of $t_1, \ldots, t_k$ with σ, denoted by $\sigma[t_1, \ldots, t_k]$ is the tree obtained by attaching the trees $t_1, \ldots, t_k$ as children underneath σ. More formally, $\sigma[t_1, \ldots, t_k] = ((V, E, \mathit{src}, \mathit{tar}, \mathit{lab}, v), \leq)$ where:

- $V = \{v\} \cup \bigcup_{i \in [k]} V_i$, for some $v \notin \bigcup_{i \in [k]} V_i$,
- $E = \{e_1, \ldots, e_k\} \cup \bigcup_{i \in [k]} E_i$, for some $e_1, \ldots, e_k \notin \bigcup_{i \in [k]} E_i$,
- for every $i \in [k]$, $\mathit{src}(e_i) = v$, and for every $e \in E_i$, $\mathit{src}(e) = \mathit{src}_i(e)$,
- for every $i \in [k]$ and $u \in V_i$, $\mathit{lab}(u) = \mathit{lab}_i(u)$, and $\mathit{lab}(v) = \sigma$, and
- for every $i \in [k]$, $\mathit{tar}(e_i) = \mathit{root}(t_i)$, and for every $e \in E_i$, $\mathit{tar}(e) = \mathit{tar}_i(e)$.

For every $e, e' \in E$, $e \leq e'$ if $\exists i \in [k]$ s.t. $e \leq_i e'$, or if $e = e_i$, $e' = e_j$, and $i \leq j$.

Top-concatenation is analogously defined for single-rooted graphs, so we may write $f[g_1, \ldots, g_k]$ without risk for confusion. However, in this case, the edges $e_1, \ldots, e_k$ that attach the subgraphs $g_1, \ldots, g_k$ are taken to be unordered.

We also introduce a recursive definition of (ordered ranked) trees, equivalent to that above, but more convenient for inductive arguments. Given a unary alphabet Q, and a set S of trees, we denote by $Q(S)$ the set of trees on the form $q[s]$, where $q \in Q$ and $s \in S$. The set of trees $\mathbb{T}_\Sigma(S)$ is the smallest superset of S that is closed under top-concatenation with symbols in Σ. If $S = \emptyset$, we simply write $\mathbb{T}_\Sigma$. A tree obtained by top-concatenating the trees $t_1, \ldots, t_k$ with the symbol $f^{(k)} \in \Sigma$ is written $f[t_1, \ldots, t_k]$, or simply f if $k = 0$. A *tree language* is a subset of $\mathbb{T}_\Sigma$.

Given a (partial) mapping $\theta\colon X \to \mathbb{T}_\Sigma(X)$ and a tree $t \in \mathbb{T}_\Sigma$, the application $t\theta$ is inductively defined: If $t \in X$, then $t\theta = \theta(t)$, and if $t = f[t_1, \ldots, t_n]$, for some $f \in \Sigma$, then $t\theta = f[t_1\theta, \ldots, t_n\theta]$. From here on, we write $t\theta$ as $t[x \leftarrow \theta(x) \mid x \in X']$, where X' is the subset of X on which θ is the non-identity.

Let $x \in X$ and S be a set disjunct from X. A *context over Σ indexed by S* is a tree in $\mathbb{T}_\Sigma(\{x\} \cup S)$ containing exactly one occurrence of x. The set of contexts over Σ indexed by S is written $\mathbb{C}_\Sigma(S)$. The set of *contexts* over Σ is $\mathbb{C}_\Sigma(\emptyset)$ and written $\mathbb{C}_\Sigma$. The substitution of $t \in \mathbb{T}_\Sigma(S)$ into $c \in \mathbb{C}_\Sigma(S)$ is $c[\![t]\!] = c[x \leftarrow t]$. The tree t is a *subtree* of $s \in \mathbb{T}_\Sigma(S)$ if there is a $c \in \mathbb{C}_\Sigma(S)$, such that $s = c[\![t]\!]$.

Grammars and transducers. A *regular tree grammar* (RTG) is a tuple (N, Σ, I, R), where: N is a ranked alphabet of symbols of rank 0 called *nonterminals*; Σ is a ranked alphabet of *input symbols*; $I \in N$ is the *initial nonterminal*; and R is a finite set of *rules* of the form $A \to \xi$ where $A \in N$ and $\xi \in \mathbb{T}_\Sigma(N)$.

Let G be a RTG. The *derivation relation induced by G* is the binary relation $\Rightarrow_G$ over the set $\mathbb{T}_\Sigma(N)$, such that $\phi \Rightarrow_G \varphi$ if and only if there is: a context $\beta \in \mathbb{C}_\Sigma(N)$; a nonterminal $A \in N$; and a rule $A \to \xi \in R$; and these are such that $\phi = \beta[\![A]\!]$, and $\varphi = \beta[\![\xi]\!]$. The *language generated by G* is $\mathcal{L}(G) = \{t \in \mathbb{T}_\Sigma \mid I \Rightarrow_G^* t\}$. A tree language is *regular* if it is generated by an RTG.

A *(top-down) tree transducer* (TDTT) is a tuple $T = (Q, \Sigma, \Delta, q_0, R)$, where: Q is a ranked alphabet of unary symbols called *states*; Σ and Δ are ranked alphabets of *input* and *output symbols*, respectively; q_0 is an *initial state*; and R is a finite set of *rules* of the form $q[\sigma[x_1, \ldots, x_m]] \to \xi$, where $q \in Q$, σ is an input symbol of rank $m \in \mathbb{N}$, $x_1, \ldots, x_m$ are fixed but arbitrary, pair-wise distinct, elements in X, and the right-hand side ξ is in $\mathbb{T}_\Delta(Q(\{x_1, \ldots, x_m\}))$. The transducer T is *linear* if for every $q[\sigma[x_1, \ldots, x_m]] \to \xi \in R$, the variable x_i, $i \in [m]$, occurs at most once in ξ.

