A Feature Structure Algebra for FTAG

Alexander Koller
Saarland University
koller@coli.uni-saarland.de

Abstract

FTAG, the extension of TAG with feature structures, lags behind other feature-based grammar formalisms in the availability of efficient chart parsers. This is in part because of the complex interaction of adjunction and unification, which makes such parsers inconvenient to implement. We present a novel, simple algebra for feature structures and show how FTAG can be encoded as an Interpreted Regular Tree Grammar using this algebra. This yields a straightforward, efficient chart parsing algorithm for FTAG.

1 Introduction

Like many other grammar formalisms, tree-adjoining grammars (TAG) have been extended with feature structures to model linguistic phenomena such as agreement conveniently. The FTAG formalism of Vijay-Shanker and Joshi (1988) equips each node in each elementary tree with a "top" and "bottom" feature structure. These are unified with each other at the end of the derivation if no auxiliary tree is adjoined at this node; otherwise they are unified with feature structures from the root and foot node of such an auxiliary tree. FTAG has been used successfully in large-scale grammar engineering, such as in the XTAG grammar (XTAG Research Group, 2001).

One aspect in which FTAG has lagged behind other feature grammar formalisms, such as HPSG and LFG, is in parsing. Recent efficient parsers for TAG, such as MICA (Bangalore et al., 2009) and Alto (Koller and Kuhlmann, 2012; Groschwitz et al., 2016), do not support feature structures. The recent TuLiPA parser (Kallmeyer et al., 2010) does support feature structures in FTAG parsing, but can be inefficient in practice because it enumerates all TAG derivation trees and then checks each of them for feature structure violations individually, instead of checking features on the parse chart directly. On the theoretical side, Schmitz and Le Roux (2008) explain FTAG through a feature-based formalism for describing languages of derivation trees, with unclear implications for parsing efficiency. Overall, the sense is that because of the complex interaction of unification and adjunction in FTAG, implementing efficient FTAG parsers is uncomfortable and something that tends to get avoided.

In this paper, we offer a simple and modular approach to efficient parsing with FTAG. We encode an FTAG grammar into an Interpreted Regular Tree Grammar (IRTG, Koller and Kuhlmann (2011)) by extending the TAG-to-IRTG encoding of Koller and Kuhlmann (2012) with an additional interpretation into feature structures. This interpretation maps each derivation tree into a term over a novel algebra of feature structures, in a similar way as c-structures are mapped into f-structures in LFG (Kaplan and Bresnan, 1982). This term can be evaluated to a value in the algebra if and only if all unifications required by the grammar succeed. We then show how known algorithms for IRTG chart parsing – which can be efficient for TAG (Groschwitz et al., 2016) – extend naturally to FTAG parsing.

We offer a view on FTAG which brings it more in line with other feature-based grammar formalisms, in that the distinction between top and bottom feature structures is represented correctly, but requires no special treatment by the parser. This simplifies, for instance, the use of existing unification algorithms. At the same time, we offer a very general and modular approach to checking feature unification on a parse chart; no unpacking of the individual derivation trees is required in our algorithm. This approach generalizes straightfor-

Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+13), pages 1–10,
Umeå, Sweden, September 4–6, 2017.

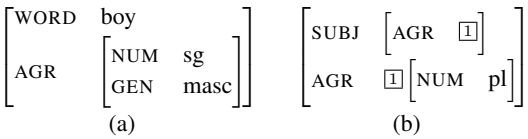

Figure 1: Two example feature structures.

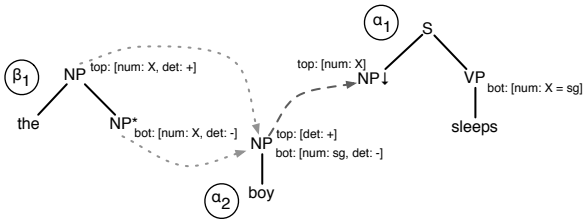

Figure 2: An example FTAG derivation.

wardly to other mechanisms for filtering derivation trees, as long as they can be expressed in terms of finite-state constraints on trees.

Plan of the paper. We will review FTAG in Section 2 and IRTGs and the TAG-to-IRTG encoding in Section 3. We will then define the feature structure algebra and show how it can be used to encode FTAG in Section 4. We show how to do efficient and modular chart parsing for FTAG in Section 5. Section 6 concludes.

2 TAG and feature structures

2.1 Feature structures

Feature structures (Shieber, 1986; Kasper and Rounds, 1986; Carpenter, 1992) are used in many grammar formalisms to represent grammatical information. Intuitively, a feature structure (FS) assigns values to a finite set of features; these values may either be atomic, or they may be feature structures themselves. For instance, the FS in Fig. 1a specifies that the WORD feature has the atomic value 'boy', and the value of the AGR feature is a feature structure with the features NUM and GEN.

Technically, feature structures represent directed, acyclic graphs in which both the top-level FS and all of the FSs it contains are nodes. If F has a feature FT, and the value of this feature in F is G, then there is an edge (with label FT) from the node F to the node G. We define a *feature path* to be a sequence $\text{F}_1.\text{F}_2.\ \ldots\ .\text{F}_n$ of features F_i. Then if F is a feature structure, we write $F.f_1.\ldots.f_n$ for feature *selection*, i.e. for the node in F that is reached by the given feature path. We define the *depth* of an FS as the length of the longest defined feature path.

