Appears in the Proceedings of TAG+, 4th Workshop, 1998, Philadelphia, PA.

TAGS $\mathcal{M}$-CONSTRUCTED

Uwe Mönnich
Seminar für Sprachwissenschaft
Tübingen University
Wilhelmstrasse 113
D-72074 Tübingen
Uwe.Moennich@uni-tuebingen.de

Abstract

This paper puts TAGs into an algebraic perspective. The operation of tree adjunction is shown to be a special case of function substitution within a derived theory. The underlying process of theory derivation is illustrated with the concrete example of free continuous tree algebras.

1 Introduction

The aim of this paper is to relate two notions. The first one is that of tree adjunction. The operation of tree adjunction serves to seperate dependency and recursion within a mild extension of the context-free grammar formalism. The second notion is that of a polyadic procedure. It generalizes the operation of making several identical copies of a string and was introduced in formal language theory by Fischer (1968).

The two notions are related in the following way. The operation of tree adjunction builds a new tree t from two input trees t_1 and t_2 by replacing a subtree of t_1 displaying a root label identical to t_2's root label with the tree t_2 and appending the replaced subtree of t_1 to an especially marked leaf node of t_2. The name of a polyadic procedure in a tree can similary be replaced by a tree with dummy symbols at some of its leaves into which the arguments of the replaced procedure are to be inserted.

It has long been realized that the introduction of higher order auxiliairy symbols into a grammar formalism is an iterable process that leads to an algebraic refinement of the Chomsky hierarchy. The most general characterization of this iterable process is due to the ADJ group and presented by them within the category theoretic framework of finitary algebraic theories (Bloom et al. 1983). Based on their presentation, we propose an abstract formulation of tree-adjoining grammars in which its rule systems correspond to morphisms of an algebraic theory that is constructed from the algebraic theory of context-free grammars along the lines indicated by the ADJ group.

The notion of an algebraic refinement of the Chomsky hierarchy was first formulated by Wand (1975). He shows that solving regular equations in function spaces over languages leads to a hierarchy of language families beginning with the regular languages, the contex-free languages and the indexed language. His conjecture that these language families are but the first steps in an infinite hierarchy was later confirmed by Damm (1982).

The original motivation for our interest is an algebraic formulation of tree adjoining grammars has come from a long term project on denotational semantics for grammar formalisms. Algebraic semantics seems to provide a uniform framework for such an attempt. In the present connection the algebraic perspective not only adds another characterization of the tree adjoining languages to the already long list of equivalences with restricted production systems, but it also makes available the whole gamut of techniques that have been developed in the tradition of algebraic language theory (Maibaum 1978, Mehlhorn 1979, Schimpf and Gallier 1985).

In the interest of a more concrete presentation we restrict ourselves to the special case of tree algebras. The basic notions from universal algebra which we need in the sequel are introduced in the next section. For reasons of space we have refrained from supplying the details of the general $\mathcal{M}$-functor.

2 Basic Definitions

Let S be a set of sorts. A *many-sorted signature* Σ is an indexed family $\langle \Sigma_{w,s} | w \in S^*, s \in S \rangle$ of disjoint sets. A symbol in Σ_w, s is called an *operator* of *type* $\langle w, s \rangle$, *arity* w, *sort* s and *rank* $\ell(w)$, where $\ell(w)$ denotes the length of w. In the case of a single-sorted signature we write $\Sigma_{s^n,s}$ as Σ_n. The set of n-ary *trees* over such a single-sorted signature Σ is built up from a finite set $X_n = \{x_1, \ldots, x_n\}$ of *variables* using the operators in the expected way: If $\sigma \in \Sigma_n$ and $t_1, \ldots, t_n$ are n-ary trees, then $\sigma(t_1, \ldots, t_n)$ is an n-ary tree.

The operator symbols induce operations on an algebra of the appropiate structure. A Σ-*algebra* A consists of an S-indexed family of sets $A = \langle A_s \rangle_{s \in S}$ and for each operator $\sigma \in \Sigma_{w,s}$, a function $\sigma : A^w \to A^s$ where $A^w = A_1^w \times \cdots \times A_n^w$ and $w = w_1 \cdots w_n$. The set of n-ary trees $T(\Sigma, X_n)$ can be made into a Σ-algebra by specifiying the operations as follows. For every $\sigma \in \Sigma_n$ and every $t_1, \ldots, t_n \in T(\Sigma, X_n)$ we identify $\sigma_{T(\Sigma, X_n)}(t_1, \ldots, t_n)$ with $\sigma(t_1, \ldots, t_n)$.