Let T be a TDTT. The *derivation relation induced by T* is the binary relation $\Rightarrow_T$ over the set $\mathbb{T}_\Delta(Q(\mathbb{T}_\Sigma))$, such that $\phi \Rightarrow_T \varphi$ if and only if: There are a context $\beta \in \mathbb{C}_\Delta(Q(\mathbb{T}_\Sigma))$; a state $q \in Q$; a number $k \geq 0$ and a symbol $\sigma \in \Sigma$ of rank k; k trees $s_1, \ldots, s_k \in \mathbb{T}_\Sigma$; and these are such that $\phi = \beta[\![q[\sigma[s_1, \ldots, s_k]]]\!]$, and $\varphi = \beta[\![\xi[x_1 \leftarrow s_1, \ldots, x_k \leftarrow s_k]]\!]$, for some rule $q[\sigma[x_1, \ldots, x_k]] \to \xi$ in R. The *transduction computed by T* is a relation $[\![T]\!]$ on $\mathbb{T}_\Sigma \times \mathbb{T}_\Delta$ given by $(t, s) \in [\![T]\!]$ if and only if $q_0[t] \Rightarrow_T^* s$, for every $(t, s) \in \mathbb{T}_\Sigma \times \mathbb{T}_\Delta$.

Example 1 (Composition). Consider the RTG $G = (\{I, A\}, \Sigma, I, R)$ where $\Sigma = \{a^{(0)}, f^{(1)}\}$ and $R = \{I \to f[A], A \to f[A], A \to a\}$. Clearly, $\mathcal{L}(G)$ contains monadic trees over f and a. Now let $T = (Q, \Sigma, \Delta, q_0, R')$, where $Q = \{q_0, q_1\}$,

$\Delta = \{a^{(0)}, f^{(1)}, g^{(2)}\}$, and R' is:

$$\begin{array}{ll} \{\ q_0[f[x]] \to f[q_0[x]], & q_0[f[x]] \to f[q_1[x]],\ q_0[a] \to a, \\ \ \ q_1[f[x]] \to g[q_1[x], q_1[x]],\ q_1[a] \to a & \}\ . \end{array}$$

Applied to trees in $\mathcal{L}(G)$, the transducer T non-deterministically chooses a node v, and transform the subtree rooted at v into a balanced binary tree.

3 Tree-to-graph transductions and AMR modelling

The grammars and transducers recalled in Section 2 create and manipulate trees, but in AMRs we find node sharing and sometimes even cycles. To accommodate these structures, we introduce an operator that folds trees into graphs by merging nodes labelled with an auxiliary alphabet. The operator is applied recursively, top-down, taking the input tree as a term over a domain of graphs. The term is then evaluated bottom-up into a set of output graphs.

Definition 1 (The fold operator F). *Let Σ be a ranked alphabet, and Δ a unary ranked alphabet, disjoint from Σ. We denote by $\Sigma \otimes \Delta$ the ranked alphabet $\Sigma \times \Delta$, such that for every $i \in \mathbb{N}$, $(\Sigma \otimes \Delta)_i = \Sigma_i \times \Delta$. We write Γ for the ranked alphabet $\Sigma \cup (\Sigma \otimes \Delta) \cup \Delta$.*

We associate with each $\delta \in \Delta$ a relation $[\![\delta]\!]\colon \mathbb{G}_\Gamma \to \mathbb{G}_\Gamma$. When applied to a single-rooted graph $g = (V, E, src, tar, lab, v) \in \mathbb{G}_\Gamma$, $[\![\delta]\!]$ merges the set of nodes $V_\delta = \{v \in V \mid \exists \sigma \in \Sigma : lab(v) = \langle \sigma, \delta \rangle\}$ into a node u, and then nondeterministically assigns u a label in $\{\sigma \mid \exists v \in V_\delta : lab(v) = \langle \sigma, \delta \rangle\}$. If $root(g) \in V_\delta$, then $root([\![\delta]\!](g)) = u$; otherwise $root([\![\delta]\!](g)) = root(g)$.

The fold operator *F (over Σ with folding symbols in Δ) is a relation $\mathbb{T}_\Gamma \to \mathbb{G}_\Gamma$, defined for every tree $t = \gamma[t_1, \ldots, t_k] \in \mathbb{T}_\Gamma$ by:*

$$F(t) = \begin{cases} \bigcup_{s_1 \in F(t_1)} [\![\gamma]\!](s_1) & \text{if } \gamma \in \Delta \text{ (so } k = 1\text{)}, \\ \bigcup_{s_1 \in F(t_1), \ldots, s_k \in F(t_k)} \gamma[s_1, \ldots, s_k] & \text{otherwise.} \end{cases}$$

From here on, let Σ, Δ, and Γ be as above. The fold operator F extends to sets of trees in the expected way: For $L \subseteq \mathbb{T}_\Gamma$, $F(L) = \bigcup_{t \in L} F(t)$.

Example 2 illustrates how the fold operator turns trees into – potentially cyclic – graphs.

Example 2. Recall the RTG G and TDTT T of Example 1, and consider Figure 2. The tree in (a) is in $\mathcal{L}(G)$, and that in (b) is in $\mathcal{L}(G) \circ [\![T]\!]$. Now, let $\Delta = \{\alpha, \beta\}$ and let T' be a TDTT with input alphabet Σ and output alphabet Γ. The transducer T' places the symbols in Δ on top of the tree, and then randomly decides for each node labelled $\sigma \in \Sigma$ whether its label should be replaced by one in $\{\sigma\} \otimes \Delta$. The tree t in (c) is thus in $\mathcal{L}(G) \circ [\![T]\!] \circ [\![T']\!]$. When the evaluation of $F(t)$ has reached β, the result is the graph in (d), and when the entire tree has been evaluated, the graph in (e).