As a special case, it can happen that the same node is reachable via two different feature paths, i.e. we have $F.p_1 = F.p_2$ for feature paths $p_1 \neq p_2$. This is called a *reentrancy* in F. We represent reentrancies in the attribute-value matrix notation of Fig. 1 by marking the endpoints of the reentrant feature paths with the same number. In Fig. 1b, the marker 1 indicates that $F.\text{SUBJ.AGR}$ and $F.\text{AGR}$ are the same node. Thus, we have $F.\text{SUBJ.AGR.NUM} = \text{pl}$.

We say that the FS F *subsumes* the FS G if all the information that F specifies, including reentrancies, is also present in G. The *unification* $F \sqcup G$ of F and G is the least informative FS that is subsumed both by F and G. Not all FSs can be unified. For instance, the two example FSs in Fig. 1 cannot be unified because they specify contradictory values for the feature path AGR.NUM, so no feature structure can be subsumed by both.

2.2 Feature TAG

Vijay-Shanker and Joshi (1988) extend TAG with feature structures, obtaining Feature-Based TAG (FTAG). The core point about FTAG is that every node in every elementary tree is decorated with two feature structures (FSs) – one for the "top" half of the node and one for the "bottom" half. If no auxiliary tree is adjoined at a given node u, these two feature structures must be unified with each other. However, if an auxiliary tree β is adjoined, then the top FS of u will be unified with the top FS of the root of β, and the bottom FS of u will be unified with the bottom FS of the foot node. Finally, substitution unifies the top FSs of the substitution node and the root of the initial tree. This makes unification consistent with adjunction, and allows an elegant encoding of obligatory and selective adjunction constraints in terms of FSs.

FTAG assumes that the value of each feature in each top or bottom FS is an atomic value, ensuring that all FSs have depth one. This makes the set of possible FSs finite, and thus the expressive capacity of FTAG is the same as that of TAG without features.

An example derivation of FTAG is shown in Fig. 2. This derivation combines three elementary trees α_1, α_2, β_1 by substituting α_2 into the substitution node of α_1 and then adjoining β_1 into the root of α_2. Every node in these elementary trees is annotated with a top and bottom feature

structure; feature structures whose value is $\top$, the trivial feature structure which subsumes all other feature structures, are not shown in the figure. The top and bottom FSs of the root of α_2 cannot be unified with each other because they require different values for the "det" feature; thus a derivation tree in which nothing is adjoined into this node would be ungrammatical. By adjoining β_1 into this node, we instead unify the top FS of α_2's root with the top FS of β_1's root, and the bottom FS of α_2's root with the bottom FS of β_1's foot node. Both unifications succeed.

FTAG requires that multiple occurrences of the same variable (such as X) within the FSs of the same elementary tree must be instantiated with the same value. Thus when the occurrence of X in the bottom FS of β_1's foot node is bound to 'sg', the occurrence of X in the top FS of β_1's root gets the value 'sg' too. Further unifications percolate this value into the occurrence of X in α_1, which is consistent with the requirement defined at α_1's V node. By contrast, an elementary tree for "boys" would assign 'pl' to the "num" feature at its root node, making this unification fail and establishing ungrammaticality of "boys sleeps".

The decision to assign feature structures to nodes (and not elementary trees), and to assign *two* of them to each node, is necessary to make unification consistent with adjunction. However, it makes implementing parsers inconvenient, because we do not even know what FSs need to be unified with each other before we have decided whether and what to adjoin at each node. One way to read this paper is as an illustration that these decisions are not so inextricably linked with each other as it may seem at first glance. Instead, unifications can simply be performed bottom-up at the level of the derivation tree, enabling efficient chart parsing.

We identify the nodes in an elementary tree α through their *node addresses*, i.e. words $\pi \in \mathbb{N}^*$ such that ϵ is the address of the root, and πk is the address of the k-th child of π. Given an elementary tree α in an FTAG grammar, we write $\mathsf{top}_\alpha(\pi)$ for the top FS at the node π and $\mathsf{bot}_\alpha(\pi)$ for the bottom FS. We combine the two into the feature structure

$$\phi_\alpha(\pi) = \begin{bmatrix} \text{TOP} & \mathsf{top}_\alpha(\pi) \\ \text{BOT} & \mathsf{bot}_\alpha(\pi) \end{bmatrix}$$

We drop the subscript α if the elementary tree is clear from the context.

3 Interpreted Regular Tree Grammars

We tackle FTAG parsing below by encoding it as an Interpreted Regular Tree Grammar (IRTG, Koller and Kuhlmann (2011)). IRTG is a grammar formalism in which a language of derivation trees is described using a *regular tree grammar* (RTG, (Comon et al., 2008)) or, equivalently, a *finite tree automaton*, and each derivation tree is then *interpreted* in one or more algebras. Different grammar formalisms can be encoded into IRTG by varying the algebras; for instance, (Koller and Kuhlmann, 2012) encoded TAG into IRTG by introducing two novel algebras for strings and trees.

An IRTG which encodes the TAG grammar underlying the derivation in Fig. 2 is shown in Fig. 3. The first column contains an RTG $\mathcal{G}$ which describes a language $L(\mathcal{G})$ of derivation trees. The RTG contains one rule for each elementary tree α in the TAG grammar, expanding a nonterminal symbol of the form X_S if α is an initial tree with root symbol X, or of the form X_A if α is an auxiliary tree (cf. Schmitz and Le Roux (2008)). Each rule also specifies the substitutions that must and the adjunctions that may take place at this elementary tree. Each of these nonterminals must be expanded by applying another RTG rule, corresponding to performing the respective substitution and adjunction. We can choose not to adjoin an auxiliary tree at a node where an adjunction could take place by expanding the nonterminal X_A with the rule nop_X. One derivation tree $t \in L(\mathcal{G})$ is shown in Fig. 4; note the obvious correspondence to the derivation tree underlying the TAG derivation in Fig. 2.