3 Lawvere Theories

Our main notion is that of an *algebraic (Lawvere) theory*. Given a set of sorts S, an algebraic theory, as an algebra, is an $S^* \times S^*$-sorted algebra $\mathbf{T}$, whose carriers $\langle T(u, v) | u, v \in S^* \rangle$ consist of the morphisms of the theory and whose operations are of the following types:

- projection: $x_i^u \in T(u, u_i)$ $(u = u_1 \ldots u_n \in S^*)$
- composition: $\cdot_{u,v,w} \in T(u, v) \times T(v, w) \to T(u, w)$ $(u, v, w \in S^*)$
- target tupling: $(\quad, \ldots, \quad)_{u,v} \in T(u, v_1) \times \cdots \times T(u, v_n) \to T(u, v)$ $(u, v = v_1 \ldots v_n \in S^*)$

The projections and the operations of target tupling are required to satisfy the obvious identities for products and the composition operations are required to be associative:

- $x_i^v \cdot (\alpha_1, \ldots, \alpha_n)_{u,v} = \alpha_i$ for all $\alpha_i \in T(u, v_i)$
- $(x_1^v \cdot \beta, \ldots, x_n^v \cdot \beta)_{u,v} = \beta$ for all $\beta \in T(u, v)$, where $v = v_1 \cdots v_n$
- $(\gamma \cdot \beta) \cdot \alpha = \gamma \cdot (\beta \cdot \alpha)$ for all $\alpha \in T(u, v), \beta \in T(v, w), \gamma \in T(w, z)$
- $\alpha \cdot (x_1^u, \ldots, x_1^u)_{u,u} = \gamma$ for all $\alpha \in T(u, v)$, where $u = u_1 \cdots u_n$

By rearranging the ingredients of the prededing definition algebraic theories can be looked upon as categories. Under this conceptualization an algebraic theory $\mathbf{T}$ has as objects $|\mathbf{T}|$ the set of sort-strings S^*, the elements of the carrier sets become morphisms in the category theoretic sense and the following tuples of the projection morphisms $(x_1^u, \ldots, x_1^u)_{u,u}$ function as identities. The axioms for the composition operation ensure that it behaves as is required by the basic category theoretic postulates for the operation of the same name and the axioms for target tupling ensure its status as a category theoretic product.

With S being a singleton, the powerset $\wp(T(\Sigma))$ of n-ary trees constitutes the central example of interest for formal language theory. The carriers $\langle \wp T(n, m) | n, m \in w \rangle$ consist of sets of m-tuples of n-ary trees $\{(t_1, \ldots, t_m)\}$. The operation of composition is defined as substitution for the projection constants and target tupling is just tupling.

The M-construction can be characterized as a functorial generalization of the device of signature extension. For lack of space we abstain from giving the general definition and restrict ourselves to outlining the relevant features for the case of free continuous theories. Suppose that Σ is an one-sorted signature. Elements of $S^* \times S^*$ can then be identified with elements of $\omega \times \omega$. Given a finite set of *function variables* F, we obtain the extended signature $\Sigma + F$, where $(\Sigma + F)_n = \Sigma_n \cup \{f | f \in F \ \& \ \mathrm{arity}(f) = n\}$. Based on this signature we are able to define the notion of a finite tree t of *recursion-sort* n and *recursion-arity* w, $w \in \omega^*$. This says that nodes in t dominating w_i daughters may be labeled with $f \in F$ of arity w_i and that its projection labels come from $X_n = \{x_1, ..., x_n\}$. Given Σ and F, we can now define the M-constructed *continuous, one-sorted recursion theory* $M(\wp(T(\Sigma)))$ as follows. For $v \in \omega^n$, $w \in \omega^*$, $M(\wp(T(\Sigma)))(w, v)$ is the powerset of all n-tuples of trees $t = (t_1, ..., t_n)$, where t_i is of recursion sort v_i and of recursion arity w. Tupling is again tupling, the function variables play the role of "higher-order" projections, but composition is specified as substitution for function-variables which label internal tree nodes, rather than as substitution for projection labels at the leaves of trees. For $u \in \omega^n$, $v \in \omega^p$ and $w \in \omega^*$, let T' be a set of p-tuples of trees $t' = (t'_1, ..., t'_p)$ of recursion arity w and of recursion sort v and let T be a set of n-tuples of trees $t = (t_1, ..., t_n)$ of recursion arity v and of recursion sort u , then their composition $T \cdot T' = \{t''\} = \{(t''_1, ..., t''_n)\} = \{(t_1 \cdot t', ..., t_n \cdot t')\}$ is defined recursively as follows:

- $t''_i = \{x_j^{v_i}\}$ for $t_i = x_j^{v_i}$
- $t''_i = \{\sigma(\tau_1 \cdot t', \ldots, \tau_q \cdot t')\}$ for $t_i = \sigma(\tau_1, \ldots, \tau_q)(\sigma \in \Sigma_q)$
- $t''_i = \{t'_j(\tau_1 \cdot t', \ldots, t_r \cdot t')\}$ for $t_i = f_j(\tau_1, \ldots, \tau_r)(f_j \in F_r)$

4 Context-Free and Tree Adjoining Languages

Consider the example of a single-sorted signature of monadic algebras:

$$\Sigma_0 = \{\epsilon\} \qquad \Sigma_1 = \{a | a \in V\}$$

Due to the fact that Σ is a *monadic* signature trees in $T(\Sigma, X)$ may not contain more than a single variable. Observe that this corresponds exactly to the rule format of regular (string) languages, where the righthand sides of production rules are either strings in the terminal alphabet or concatenations of such a string with a single non-terminal. The regular language V^*, e.g., is the solution of the set of equations $\{x = a(x)|a\}(a \in V)$ in the space $\wp(T(\Sigma))$. It should be pointed out that V^* and the set of all variable-free trees in the monadic signature Σ, introduced a moment ago, are, strictly speaking, not the same sets. They are nevertheless related by an obvious one-to-one correspondence.

Once the signature Σ is extended with one nullary and one monadic variable, the following example shows that we obtain the context-free language $L = \{a^n b^n\}$ as solution in the same space $\wp(T(\Sigma))$, where $\Sigma_1 = \{a, b\}$:

$$G = \langle \Sigma, F, S, E \rangle$$

$$F_0 = \{S\} \qquad F_1 = \{F\}$$

$$E = \left\{ \begin{array}{lcl} S & = & \{F(\varepsilon), \varepsilon\} \\ F(x) & = & \{a(F(b(x))), a(b(x))\} \end{array} \right\}$$

$$L(E, S) = \{\overbrace{a(a \ldots}^{n} \overbrace{(b(b \ldots}^{n} (\varepsilon) \ldots)\}$$

The pair of equations E in the preceding example is represented by a morphism

$$E = (E_0, E_1) : 0 \cdot 1 \to 0 \cdot 1$$

in the recursion theory $M(P(T(\sigma)))$ and the language $L = \{a^n b^n\}$ is the first component of the least fixpoint that solves the equational system E.

Observe again that the preceding equational system looks suspiciously similar to the usual production system for the "same" language in a concatenative signature Σ':

$$G' = \langle \Sigma', F, S, P \rangle$$

$$\Sigma_0 = \{\varepsilon, a, b\} \qquad \Sigma_2 = \{\frown\} \qquad F_0 = \{S\}$$

$$P = \{S \to \varepsilon \,|\, \frown(a, \frown(S, b))\}$$

$$L(G', S) = \{\frown(a, \frown(\ldots, \frown(\varepsilon, b) \ldots b) \ldots)\}$$

where n occurrences of a precede the same number of occurrences of b for $n \geq 0$.

The following result expresses the fact that the situation above characterizes already the whole class of context-free languages: Every context-free language can be represented as the solution of a morphism in an algebraic theory that is M-constructed on the basis of a monadic tree theory.

There is actually a mechanical procedure that allows one to convert an arbitrary context-free grammar $G = \langle V, N, S, P \rangle$ in Chomsky Normal Form into a weakly equivalent equational system $E = \langle \Sigma_V, F, E \rangle$ that has a solution in the space of monadic trees (Maibaum 1974). The procedure consists in first forming the monadic signature Σ_V corresponding to the terminal vocabulary V of G:

$$(\Sigma_V)_0 = \{\varepsilon\} \qquad (\Sigma_V)_1 = \{V\}$$

The new function variables F are similarly in a one-to-one correspondence with the nonterminals of G:

$$F_0 = \{S\} \qquad F_1 = \{A | A \in N\}$$

The equational system E is then obtained through the following replacements:

$$\begin{array}{lcll} S \to AB & \Rightarrow & S = \{A(B(\varepsilon))\} & \\ S \to a & \Rightarrow & S = \{a(\varepsilon)\} & \\ S \to \varepsilon & \Rightarrow & S = \{\varepsilon\} & \\ A \to BC & \Rightarrow & A(x) = \{B(C(x))\} & \text{for } A \neq S \\ A \to a & \Rightarrow & A = \{a(x)\} & \text{for } A \neq S \end{array}$$

$L(G, S)$ equals the least solution of E at its S-component.

Recall that context-free languages are also captured by the notion of a frontier or yield of a regular tree set. The obvious question that presents itself in this connection is which language family is reached by the addition of monadic function variables to an arbitrary signature.