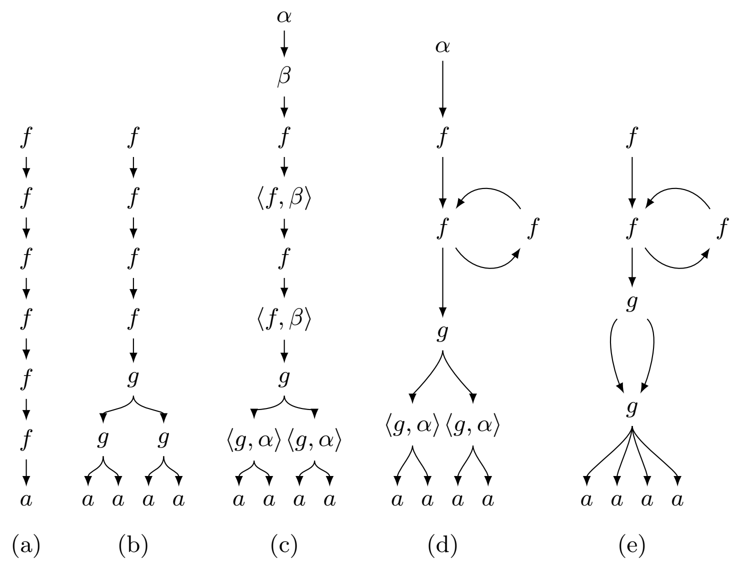


Fig. 2: The generation of a tree t (a–c), and the application of F to t (d–e).

Definition 2 (Regular Tree Folding). *A regular tree folding (RTF) R over Γ is a tuple (G, T, F), where G is an RTG, $T = T_1, \ldots, T_n$ is a (possibly empty) sequence of TDTTs, and a F is a fold operator. All devices are over the common alphabet Σ, and F additionally has folding symbols in Δ. The graph language computed by R is the composition $\mathcal{L}(G) \circ [\![T_1]\!] \circ \cdots \circ [\![T_n]\!] \circ F \subseteq \mathbb{G}_\Sigma$, written $\mathcal{L}(G \circ T_1 \circ \ldots \circ T_n \circ F)$ for short.*

Let us now apply the RTF $R = (G, T, F)$ to the generation of AMRs. The grammar G generates a tree t with labels in {Jane, Elisabeth, Mary, Catherine, Lydia, want, claim, trust, like, genuinely}. In the next step, the transducers in T decide what nodes should be merged to introduce co-referencing. One possible decoration of t is shown in Figure 3. After folding, the nondeterministic process may arrive at the AMR in Figure 1, but could equally well produce an AMR for "Mary wants Lydia to like her, and Lydia claims she genuinely does".

A simpler, alternative, definition of the fold operator avoids node labels in Δ, and simply merges nodes with labels in $\Sigma \times \Delta$, whenever these labels agree in their second argument. An advantage of our definition is that the node labels in Δ act as scoped operators and restrict foldings to limited subtrees. This means that even though the alphabet Δ is finite, the nodes can be divided into an unbounded number of groups, each merged independently from the others. Consider the tree in Figure 4 (a), which is decorated with two instances of the auxiliary symbol α. After the application of F, the result is an AMR (Figure 4 (b)) concerning two different girls, one with more confidence, one with less. If there had instead been a single root labelled α, then the AMR would describe a girl with conflicting emotions. Yet another possibility is shown in Figures 4 (c)

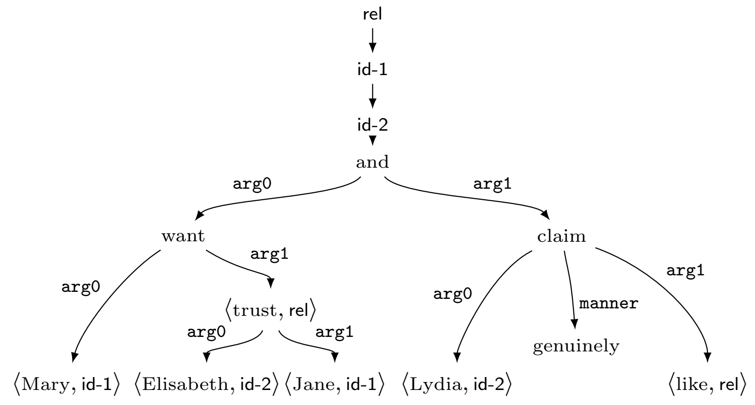


Fig. 3: A tree-based meaning representation, annotated with auxiliary symbols (id-1, id-2, and rel) to instruct the fold operator F.

and (d) in which two folding symbols are used. The AMR now tells us that one girl trusts another, but is rewarded with doubt.

4 Expressiveness and computational complexity

RTFs allow for the generalisation of finite AMR corpora to infinite graph languages. However, if these languages permit infinite node degree, then their constituent graphs will share the following structural property:

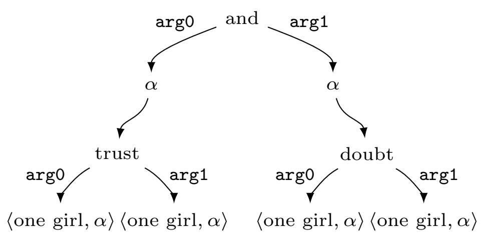
Observation 1 *Let $L \subseteq \mathbb{T}_\Gamma$ be a tree language, such that the maximal degree of any node in any tree in L is r, then for every $g \in F(L)$ the following holds: If v is a node in g with degree m, then there is a path in g of length $\lceil \log_r m \rceil$.*

Intuitively, this is due to the generation being based on trees with bounded rank: to create a node with high degree, a large set of nodes with small degree must be merged, and the generation of this set requires a tall tree. Since the fold operator preserves edges, the long paths that must necessarily exist in such a tree remain. This limitation can be avoided if the trees are generated by unranked devices, and we view this as a natural continuation of the present work.