This derivation tree can now be mapped into a term over an algebra, and evaluated there into an object of this algebra. In general, an *algebra* over the signature Σ of operation symbols is a structure $\mathcal{A} = (A, \mathcal{I})$ consisting of a set A (the *domain* of $\mathcal{A}$) and an *interpretation function* $\mathcal{I}$. This function assigns to each operation symbol $f \in \Sigma$ of arity k a function $\mathcal{I}(f)$ that takes k arguments from A and returns a value in A. Thus, a *term* τ over the signature Σ can be evaluated to a value $[\![\tau]\!]$ recursively by letting $[\![f(\tau_1, \ldots, \tau_k)]\!] = \mathcal{I}(f)([\![\tau_1]\!], \ldots, [\![\tau_k]\!])$.

The second column of Fig. 3 shows how the example IRTG interprets derivation trees into strings. This assumes the TAG string algebra $\mathcal{A}_s = (A_s, \mathcal{I}_s)$ defined by Koller and Kuhlmann (2012). The values A_s of this algebra are all strings and

RTG rule	string homomorphism h_s	FS homomorphism h_F
$\mathrm{S}_S \to \alpha_1(\mathrm{NP}_S, \mathrm{S}_A, \mathrm{VP}_A)$	$\mathsf{wrap21}(x_2, \mathsf{conc11}(x_1, \mathsf{wrap21}(x_3, \text{sleeps})))$	$\mathsf{unify}(c_{\alpha_1}, \mathsf{embi}_1(x_1), \mathsf{emba}_\epsilon(x_2), \mathsf{emba}_2(x_3))$
$\mathrm{NP}_S \to \alpha_2(\mathrm{NP}_A)$	$\mathsf{wrap21}(x_1, \text{boy})$	$\mathsf{unify}(c_{\alpha_2}, \mathsf{emba}_\epsilon(x_1))$
$\mathrm{NP}_A \to \beta_1(\mathrm{NP}_A)$	$\mathsf{wrap21}(x_1, \mathsf{conc12}(\text{the}, *))$	$\mathsf{unify}(c_{\beta_1}, \mathsf{emba}_\epsilon(x_1))$
$X_A \to \mathsf{nop}_X$	$*$	c_{nop}

Figure 3: The FTAG from Fig. 2, encoded as an IRTG. There is a nop rule for each nonterminal $X \in \{\mathrm{S}, \mathrm{VP}, \mathrm{NP}\}$.

string pairs over a given alphabet; in the example, the alphabet contains the words "sleeps", "boy", and "the". The signature Δ_s contains a constant for each of these words, a constant * denoting the pair (ϵ, ϵ) of empty strings, plus a number of binary operations which combine strings and string pairs. In particular, $\mathcal{I}_s(\mathsf{conc11})$ is a function which takes two strings as arguments and concatenates them; $\mathcal{I}_s(\mathsf{conc12})$ takes a string v and a string pair (w_1, w_2) as arguments and concatenates them into the string pair (vw_1, w_2); and $\mathcal{I}_s(\mathsf{wrap21})$ takes a string pair (v_1, v_2) and a string w as arguments, and then "wraps" the string pair around the string, yielding the string $v_1 w v_2$. Now the IRTG $\mathcal{G}$ has an *interpretation* $(h_s, \mathcal{A}_s)$. A derivation tree t is recursively mapped to a term $h_s(t)$ by the tree homomorphism h_s, and then $h_s(t)$ is evaluated to a value $[\![h_s(t)]\!]$ in $\mathcal{A}_s$. In this way, t describes a string. For instance, for the derivation tree t in Fig. 4, we obtain $[\![h_s(t)]\!] =$ "the boy sleeps".

In general, an IRTG $\mathbb{G} = (\mathcal{G}, (h_1, \mathcal{A}_1), \ldots, (h_n, \mathcal{A}_n))$ consists of an RTG $\mathcal{G}$ and $n \geq 1$ interpretations. It describes the language $L(\mathbb{G}) = \{([\![h_1(t)]\!], \ldots, [\![h_n(t)]\!]) \mid t \in L(\mathcal{G})\}$. Thus, if we consider only the first and second column in the example grammar in Fig. 3, we have a grammar which describes a language of strings, and captures precisely the TAG part of the grammar underlying Fig. 2. The third column is for the feature structure interpretation we define below, and extends the IRTG into an encoding of an FTAG grammar.

4 A feature structure algebra for FTAG

As the grammar in Fig. 3 illustrates, the basic intuition of the TAG-to-IRTG encoding is to traverse the derivation tree bottom-up, calculating an interpretation for each node in the derivation tree. Each rule of the IRTG specifies the function by which the interpretation for the parent is calculcated from those of the children; so for instance, the second column of Fig. 3 expresses for each elementary tree of the TAG grammar how it combines the strings of its children with each other.

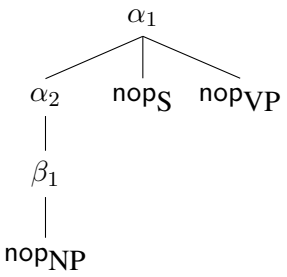

Figure 4: Derivation tree of the IRTG in Fig. 3, representing the (F)TAG derivation of Fig. 2.

We will follow the same intuition here and specify how the feature structure for a subtree of the derivation tree is computed out of the FSs of its parts. We will first introduce an algebra $\mathbb{F}$ for feature structures, with novel operations that are suitable for FTAG, and then show how an FTAG can be converted into an IRTG over this algebra.