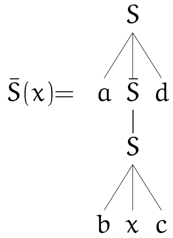
In the way of motivating the answer to this question it is useful to consider once more the example of a simple morphism $E' : 0 \cdot 1 \to 0 \cdot 1$ in an M-constructed recursion theory that is based on a signature Σ of arity 3:

$$\Sigma = \Sigma_0 \cup \Sigma_3 \text{ where } \Sigma_0 = \{a, b, c, d\} \text{ and } \Sigma_3 = \{S\}$$

$$F = F_0 \cup F_1 \text{ where } F_0 = \{S'\} \text{ and } F_1 = \{\bar{S}\}$$

$$E = \{S' = \{\bar{S}(\varepsilon)\}, \bar{S}(x) = \{S(a, \bar{S}(S(b, x, c)), d), x\}\}$$

In tree form the last equation has the following shape:



This system specifies the string language $\{a^n b^n c^n d^n\}$. Apart from minor notational modifications the grammar in the last example corresponds to a well-known tree adjoining grammar. Note that apart from the start symbol the only other nonterminal is of arity one. As was the case in connection with the context-free string languages, the preceding example is a particular instance of the general situation. The tree adjoining languages correspond to languages that are M-constructed from arbitrary signatures through the addition of monadic function variables.

As in the case of context-free grammars there exists a mechanical procedure that allows one to produce for any given tree adjoining grammar G a weakly equivalent equational system E that specifies the "same" set of trees. Strict identity is not guaranteed for grammars that contain nonterminals with variable arities. To remain within the algebraic setup, nonterminals that label nodes which branch out into different numbers of daughters, have to be assigned to different components of the indexed set Σ. Otherwise the procedure that resulted in the system of the example is completely general. Terminals and nonterminals alike are collected into the new signature Σ. All nonterminals that are free for an adjunction become duplicated by a monadic member of the set of function variables F. Adjunction constraints have to be taken over with one modification: When sa is the empty set the nonterminal has no duplicate in F.

5 Conclusion

The M-construction in its general form is conceived for Lawvere theories regarded as categories. The main prerequisites a category of such theories has to satisfy in order for it to be M-able is the existence of a free theory and of coproducts. Both conditions are fulfilled by the powerset of n-ary trees.

In compliance with the spirit of algebraic semantics I have considered tree adjoining languages as solutions of morphisms in a derived theory. Under the perspective of an operational semantics an analoguous characterization can be obtained by considering tree adjoining grammars as a restricted form of context-free tree grammars (Engelfriet and Schmidt 1977). This has been the topic of a previous publication where it is shown that not only any tree adjoining language is presentable as a monadic context-free tree language, but that the opposite implication holds as well (Mönnich 1997). The proof in that paper for this opposite direction of the implication is easily adapted to the framework of denotational semantics. As was adumbrated in the introductory section, the particular conception of denotational semantics that is being developed within the algebraic tradition promises to provide the right level of abstraction from where to investigate the connections between different types of grammatical formalisms.

References

Bloom, S.L., J.W. Thatcher, E.G. Wagner, and J.B. Wright. 1983. Recursion and iteration in continuous theories: The M-construction. *J. of Computer and System Sciences*, 27(2):148–164.

Damm, Werner. 1982. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–207.

Engelfriet, Joost and E.M. Schmidt. 1977. IO and OI, part I. *J. Comput. System Sci.*, 15:328–353.

Fischer, Michael J. 1968. Grammars with macro-like productions. In *Proceedings of the 9th Annual Symposium on Switching and Automata Theory*, pages 131–142. IEEE.

Joshi, A.K. and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages Vol. 3: Beyond Words*, volume 3. Springer, pages 69–124.

Maibaum, T.S.E. 1974. A generalized approach to formal languages. *J. Comput. System Sci.*, 8:409–439.

Maibaum, T.S.E. 1978. Pumping lemmas for term languages. *Journal of Computer and System Science*, 17:319–330.

Mehlhorn, Kurt. 1979. Parsing macro grammars top down. *Information and Control*, 40:123–143.

Mönnich, U. 1997. Adjunction as substitution. In G-J. M. Kruijff, G.V. Morrill, and R.T. Oehrle, editors, *Formal Grammar 1997: Proceedings of the Conference*. Aix-en-Provence, pages 169–178.

Schimpf, Karl and Jean Gallier. 1985. Tree pushdown automata. *Journal of Computer and System Sciences*, 30:25–40.

Wagner, E.G. 1994. Algebraic semantics. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science Vol. 3: Semantic Structures*. OUP, pages 323–393.

Wand, Michell. 1975. An algebraic formulation of the chomsky hierarchy. In *Category Theory Applied to Computation and Control*, number 25 in Lecture Notes in Computer Science, pages 209–213.