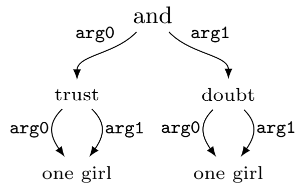
Another structural property of the graph languages produced by RTFs is that they, similar to the languages produced by HRGs, have bounded *tree width.*

Definition 3. *Let $g = (V, E, src, tar, lab)$ be a graph in $\mathbb{G}_\Sigma$. A* tree decomposition *of a graph g is a tree $t = (V_t, E_t, src_t, tar_t, lab_t, v_t) \in \mathbb{T}_{2^V}$ such that:*

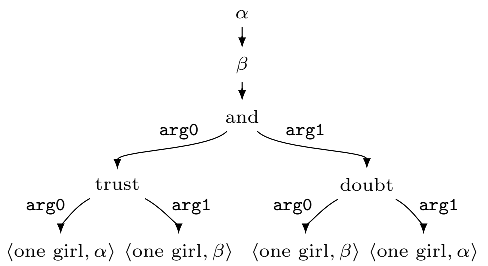
1. *For every $v \in V$, there is a $u \in V_t$ such that $v \in lab_t(u)$.*
2. *For every $e \in E$, there is a $u \in V_t$ such that $\{src(e), tar(e)\} \subseteq lab_t(u)$.*
3. *For every $v \in V$, the subgraph of t induced by $\{u \in V_t \mid v \in lab_t(u)\}$ is connected.*



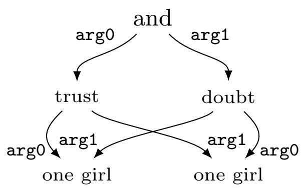
(a) A tree-based semantic representation relating to girls, trust, and doubt.



(b) The tree in (a) after folding. The resulting graph encodes the meaning "One girl trusts herself, and one girl doubts herself."



(c) The same semantic representation as in Subfigure (a) above, with an alternative decoration of auxiliary symbols.



(d) The tree in Subfigure (c) after folding. The meaning now becomes "One girl trusts another, who in turn doubts her."

Fig. 4: The auxiliary symbols decide what output tree F will produce.

The width *of t is* $(\max_{u \in V_t} |lab_t(u)|) - 1$*, and the* treewidth *of g is the minimum of the widths of its tree decompositions.*

For instance, a tree has treewidth 1, a cycle has treewidth 2, and for every $k \in \mathbb{N} \setminus \{0\}$, the k-complete graph K_k has treewidth $k - 1$.

Once a fold operator has been applied to a set of nodes, these cannot be included in later mergings. This limits the degree of interconnectedness that can be achieved in the generated graphs. Before we continue our investigation, we introduce some additional notation to make the argumentation more precise.

Let $t = (V_t, E_t, src_t, tar_t, lab_t, v_t) \in \mathbb{T}_\Gamma$, and let Δ_t denote the set $\{\delta \in \Delta \mid \exists v \in V_t : lab_t(v) = \delta\}$. For every node $v \in V_t \setminus \{v_t\}$, the parent of v, written $parent(v)$, is the unique ancestor of v from which there is a path to v of length exactly 1. For every $v \in V_T$, we denote by t/v the subtree of t rooted at v. Note that $t/v_t = t$.

Let P be a set, $p \in P$, and s a tree with node set V_s and node labels in 2^P. For every $P' \subseteq P$, the function $replace(s, P', p)$ replaces every occurrence of an element in P' in the node labels of s by p. This means that after an application of $replace(s, P', p)$, every set that labels a node in s is free from elements in P', and may have become smaller (if it originally contained more than one element in P'), but not larger.

For every $k \in \mathbb{N}$ and alphabet $\Sigma = \{\sigma\}$, the *k-complete graph* (over Σ) is $K_k = (V, E, src, tar, lab)$, where $V = \{v_1, \ldots, v_k\}$; $E = \{\langle v_i, v_j\rangle \mid v_i, v_j \in V, i < j\}$; $src(\langle v_i, v_j\rangle) = v_i$ and $tar(\langle v_i, v_j\rangle) = v_j$ for every $\langle v_i, v_j\rangle \in E$; and $lab(v) = \sigma$ for every $v \in V$.

Lemma 1. *For every $t \in \mathbb{T}_\Gamma$, the treewidth of $F(t)$ is at most $|\Delta_t| + 1$, and this is a tight upper bound.*

Proof. Let $t = (V_t, E_t, src_t, tar_t, lab_t, v_t) \in \mathbb{T}_\Gamma$. To prove Lemma 2, we construct a tree-decomposition for $F(t) = g = (V_g, E_g, src_g, tar_g, lab_g)$ of width $|\Delta_t| + 1$.

The construction starts out from a tree $s = (V_s, E_s, src_s, tar_s, lab_s, v_s)$, where each component is initially identical to the corresponding component in t, except for lab_s, which is given by $lab_s(v_t) = \{v_t\}$ and $lab_s(v) = \{v, parent(v)\}$ for every $v \in V_t \setminus \{v_t\}$. It is easy to verify that s is a tree-decomposition for t of width 1.

To relate the trees t and s, we use the mapping $\varphi : V_t \to V_s$ that takes each node $v \in V_t$ to its copy in V_s. The function φ is thus initially a bijection, but as we start to restructure s, the function will cease to be injective.

The construction process now proceeds incrementally, paralleling the bottom-up evaluation of $F(t)$. Before we describe the modifications made to s, we give a number of invariants that will hold at the completion of the evaluation at node $v \in V_t$. The first of these is simply that:

1. The tree $s/\varphi(v)$ is a tree-decomposition of $F(t/v)$ of width at most $\left|\Delta_{t/v}\right| + 1$.

To keep track of the nodes in $V_{F(t/v)}$ that have previously been cleared of labels in $\{\delta\} \cup (\Sigma \otimes \{\delta\})$ due to a merge with respect to $\delta \in \Delta$, we compute a mapping $H_v : \Delta \to 2^{V_{F(t/v)}}$ for every $v \in V_t$. We then define $J_{\varphi(v)}(\delta)$ to be the nodes in $s/\varphi(v)$ with labels that intersect $2^{H_v(\delta)}$, i.e., $J_{\varphi(v)}(\delta) = \{u \in V_{s/\varphi(v)} \mid lab_{s/\varphi(v)}(u) \cap H_v(\delta) \neq \emptyset\}$.

At the completion of the evaluation at node $v \in V_t$, it will be the case that:

2. For every $\delta \in \Delta$, $(s/\varphi(v))[J_{\varphi(v)}(\delta)]$ is a set of subtrees of s.
3. For every $\delta \in \Delta$ and $u \in H_v(\delta)$, we have $lab_{F(t/v)}(u) \in \Sigma \cup (\Sigma \otimes (\Delta \setminus \{\delta\}))$.