4.1 A feature structure algebra

We define an algebra $\mathbb{F} = (\mathcal{F}, \mathcal{I})$ consisting of a set $\mathcal{F}$ of feature structures and an interpretation function $\mathcal{I}$. We let $\mathcal{F}$ contain all feature structures whose feature paths have the form $\pi\ tb\ f$, where $\pi \in \mathbb{N}^* \cup \{\mathrm{RT}, \mathrm{FT}\}$ is either a node address in an elementary tree or a special feature for the root or foot node, respectively; $tb \in \{\mathrm{TOP}, \mathrm{BOT}\}$; and f is a feature from an FTAG grammar, as in Section 2. The value reached under each feature path is either an atomic value, such as 'sg', or the placeholder $\top$, which unifies with any atomic value. We assume that every feature structure in $\mathcal{F}$ contains a RT feature; it may or may not contain a FT feature as well.

We define $\mathcal{I}$ to provide interpretations for the following four classes of function symbols.

- Any *constant* c is interpreted as a feature structure $\mathcal{I}(c) \in \mathcal{F}$. The construction in Section 4.2 will yield a finite set of feature structures $F \in \mathcal{F}$, one for each elementary

tree. We assume that $\mathbb{F}$ has a constant c_F with $\mathcal{I}(c_F) = F$ for each of these.

- For any path $\pi \in \mathbb{N}^*$, we have a function symbol embi_π for *initial-tree embedding*. Given a feature structure $F \in \mathcal{F}$, we let $\mathcal{I}(\mathsf{embi}_\pi) = \mathrm{EI}_\pi$ with $\mathrm{EI}_\pi(F) = [\pi \;\; F.\textsc{rt}]$.

- For any path $\pi \in \mathbb{N}^*$, we have a function symbol emba_π for *auxiliary-tree embedding*. Given a feature structure $F \in \mathcal{F}$, we let $\mathcal{I}(\mathsf{emba}_\pi) = \mathrm{EA}_\pi$ with

$$\mathrm{EA}_\pi(F) = \begin{bmatrix} \pi & \begin{bmatrix} \textsc{top} & F.\textsc{rt}.\textsc{top} \\ \textsc{bot} & F.\textsc{ft}.\textsc{bot} \end{bmatrix} \end{bmatrix}$$

- The binary function symbol unify represents unification of two feature structures. For any two arguments $F, G \in \mathcal{F}$, we let $\mathcal{I}(\mathsf{unify})(F, G) = F \sqcup G$ if F and G can be unified; otherwise $\mathcal{I}(\mathsf{unify})(F, G)$ is undefined.

Note that $\mathbb{F}$ is a partial algebra, in that not every term over the algebra has a value. This happens in particular if the unify operation attempts to unify two feature structures which cannot be unified. We say that $[\![t]\!]$ is undefined in such a case.

4.2 Encoding FTAG with the feature structure algebra

We will now use this algebra to encode arbitrary FTAG grammars into IRTGs. We extend the TAG-to-IRTG encoding of Koller and Kuhlmann (2012) with an additional interpretation $(h_F, \mathbb{F})$ into the feature structure algebra (see the third column of Fig. 3). This means that we need to define, for each elementary tree α in the FTAG grammar, an image $h_F(\alpha)$ for the homomorphism into the FS algebra, which specifies how α combines the FSs of its children.

Let us say that the elementary tree α was encoded into an IRTG rule $N \to \alpha(N_S^{(1)}, \ldots, N_S^{(k)}, N_A^{(k+1)}, \ldots N_A^{(n)})$ by the conversion in Section 3, with k child nonterminals for substitution and $n - k$ for adjunction. The homomorphic image $h_F(\alpha)$ will select the RT and FT entries of the FSs for the child nonterminals and unify them with one large FS $T(\alpha)$ representing the functional behavior of α; we will define $T(\alpha)$ below. Let c_α be the constant with $\mathcal{I}(c_\alpha) = T(\alpha)$ (assumed to exist above). Then we let

$$\begin{aligned} h_F(\alpha) = \;\; & \mathsf{unify}(c_\alpha, \\ & \quad \mathsf{embi}_{\pi_1}(X_1), \ldots, \mathsf{embi}_{\pi_k}(X_k), \\ & \quad \mathsf{emba}_{\pi_{k+1}}(X_{k+1}), \ldots, \mathsf{emba}_{\pi_n}(X_n)), \end{aligned}$$

where we abbreviate the $n - 1$ binary unify operations that are needed to combine the arguments by simply writing unify once.

We construct $T(\alpha) = F(\alpha) \sqcup I(\alpha) \sqcup N(\alpha) \sqcup R(\alpha)$ by unifying four smaller feature structures, each of which encodes one specific aspect of α:

1. A feature structure $F(\alpha)$ which captures the top and bottom feature structures at each node of α. For any node π of α, $F(\alpha)$ contains an entry $[\pi \; \phi_\alpha(\pi)]$.

2. A feature structure $I(\alpha)$ which enforces the coindexations for features in α that share variables. If a variable X occurs both at feature f_1 in the tb_1 feature structure of the node π_1 (where $tb_1 \in \{\textsc{top}, \textsc{bot}\}$) and at feature f_2 in the tb_2 FS of the node π_2, we let

$$\begin{bmatrix} \pi_1 & \begin{bmatrix} tb_1 & \begin{bmatrix} f_1 & \boxed{1} \end{bmatrix} \end{bmatrix} \\ \pi_2 & \begin{bmatrix} tb_2 & \begin{bmatrix} f_2 & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

   subsume $I(\alpha)$.