Note that before any evaluation step has been made, Invariant 1–3 hold trivially.

Assume now that the evaluation of F has reached a node $v \in V_t$. There are two cases:

1. $t(v) \in \Gamma \setminus \Delta$. Here, s is kept unchanged and for every $\delta \in \Delta$, we let $H_v(\delta) = \cup_{u \in children(v)} H_u(\delta)$. This means in particular that $H_v(\delta) = \emptyset$ if v is a leaf. It is easy to verify that Invariant 1–3 continue to hold.
2. $t(v) = \delta$ for some $\delta \in \Delta$. Since Δ is a unary alphabet, v has a unique child u, and $\varphi(v)$ has the unique child $\varphi(u)$ in s. Let P be the set of nodes in $F(t/u)$ that have labels in $\Sigma \otimes \{\delta\}$. Due to the symbol δ labelling v, the set P will be merged into a single node w with a label in Σ. To reflect this change, we update s as follows:

(a) First, we invoke *replace*$(s/\varphi(u), P, w)$ to replace every occurrence of a node in P in the labels of $s/\varphi(u)$ by w. Intuitively, this implements to the actual merging, and in combination with Step 2 (b) below guarantees that Invariant 1 continues to hold. Invariant 3 ensures that no node in $J_{\varphi(u)}(\delta)$ is affected, as the nodes in $H_u(\delta)$ have previously been cleared of labels in $\{\delta\} \cup (\Sigma \otimes \{\delta\})$.
(b) Second, the node w is added to the label of every node in $s/\varphi(u)$ outside of $J_{s/\varphi(u)}(\delta)$. Since by Invariant 2, $(s/\varphi(u))[J_{s/\varphi(u)}(\delta)]$ is a set of subtrees, the nodes outside this set will form a connected (though possibly empty) subgraph of $s/\varphi(u)$. This guarantees that Invariant 2 will continue to hold and combines with Step 2 (a) to ensure Invariant 1. Since the label of w is in Σ, the addition of w will not violate Invariant 3 for some other $\delta' \in \Delta$.
(c) Third, we replace the subtree $s/\varphi(v)$ in s by $s/\varphi(u)$, so $\varphi(v)$ now points to the root of $s/\varphi(u)$.
(d) Finally, we let $H_v(\delta)$ be the entire set of nodes of $F(t/v)$, so $J_{s/\varphi(v)}(\delta)$ will be all of $s/\varphi(v)$. As the set of node labels of $F(t/v)$ is disjoint from $\{\delta\} \cup (\Sigma \otimes \{\delta\})$, Invariant 3 continues to hold.

When the evaluation of t has reach the root v_t, Invariant 1 ensures that s is a tree-decomposition of $F(t)$ of width at most $|\Delta_t| + 1$.

The bound is tight: When $|\Delta| = n$, it is straight-forward to construct a grammar G such that $\mathcal{L}(G \circ F) = \{K_{n+2}\}$. Since K_m has treewidth $m-1$, K_{n+2} has treewidth $n+1$. □

Lemma 1 immediately yields Theorem 1.

Theorem 1. *The treewidth of every $g \in \mathcal{L}(R)$ is less or equal to $|\Delta| + 1$.*

The *folding depth* of a tree $t \in \mathbb{T}_\Gamma$ is the maximum number of symbols in Δ on a path from the root to a leaf of t.

Lemma 2. *Let $t \in \mathbb{T}_\Gamma$ be a tree with folding depth k, then the treewidth of $F(t)$ is at most $k+1$.*

Proof. Let p be a leaf-to-root path through g on which there are k symbols in Δ. We prove by induction on the nodes $v_0, \ldots, v_m$ of p, starting at the leaf v_0, that after the recursive evaluation of F has been applied to node v_i, $i \in [m]$, the treewidth of the resulting graph equals the number of symbols in Δ on the sub-path $v_0, \ldots, v_i$.

For the base case, consider the situation at v_0. Since Δ is a unary alphabet, $lab_t(v_0) \notin \Delta$. After the evaluation of F on v_0, there have been $n = 0$ symbols in Δ, and the resulting graph is a tree of size 1 with treewidth $n + 1 = 1$. For the inductive case, assume that the computation has reached the node v_i, that we have processed k' symbols in Δ, and that Theorem 1 holds for every node below v_i. If $lab_t(v_i) \notin \Delta$, then the graphs below v_i are top-concatenated with $lab_t(v_i)$ and the computation continues. As top-concatenation of disjoint graphs does not increase the treewidth, the statement remains true.

If $lab_t(v_i) \in \Delta$, then a set of nodes N in the graph h below v_i is to be merged into a single node. The merge can be done by (i) deleting the subgraph induced by N, (ii) removing the set E of dangling edges in the remaining part h' of h, but remembering their attachment points, (iii) adding a new node u to h', and finally (iv) adding the edges E back in again, now attached at one end to u. By the induction hypothesis, the treewidth of h is at most $k'+1$, and since h' is a subgraph of h, the same is true for h'. The addition and attachment of a new node can cause the treewidth to increase by at most 1. To see this, note a tree decomposition for the new graph can be created by simply adding the node u to every label set. After the application of the $(k'+1)$th $[\![\delta]\!]$, where $\delta \in \Delta$, the treewidth is at most $k'+2$, so the statement of Theorem 1 remains valid. □

The remainder of this section is dedicated the proof of Theorem 2 which states that the membership problem for RTFs is solvable in polynomial time.

Problem 1. Let $R = (H, T, F)$ be an RTF over Γ. Given $g \in \mathbb{G}_\Gamma$, is $g \in \mathcal{L}(R)$?

We outline a decision algorithm for Problem 1 and show that it runs in polynomial time. The proof argument is standard: it is an induction on the tree decomposition of the graph g. The proof hinges on edges in g being unordered, which significantly limits the size of the search space.

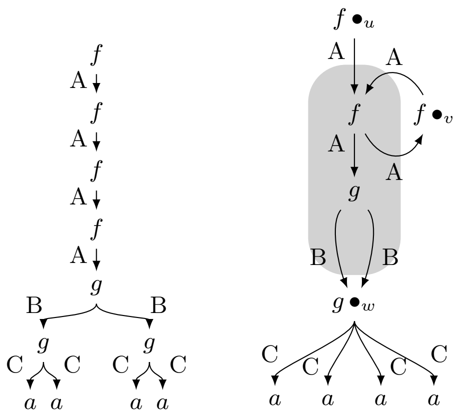
Since the regular tree languages are closed under application of linear top-down tree transducers [2, Proposition 3], and the size of the generating devices are taken as constant, we can assume without loss of generality the existence of an RTG $G = (N, \Gamma, I, R)$ such that $\mathcal{L}(G) = \mathcal{L}(H \circ T_1 \circ \ldots \circ T_n)$, where $T_1, \ldots, T_n = T$. For technical reasons, we also assume that G is in the normal form where every right-hand side has height 1, and the nonterminal I does not occur on any right-hand side [12, Theorem 3.22].

The main task of the algorithm is to decide whether the nodes in g can be "unmerged" to recover a tree t in $\mathcal{L}(G)$, such that $g \in F(t)$. It traverses g in such an order, that the processed part of g is connected to a limited set of nodes in the unprocessed part. The size of the set depends on the treewidth of g.

From here on, let $g = (V, E, src, tar, lab)$ be a graph in $\mathbb{G}_\Gamma$ with treewidth k. To avoid a combinatorial explosion, the algorithm represents every subgraph h of g as a relation $\mathcal{R}_h$. This relation describes how the part of the input tree t that gave rise to h would have interacted in a derivation with those parts of t that ended up outside of h. The idea is illustrated in Figure 5: Subfigure (a) shows the tree in Figure 2 (b), together with a *witness annotation* for g.

Definition 4 (Witness annotation). *An edge labelling $lab_w : E \to N$ is a* witness annotation *for g if it can be constructed as follows: Let $t \in \mathcal{L}(G) \cap F^{-1}(g)$ and $\pi : V_t \to N$ be a node labelling that describes a successful derivation of t by G (i.e., π is an accepting run of G on t, when G is viewed as an automaton). Label every $e \in E_t$ by $\pi(tar_t(e))$ and then apply F to fold t into g. After the folding, $E = E_t$, so the result is an edge labelling of E.*

In Subfigure (b), the tree has been folded, and a subgraph h marked out against a grey background. The same figure also shows the ports u, v, and w



(a) (b)

$$\begin{bmatrix} & u & v & w \\ A & 1 & 1 & 0 \\ B & 0 & 0 & 0 \\ C & 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} & u & v & w \\ A & 0 & 1 & 0 \\ B & 0 & 0 & 2 \\ C & 0 & 0 & 0 \end{bmatrix}$$

(c) (d)

Fig. 5: Subfigure (a) shows the tree t in Figure 2 (b), annotated with nonterminals according to a derivation in G. In (b), this witness annotation is projected onto $[\![F]\!](t)$. To represent the subgraph h (gray background), we compute the pair of matrices $(M, M') \in \mathcal{R}_h$ shown in (c) and (d): M encodes that h receives one A from each of u and v, and M' that h propagates one A to v, and two Bs to w.

that connect h to the rest of g. The algorithm abstracts away from h's internal structure, and only remembers what nonterminals are received from, and transmitted to, each node in *ports*(h). In doing so, it makes use of the tuple *attach* which counts how many edges h has to each node in *ports*(h).

Definition 5 (Attach). *Let $h = (V_h, E_h, src_h, tar_h, lab_h)$ be a subgraph of g, and let $U = ports(h)$. The tuple $attach(h) = (\mathbb{v}, \mathbb{v}')$, where $\mathbb{v}, \mathbb{v}' \in \mathbb{N}^U$, is such that for every $u \in U$, $\mathbb{v}(u) = |\{e \in E \mid src(e) = u, tar(e) \in V_h\}|$, and $\mathbb{v}'(u) = |\{e \in E \mid src(e) \in V_h, tar(e) = u\}|$. For $\mathbb{v} \in \mathbb{N}^U$, we denote by $M_{\mathbb{N}}^{N,U,\mathbb{v}}$ the set of $N \times U$ matrices of natural numbers, such that for every $u \in U$, the column vector indexed by $u \in U$ has size at most $\mathbb{v}(u)$.*

The matrices in $M_{\mathbb{N}}^{N,U,\mathbb{v}}$ are useful to remember how many nonterminals of each type are passed to (or received from) each port u in U, when we know that there are $\mathbb{v}(u)$ edges between h and u.

We can now described the interaction between h and g as a binary relation $\mathcal{R}_h$ on matrices. Before we give the formal definition, we introduce some additional notation: If $t \in \mathcal{L}(G) \cap F^{-1}(g)$, then we denote by $F_t^{-1}(h)$ the subgraph of t mapped by F to h, and by $F_t^{-1}(u)$, where $u \in V_g$, the set of nodes in V_t that were merged by F to form u.

Definition 6 (The relation $\mathcal{R}_h$). *Let h be a subgraph of g. A pair of matrices $(M, M') \in M_{\mathbb{N}}^{N,U,\mathbb{v}} \times M_{\mathbb{N}}^{N,U,\mathbb{v}'}$ is in $\mathcal{R}_h$ if and only if the following holds: There is*

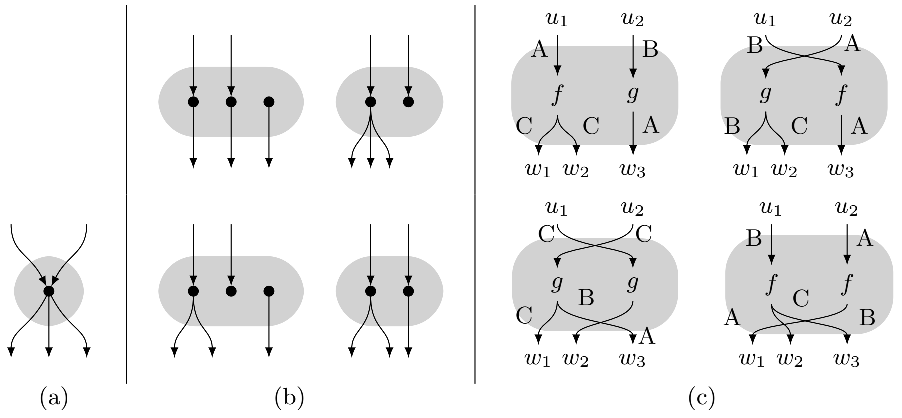


Fig. 6: The node in Subfigure (a) can be "unmerged" in several ways, including those in Subfigure (b). For each resulting configuration, there are alternative ways to connect the edges to ports, assign node labels, and add a witness annotation with nonterminals. Subfigure (c) shows four variants based on the lower-right configuration in Subfigure (b).