3. A feature structure $N(\alpha)$ which unifies the top and bottom feature structures at nodes where nothing can be adjoined. This includes all foot nodes, as well as all nodes explicitly marked with a null adjunction constraint. The IRTG rule contains no child nonterminal for adjoining at this node; thus without $N(\alpha)$, the top and bottom FSs at these nodes would not get unified at all. For all such nodes π, we let

$$\begin{bmatrix} \pi & \begin{bmatrix} \textsc{top} & \boxed{1} \\ \textsc{bot} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

   subsume $N(\alpha)$. Note that substitution nodes do *not* generate entries in $N(\alpha)$, as this would unify the top and bottom FS of the root of an initial tree substituted there.

4. A feature structure $R(\alpha)$ which makes the correct sub-feature-structures accessible under the RT and FT features. If α is an initial

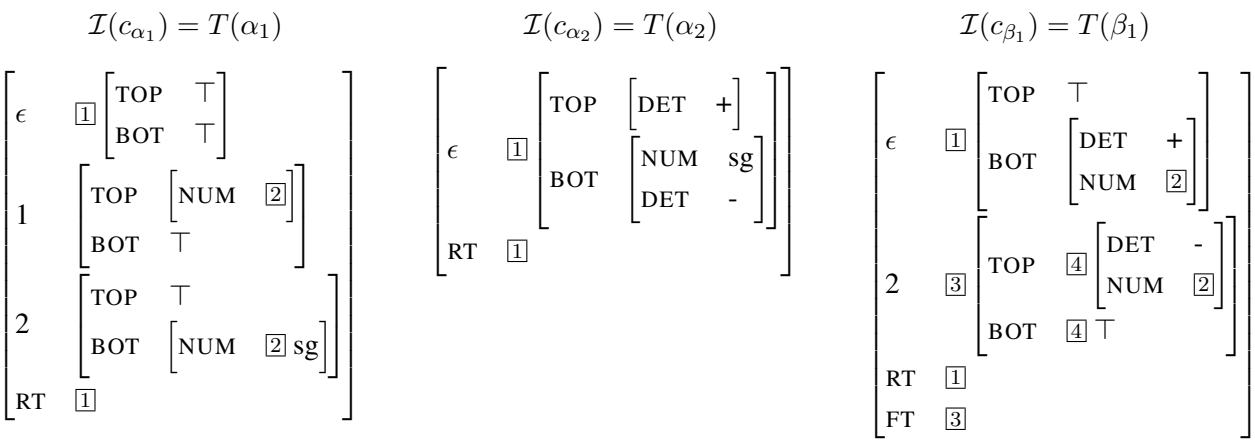

Figure 5: Feature structure encodings for the elementary trees in Fig. 2.

tree, we let

$$R(\alpha) = \begin{bmatrix} \text{RT} & \boxed{1} \\ \epsilon & \boxed{1} \end{bmatrix}$$

If α is an auxiliary tree with its foot node at address π, we let

$$R(\alpha) = \begin{bmatrix} \text{RT} & \boxed{1} \\ \text{FT} & \boxed{2} \\ \epsilon & \boxed{1} \\ \pi & \boxed{2} \end{bmatrix}$$

In addition to rules that encode elementary trees, the IRTG also contains rules $X_A \to \mathsf{nop}_X$ for every nonterminal symbol X. For these symbols, we let $h_F(\mathsf{nop}_N) = c_{\mathsf{nop}}$, with a constant c_{nop} which evaluates to the feature structure

$$\mathcal{I}(c_{\mathsf{nop}}) = F_{\mathsf{nop}} = \begin{bmatrix} \text{RT} & \begin{bmatrix} \text{TOP} & \boxed{1} \end{bmatrix} \\ \text{FT} & \begin{bmatrix} \text{BOT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Thus, whenever we choose not to adjoin an auxiliary tree at a certain node, the IRTG encodes this with a nop operation. When the FS for this nop operation is unified into the parent FS using an emba_π operation, this unifies the top and bottom feature structures of π, as required in FTAG.

4.3 An example

We illustrate this construction with the example FTAG grammar from Fig. 2. The homomorphism h_F is shown in the third column of Fig. 3; it uses constants $c_{\alpha_1}, c_{\alpha_2}, c_{\beta_1}$, whose values are the FSs shown in Fig. 5.

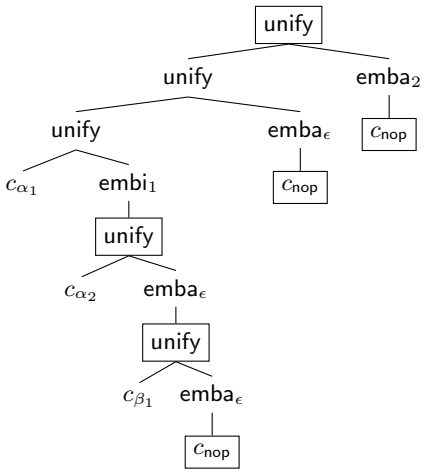

Figure 6: The term $h_F(t)$ over the feature structure algebra for the derivation tree t from Fig. 4.

Note first that all three FSs have one feature for each (non-lexical) node in the respective elementary tree, with a TOP and BOT feature nested below, representing the upper and lower feature structure of that node. This part of the feature structure is contributed by $F(\alpha)$. Any two occurrences of the same variable are coindexed through $I(\alpha)$; see e.g. the index $\boxed{2}$ in $T(\alpha_1)$ and the index $\boxed{2}$ in $T(\beta_1)$.

Furthermore, all nodes where no adjunction can take place have had their top and bottom feature structure coindexed by $N(\alpha)$. The index $\boxed{4}$ for the foot node at position 2 in β_1 was introduced like this. Finally, all three FSs have a RT feature, contributed by $R(\alpha)$ and coindexed with the ϵ feature for the root node of the elementary tree. Because β_1 is an auxiliary tree, $T(\beta_1)$ also has a FT feature, coindexed with the foot node at position 2.