a $t \in \mathcal{L}(G) \cap F^{-1}(g)$, and a witness annotation ρ such that for every $A \in N$ and $u \in U$, the subgraph $F_t^{-1}(h)$ has $M(A, u)$ incoming edges labelled by ρ with the nonterminal A from the nodes in $F_t^{-1}(u)$, and $M'(A, u)$ outgoing edges labelled by ρ with the nonterminal A to the nodes in $F_t^{-1}(u)$.

For example, Figure 5 (c) and (d) show the pair (M, M') of matrices included in $\mathcal{R}_h$, due to the combination of input tree, witness annotation, and folding shown in (a) and (b). The relation $\mathcal{R}_h$ is computed inductively. Lemma 3 gives us the complexity of the base case, where h consists of a single node.

Lemma 3. *Let $G = (N, \Gamma, I, R)$ be an RTG, $v \in V$, and $r = \max\{r \in \mathbb{N} \mid \Sigma_r \neq \emptyset\}$. If the subgraph $h = g[v]$ is such that $|ports(h)| = k \in \mathbb{N}$ and $attach(h) = (\mathbb{v}, \mathbb{v}')$, then $\mathcal{R}_h$ can be computed in time $m^{O((k|G|)^r)}$, where $m = |\mathbb{v}| + |\mathbb{v}'|$.*

Proof. If g is the result of folding a tree t into a graph, then every node v with l incoming edges must be the result of merging either l or $l + 1$ nodes, depending on whether the root of the input tree is in this set. Let us now decompose h into a set of nodes, each with in-degree at most 1 and out-degree at most r. In this step, we only need to remember how many nodes of each rank there are, and do not distinguish between nodes of the same rank. Under these assumptions, there are $O(m^r)$ possible configurations. This step is illustrated in Figure 6 (a) and (b).

For each such configuration, we can select node labels, add nonterminals to the edges, and attach edges to the k ports $O\big(m^{(k|N||\Sigma|)^r}\big)$ different ways (see Figure 6 (c)). That fact that m does not appear in the exponent is due to the fact that identically labelled edges to the same port cannot be distinguished, as the edges are not ordered. There are thus $O\big(m^{r+(k|N||\Sigma|)^r}\big)$, or $m^{O((k|G|)^r)}$ configurations to consider. This function is polynomial in the size of h, and the local validity of each configuration can be checked against the rules R in $O(m\,|R|)$. If the configuration respects G, then a matrix representation (M, M') of the configuration is added to $\mathcal{R}_h$. After this factor has been included, the overall complexity remains within $m^{O((k|G|)^r)}$. □

If $\mathcal{R}_{h_i}$ is known for l disjoint subgraphs $h_i = g[V_i], i \in [l]$, $V_i \subseteq V$, then the relation $\mathcal{R}_h$ for the subgraph $h = g[\cup_{i\in[l]} V_i]$ can be efficiently computed:

Lemma 4. *Given l disjoint subgraphs $h_i = g[V_i]$, $i \in [l]$, $V_i \subseteq V$, such that $|ports(h_i)| \leq k$, $attach(h_i) = (\mathbb{v}_i, \mathbb{v}'_i)$, and $\mathcal{R}_{h_i}$ is known. The relation $\mathcal{R}_h$ for $h = g[\cup_{i\in[l]} V_i]$ is computable in $m^{O(l(k|G|)^r)}$ steps, where $m = \sum_{i=1}^{l} |\mathbb{v}_i| + |\mathbb{v}'_i|$.*

Proof. To derive $\mathcal{R}_h$, the algorithm enumerates all consistent assignments of nonterminals to the edges between between h_i and $ports(h_i)$, $i \in [l]$. The number of elements $\langle (M_1, M'_1), \cdots, (M_l, M'_l)\rangle \in \mathcal{R}_{h_1} \times \ldots \times \mathcal{R}_{h_l}$ is $\prod_{i=1}^{l} |\mathcal{R}_{h_i}|$, which is in $\prod_i^l (|\mathbb{v}_i| + |\mathbb{v}'_i|)^{O((k|G|)^r)}$, which in turn is in $\prod_i^l m^{O((k|G|)^r)}$, and so in $m^{O(l(k|G|)^r)}$. For each such combination, we verify that for every $i, j \in [l]$, $i \neq j$, the following holds: (1) For every $v_i \in V_i$ and $v_j \in V_j$ such that there is an $e \in E$ with $src(e) = v_i$ and $tar(e) = v_j$, we have $M'_i(\bullet, v_j) = M_j(\bullet, v_i)$. If Condition 1 is true, then we note that $ports(h) = (\cup_{i\in[k]} ports(h_i)) \setminus \cup_{i\in[k]} V_i$, and let $M(\bullet, u) = \sum_{i\in[k]} M_i(\bullet, u)$ and $M'(\bullet, u) = \sum_{i\in[k]} M'_i(\bullet, u)$, for every $u \in ports(h)$, and finally add the tuple (M, M') to the relation $\mathcal{R}_h$. The verification of Condition 1 is in $O\big(k^2 l^2\, |G|\big)$, and as $|G|$ is a constant (because we are looking at the non-uniform version of the membership problem), the whole operation is in $m^{O(l(k|G|)^r)}$. □

When combined, Lemma 3 and 4 yield the main result of this work:

Theorem 2. *For every RTF R, Problem 1 is decidable in polynomial time in the size of the input graph.*