Now consider the IRTG derivation tree t in Fig. 4, which encodes the TAG derivation in Fig. 2.

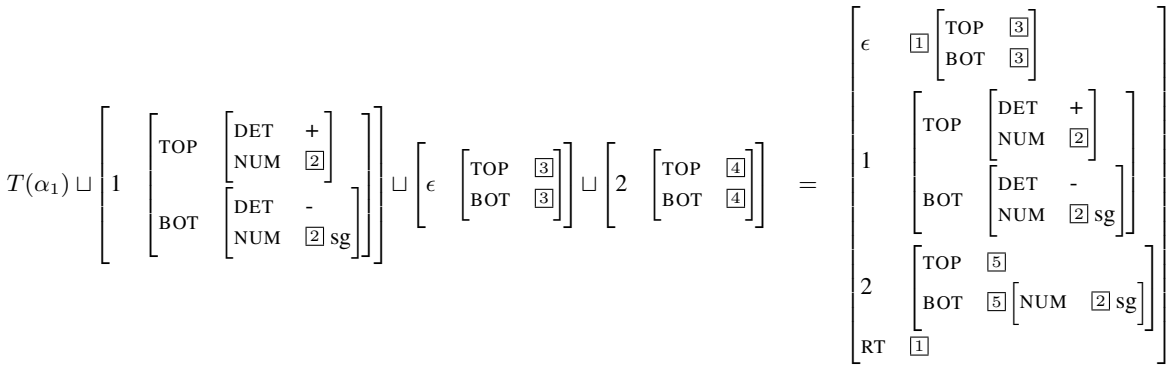

Figure 7: Computing the value $[\![h_F(t)]\!]$ of the term in Fig. 6.

The homomorphism h_F maps t to a term $h_F(t)$ over the FS algebra $\mathbb{F}$, shown in Fig. 6. We have marked with a box the root of each subtree $h_F(t')$ where t' is a subtree of t. Observe that each "block" between two boxed nodes has a consistent shape, in that it unifies c_α with a number of FSs, each of which is obtained by embedding the FS for a subtree into the features of $T(\alpha)$.

The term $h_F(t)$ can then be evaluated to a feature structure $[\![h_F(t)]\!]$ in the algebra $\mathbb{F}$. The last step of computing $[\![h_F(t)]\!]$ for the entire term $h_F(t)$ in Fig. 6 is shown in Fig. 7. We perform three unification operations, with the first argument being $\mathcal{I}(c_{\alpha_1}) = T(\alpha_1)$ and the second being the value of the sub-derivation-tree $\alpha_2(\beta_1(\mathsf{nop}_{\mathrm{NP}}))$, embedded at the substitution node 1. The other two arguments come from the nop nodes, embedded at the appropriate features through emba_π operations. Notice that the top and bottom feature structures at 1 are not coindexed, because we adjoined β_1 into the root of α_2, and the top and bottom features are not coindexed in $T(\beta_1)$.

5 Parsing

We now turn to the issue of FTAG parsing. By encoding an FTAG grammar as an IRTG with an interpretation into the FS algebra, we can apply standard methods from IRTG parsing to FTAG parsing. We will first compute a parse chart for the input string while ignoring the feature structures, and then intersect it with an RTG that carries out the unifications.

The specific parsing problem we solve is as follows: Given an IRTG $\mathbb{G} = (\mathcal{G}, (h_s, \mathcal{A}_s), (h_F, \mathbb{F}))$ and an input string w, find a compact representation C_w of the set $\mathsf{parses}(w) = \{t \in L(\mathcal{G}) \mid [\![h_s(t)]\!] = w$ and $[\![h_F(t)]\!]$ is defined$\}$ – that is, the derivation t must be grammatical according to $\mathcal{G}$; it must map to the input string under the string interpretation; and all unifications required by the feature structure interpretation succeed. We will represent this *parse chart* C_w as an RTG such that $L(C_w) = \mathsf{parses}(w)$.

5.1 Parsing without feature structures

In a first step, we apply standard methods from IRTG parsing (Koller and Kuhlmann, 2011; Groschwitz et al., 2016) to obtain a parse chart of all grammatical derivation trees that interpret to w. We do this by computing a *decomposition grammar* D_w for w in the TAG string algebra – i.e., an RTG such that $L(D_w)$ is the set of all terms over the TAG string algebra that evaluate to w. We then look at the set $L_I = h_s^{-1}(L(D_w))$ of derivation trees t such that $h_s(t) \in L(D_w)$, i.e. of derivation trees that interpret to w. Because regular tree languages are closed under inverse tree homomorphisms (Comon et al., 2008), we can calculate an RTG I_w such that $L(I_w) = L_I$. Because regular tree languages are also closed under intersection, we can then intersect I_w with $\mathcal{G}$ to obtain an RTG C_w^0 such that $L(C_w^0) = \{t \in L(\mathcal{G}) \mid [\![h_s(t)]\!] = w\}$. We call C_w^0 the *pre-chart* for w.

Consider, by way of example, the pre-chart C_w^0 for the input string $w =$ "the boy sleeps" and our example grammar; see Koller and Kuhlmann (2012) for details. The nonterminals of C_w^0 are pairs of nonterminals of the derivation tree RTG $\mathcal{G}$ and nonterminals of I_w. Nonterminals of I_w of the form $i{-}k$ can derive terms which evaluate to the substring of w from position i to $k-1$; we write pairs of nonterminals N_s with such nonter-

minals as $[N_s, i{-}k]$. Nonterminals of I_w can also be of the form $(i{-}j, k{-}l)$, for terms which evaluate to a pair of substrings of w (from position i to $j-1$ and from k to $l-1$ respectively). We write pairs of nonterminals N_A with such nonterminals as $\langle N_a, i{-}j, k{-}l\rangle$. Note that a span $i{-}i$ represents an empty string at position i. Some of the rules in C_w^0 are as follows:

$$\begin{array}{rcl}
\langle \mathrm{NP}_A, 1{-}1, 3{-}3\rangle & \to & \mathsf{nop}_{\mathrm{NP}} \\
\langle \mathrm{NP}_A, 2{-}2, 3{-}3\rangle & \to & \mathsf{nop}_{\mathrm{NP}} \\
\langle \mathrm{NP}_A, 1{-}2, 3{-}3\rangle & \to & \beta_1(\langle \mathrm{NP}_A, 1{-}1, 3{-}3\rangle) \\
[\mathrm{NP}_S, 2{-}3] & \to & \alpha_2(\langle \mathrm{NP}_A, 2{-}2, 3{-}3\rangle) \\
[\mathrm{NP}_S, 1{-}3] & \to & \alpha_2(\langle \mathrm{NP}_A, 1{-}2, 3{-}3\rangle)
\end{array}$$

Using these rules, we can derive both the sub-derivation-tree $t_1 = \alpha_2(\mathsf{nop}_{\mathrm{NP}})$, which evaluates to the string "boy" (as an NP) on the string interpretation of the IRTG, and the sub-derivation-tree $t_2 = \alpha_2(\beta_1(\mathsf{nop}_{\mathrm{NP}}))$, which evaluates to "the boy". Notice that while both of these subtrees are allowed by the underlying TAG grammar, only $h_F(t_2)$ can be evaluated over the FS algebra. To evaluate $h_F(t_1)$, we would have to unify the top and bottom FS at the root of α_2, which fails.

5.2 Feature structure filtering

In order to exclude t_1 from the language of the chart, while retaining t_2, we can define an RTG FG, which tracks feature structures and filters out trees with unification failures. The nonterminals of FG are symbols $[F]$, where F is any feature structure in $\mathcal{F}$; this is a finite set because we have limited the depth of these feature structures to three. We can then define rules for FG which simply interpret the function symbols of $\mathbb{F}$:

$$\begin{array}{rcll}
[\mathcal{I}(c)] & \to & c & \text{for constants } c \\
[\mathrm{EI}_\pi(F)] & \to & \mathsf{embi}_\pi([F]) & \\
[\mathrm{EA}_\pi(F)] & \to & \mathsf{emba}_\pi([F]) & \\
[F \sqcup G] & \to & \mathsf{unify}([F],[G]) & \text{if defined}
\end{array}$$

We add a start symbol S_{FG} and rules $S_{\mathrm{FG}} \to [F]$ for all feature structures F. Then the language $L(\mathrm{FG})$ consists exactly of all terms τ over $\mathbb{F}$ such that $[\![\tau]\!]$ is defined. As a consequence, we can intersect C_w^0 with $h_F^{-1}(\mathrm{FG})$ to obtain the chart C_w, which describes only derivation trees that can be interpreted in the FS algebra, i.e. where all unifications succeed.

In our example, the intersection C_w of C_w^0 with FG contains (among others) the following rules:

$$\begin{array}{rcl}
\langle \mathrm{NP}_A, 1{-}1, 3{-}3, F_{\mathsf{nop}}\rangle & \to & \mathsf{nop}_{\mathrm{N}} \\
\langle \mathrm{NP}_A, 2{-}2, 3{-}3, F_{\mathsf{nop}}\rangle & \to & \mathsf{nop}_{\mathrm{NP}} \\
\langle \mathrm{NP}_A, 1{-}2, 3{-}3, F_1\rangle & \to & \beta_1(\langle \mathrm{NP}_A, 1{-}1, 3{-}3, F_{\mathsf{nop}}\rangle) \\
[\mathrm{NP}_S, 1{-}3, F_2] & \to & \alpha_2(\langle \mathrm{NP}_A, 1{-}2, 3{-}3, F_1\rangle) \\
[\mathrm{NP}_S, 1{-}3, S_{\mathrm{FG}}] & \to & [\mathrm{NP}_S, 1{-}3, F_2],
\end{array}$$

where $F_1 = T(\beta_1) \sqcup \mathrm{EA}_\epsilon(F_{\mathsf{nop}})$ and $F_2 = T(\alpha_2) \sqcup \mathrm{EA}_\epsilon(F_1)$; we have already seen F_2 as the second argument of the unification in Fig. 7. One can think of these rules as copies of the rules from C_w^0, each decorated with a feature structure. Observe that the rule for $[\mathrm{NP}_S, 1{-}3]$, corresponding to "the boy", is simply extended with the FS F_2. The chart has further rules (not shown above) which derive the derivation tree t from Fig. 4, which contains t_2 as a subtree, we we find that $t \in L(C_w) = \mathsf{parses}(w)$.

By contrast, C_w does not contain decorated rules for $[\mathrm{NP}_S, 2{-}3]$. This is because $[\![h_F(t_1)]\!]$ is undefined in $\mathbb{F}$, and therefore t_1 is not in $L(\mathrm{FG})$, and the pre-chart rule for $[\mathrm{NP}_S, 2{-}3]$ is filtered out by the intersection algorithm.