Proof. Let G be an RTG such that $\mathcal{L}(G) = \mathcal{L}(H \circ T_1 \circ \ldots \circ T_n)$, let $g = (V, E, src, tar, lab)$, and let $s = (V_s, E_s, src_s, tar_s, lab_s, v_s)$ be a tree decomposition of g with minimum width. For every $v \in V_s$, we address subsets of V with the functions:

$$up(v) = \bigcup\nolimits_{v' \in (anc(v)\setminus\{v\})} lab_s(v') \ , \qquad mid(v) = up(v) \cup lab_s(v) \ , \text{ and}$$

$$down(v) = \bigcup\nolimits_{v' \in desc(v)} lab_s(v') \setminus mid(v) \ .$$

To decide whether $g \in \mathcal{L}(G \circ F)$, the algorithm iterates over the nodes of s, working from the leaves upwards. At the node v it computes the relation $\mathcal{R}_h$, where $h = g[down(v)]$. Let $v \in V_s$ be an here-to unprocessed node, such that every child $v_i \in V_s$, $i \in [c]$, $c \in \mathbb{N}$, of v has been processed. If $c > 1$, we pick a pair v_i, v_j, $i \neq j$, and compute $\mathcal{R}_{h'}$, where $h' = g[down(v_i) \cup down(v_j)]$, from the known relations $\mathcal{R}_{g[down(v_i)]}$ and $\mathcal{R}_{g[down(v_j)]}$. This process is repeated until we have a single subgraph $h'' = g[\cup_{i\in[c]} down(v_i)]$ below v.

The next step addresses v itself. We compute $\mathcal{R}_{g[u]}$ for every $u \in U = lab_s(v) \setminus up(v)$ and combine these relations with $\mathcal{R}_{h''}$ to obtain $\mathcal{R}_h$. Due to the structure of s, for every $U' \subseteq U$,

$$ports(g[U' \cup \bigcup_{i\in[c]} down(v_i)]) \subseteq lab_s(v) \ ,$$

so each step involves at most $k+1$ subgraphs, attached to at most k nodes.

Let us now consider the overall amount of work. Each node v of g is converted into a relation $\mathcal{R}_{g[v]}$, with $O(|g|)$ attaching edges to $ports(g[v])$. By Lemma 3, the total work is in $|g|^{O((k|G|)^r)}$. Smaller relations are combined into larger at most $|V|+|s|$ times, and the size of s is in $O(|g|^k)$. Each combination involves at most $k+1$ relations and $|g|$ attaching edges, so by Lemma 4, the work needed per combination is in $|g|^{O((k+1)(k|G|)^r)}$ and the total work for all steps in $|g|^{O\left((k|G|)^{r+1}\right)}$, that is, polynomial in $|g|$. □

5 Conclusion

We have presented a fold operator that restructures trees into graphs. By applying the operator to a regular tree language, we can produce the type of node-sharing and cyclic dependencies that are needed to model AMRs. A regular tree folding, i.e., a regular tree grammar together with the fold operator, is strictly less expressive than HRGs and s-grammars, as it cannot produce graphs that are less connected than a tree. However, in contrast to these alternative devices, the associated membership is solvable in polynomial-time.

References

1. Stefan Arnborg, Derek Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.
2. Brenda S Baker. Tree transducers and tree languages. *Information and Control*, 37(3):241–266, 1978.
3. Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract Meaning Representation (AMR) 1.2 specification. Technical report, Information Science Institute, University of Southern California, CA, USA.
4. Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract Meaning Representation for Sembanking. In *Proc. 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria, 2013. ACL.
5. Walter S. Brainerd. The minimalization of tree automata. *Information and Control*, 13(5):484–491, 1968.

6. Danqi Chen and Christopher Manning. A Fast and Accurate Dependency Parser using Neural Networks. In *Proc. 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), Doha, Qatar*, pages 740–750. ACL, 2014.
7. David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. Parsing graphs with hyperedge replacement grammars. In *Proc. 51st Meeting of the ACL*, 2013.
8. David Chiang, Frank Drewes, Daniel Gildea, Giorgio Satta, and Adam Lopez. Weighted DAG Automata for Semantic Graphs. Unpublished, 2017.
9. Bruno Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75, 1994.
10. Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard M Schwartz, and John Makhoul. Fast and Robust Neural Network Joint Models for Statistical Machine Translation. In *Proc. 52nd Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 1370–1380, Baltimore, Maryland, 2014. ACL.
11. Frank Drewes, Hans-Jörg Kreowski, and Annegret Habel. Hyperedge replacement graph grammars. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 95–162. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.
12. Joost Engelfriet. Tree automata and tree grammars. Lecture Notes on Formal Languages and Automata Theory, see `http://arxiv.org/abs/1510.02036`, 1975.
13. Joost Engelfriet and Heiko Vogler. The equivalence of bottom-up and top-down tree-to-graph transducers. *Journal of Computer and System Sciences*, 56(3):332 – 356, 1998.
14. Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. Graph parsing with s-graph grammars. *Proc. 53rd Annual Meeting of the ACL and the 7th International Joint Conference on NLP*, 1:1481–1490, 2015.
15. Alexander Koller. Semantic construction with graph grammars. In *Proc. 14th International Conference on Computational Semantics (IWCS)*, London, 2015.
16. Clemens Lautemann. Tree automata, tree decomposition and hyperedge replacement. In Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science, 4th International Workshop, Bremen, Germany, March 5-9, 1990, Proceedings*, volume 532 of *Lecture Notes in Computer Science*, pages 520–537. Springer, 1990.
17. Christopher Manning. Part-of-speech tagging from 97\% to 100\%: is it time for some linguistics? *Computational Linguistics and Intelligent Text Processing*, pages 171–189, 2011.
18. Daniel Quernheim and Kevin Knight. DAGGER: A Toolkit for Automata on Directed Acyclic Graphs. In *Proc. 10th Int. Workshop Finite-State Methods and Natural Language Processing*, pages 40–44. ACL, 2012.
19. William C Rounds. Trees, Transducers, and Transformations. Ph.D. dissertation, 1968.