5.3 Discussion

Our parsing algorithm computes the full parse chart in two steps: We first parse the input into the pre-chart and then refine the pre-chart into a chart by intersecting it with an RTG which filters out derivation trees that do not unify. The filter grammar FG and the intersection algorithm can be implemented in a way that does not require us to compute all rules of FG beforehand (which would be extremely expensive). Instead, the intersection algorithm can query the rules of FG as it goes along, which essentially amounts to evaluating the operations of $\mathbb{F}$ on the feature structure decorations.

In addition, this algorithm does not require enumerating all derivation trees before the unifications are checked; all unifications are performed on chart items. This is in line with parsers for grammar formalisms such as HPSG and especially LFG, and in contrast to other parsers for FTAG. At the same time, our algorithm avoids the exponential blowup of lexical ambiguity which would result from compiling the FTAG grammar into an ordinary TAG grammar. The filtering method presented here could be used more generally to enforce any well-formedness condition on derivation trees which can be expressed by an RTG, such as well-typedness given a type system.

5.4 Implementation

We have implemented the FS algebra and the filter grammar described above in the context of the Alto IRTG parser (Gontrum et al., 2017). Alto is available open-source from `https://bitbucket.org/tclup/alto`.

Our implementation uses well-known efficient algorithms for unification (Tomabechi, 1991) and subsumption checking (Malouf et al., 2000). Subsumption checking is needed because as the intersection algorithm discovers new candidate rules for C_w, it has to check whether another rule with an equal feature structure already exists; this equality checking is performed by testing whether each FS subsumes the other one. Parsing efficiency could be improved further by replacing this with asymmetric subsumption checks and retaining only the chart rule with the less informative FS (Zhang et al., 2007).

6 Conclusion

We have defined an algebra of feature structures and used it to encode FTAG grammars into IRTG grammars with two interpretations – one over a TAG string algebra and one over the feature structure algebra. We have shown how to do FTAG parsing by intersecting a parse chart with respect to the input string with an RTG which attempts to carry out the unifications required by the FTAG grammar. This yields a clean, LFG-style separation of the derivation process of TAG (including adjunction) from unification, and an efficient parsing algorithm that performs unifications on the chart and not the individual derivation trees.

One advantage of our FS algebra is that it can be combined freely with any other grammar formalism that can be encoded in IRTG. It should, for instance, be straightforward to define feature-based LCFRS by adding variants of emba_π for rules of higher fan-out. By adding further interpretations, we can also build feature-based *synchronous* grammars, e.g. for semantic parsing (Peng et al., 2015; Groschwitz et al., 2015). An interesting challenge would be to attempt to encode LFG into IRTG, using a string interpretation for the c-structure and a variant of the FS algebra introduced here for the f-structure. This would require dealing with arbitrarily deep feature structures (as opposed to ones of bounded depth as in FTAG), and encoding LFG's completeness and coherence constraints into a filter grammar.

Acknowledgments. I am grateful to the reviewers for their insightful and constructive comments, and to Jessica Grasso, Jonas Groschwitz, Christoph Teichmann, and Stefan Thater for discussions. I am also indebted to the students in my Grammar Formalisms classes at the University of Potsdam for demanding a faster FTAG parser, and those at Saarland University for being the first users of the system described here.

References

Srinivas Bangalore, Pierre Boullier, Alexis Nasr, Owen Rambow, and Benoit Sagot. 2009. MICA: A probabilistic dependency parser based on Tree Insertion Grammars. In *Proceedings of NAACL HLT 2009: Short Papers*.

Bob Carpenter. 1992. *The logic of typed feature structures*. Cambridge University Press.

Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Lding, Sophie Tison, and Marc Tommasi. 2008. Tree automata techniques and applications. Available on `http://tata.gforge.inria.fr/`.

Johannes Gontrum, Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2017. Alto: Rapid prototyping for parsing and translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL): Demo Session*.

Jonas Groschwitz, Alexander Koller, and Mark Johnson. 2016. Efficient techniques for parsing with tree automata. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2015. Graph parsing with s-graph grammars. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL-IJCNLP)*.

Laura Kallmeyer, Wolfgang Maier, Yannick Parmentier, and Johannes Dellert. 2010. TuLiPA – Parsing extensions of TAG with Range Concatenation Grammars. *Bulletin of the Polish Academy of Sciences – Technical Sciences* 58(3):377–391.

Ronald Kaplan and Joan Bresnan. 1982. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, MA, pages 173–381.

Robert Kasper and William Rounds. 1986. A logical semantics for feature structures. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Alexander Koller and Marco Kuhlmann. 2011. A generalized view on parsing and translation. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT)*.

Alexander Koller and Marco Kuhlmann. 2012. Decomposing TAG algorithms using simple algebraizations. In *Proceedings of the 11th TAG+ Workshop*.

Robert Malouf, John Carroll, and Ann Copestake. 2000. Efficient feature structure operations without compilation. *Natural Language Engineering* 6(1):29–46.

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 (CoNLL)*.

Sylvain Schmitz and Joseph Le Roux. 2008. Feature unification in TAG derivation trees. In *Proceedings of the 9th TAG+ Workshop*.

Stuart Shieber. 1986. *An introduction to unification-based approaches to grammar*. CSLI Publications.

Hideto Tomabechi. 1991. Quasi-destructive graph unification. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL)*.

K. Vijay-Shanker and Aravind Joshi. 1988. Feature structures based tree-adjoining grammar. In *Proceedings of COLING*.

XTAG Research Group. 2001. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania. `ftp://ftp.cis.upenn.edu/pub/xtag/release-2.24.2001/tech-report.pdf`.

Yi Zhang, Stephan Oepen, and John Carroll. 2007. Efficiency in unification-based n-best parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT)*.