Institut für Informatik und Praktische Mathematik
der
Christian-Albrechts-Universität zu Kiel
D-24098 Kiel

Algebras for Classifying Regular Tree Languages and an Application to Frontier Testability

Thomas Wilke

Bericht Nr. 9313
Juli 1993

Algebras for Classifying Regular Tree Languages and an Application to Frontier Testability

Thomas Wilke[*]

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel, D-24098 Kiel
E-mail: `tw@informatik.uni-kiel.d400.de`

Abstract

Point-tree algebras, a class of equational three-sorted algebras are defined. The elements of sort t of the free point-tree algebra $\mathbf{T}(A)$ generated by a set A are identified with finite binary trees with labels in A. A set L of finite binary trees over A is recognized by a point-tree algebra $\mathbf{B}$ if there exists a homomorphism h from $\mathbf{T}(A)$ in $\mathbf{B}$ such that L is an inverse image of h. A tree language is regular if and only if it is recognized by a finite point-tree algebra. There exists a smallest recognizing point-tree algebra for every tree language, the so-called syntactic point-tree algebra. For regular tree languages, this point-tree algebra is computable from a (minimal) recognizing tree automaton.

The class of finite point-tree algebras recognizing frontier testable (also known as reverse definite) tree languages is described by means of equations. This gives a cubic algorithm deciding whether a given regular tree language (over a fixed alphabet) is frontier testable.

The characterization of the class of frontier testable languages in terms of equations is in contrast with other algebraic approaches to the classification of tree languages (the semigroup and the universal-algebraic approach) where such equations are not possible or not known.

Introduction

Since the sixties a great number of classes of regular languages of finite words, such as the classes of star-free, locally testable, or piecewise testable languages, have been

[*]Partly supported by ESPRIT Basic Research Action No. 6317 Algebraic and Syntactic Methods in Computer Science 2 (ASMICS 2).

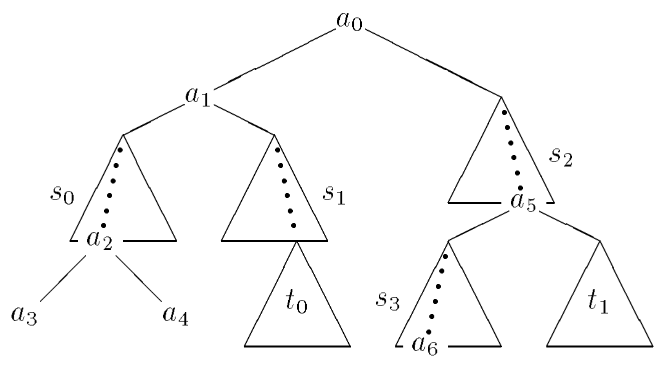


Figure 1: Sample tree skeleton corresponding to (#)

characterized in a very succinct way by giving descriptions of the corresponding classes of syntactic semigroups. For regular sets of finite trees (regular tree languages) a comparable result, using 'syntactic semigroups' for tree languages, is only known for root testability [NP89] (which corresponds to definiteness in the word case). The semigroup approach [Tho84, Heu89, NP89, PP92] to regular tree languages provides the important instrument of decomposing trees along a path (corresponding to decompositions of words), but it ignores the two-dimensional structure which trees have. So it is not surprising that the class of frontier testable tree languages cannot be classified in the semigroup framework (Example 2.17). On the other hand, the universal-algebraic approach proposed in [Ste92] provides an abstract classification theory as represented by Samuel Eilenberg's variety theorem [Eil76] for word languages, but characterizations of concrete classes (in non-trivial cases) are not known.

In this paper we introduce a new kind of algebra for the characterization and classification of regular languages of finite binary trees. We call this point-tree algebra. Basically, in a point-tree algebra trees can be decomposed in two ways: 1) along a path and 2) in the two subtrees rooted at the top node. Both methods can be combined; for instance, the two subtrees rooted at the top of a tree can be decomposed each along a path or a tree can be decomposed along a path that at some inner node splits into two paths. Even more complicated decompositions are possible. So sets of trees matching the same 'skeleton' can be described by a single term in the signature of point-tree algebras. For example, the term

$$(\#) \qquad a_0(a_1(s_0a_2(a_3,a_4),s_1t_0),s_2a_5(s_3a_6,t_1))$$

stands for the trees with shape as depicted in Fig. 1.

Sect. 1 presents the general results concerning point-tree algebras and regular tree languages. The formal connection between point-tree algebras and finite binary trees is established in Theorem 1 which states that the elements of sort t of the free point-tree algebra $\mathbf{T}(A)$ generated by a finite set A can be identified with the set of finite binary trees over A. A tree language L over A is defined to be recognized by a point-tree algebra $\mathbf{B}$ if there exists a homomorphism h from $\mathbf{T}(A)$ to $\mathbf{B}$ such that L (strictly speaking, the subset of $\mathrm{T}(A)$ associated with L) is an inverse image of h, i.e. if there is a subset P of B such that $L = h^{-1}(P)$.[1] We prove that a tree language is regular if and only if it is recognized by a finite point-tree algebra (Propositon 1.8). Moreover, it is shown that there is a smallest recognizing point-tree algebra for every tree language, the so-called syntactic point-tree algebra of the given language, which is also effectively computable (Lemma 1.12). Furthermore, it turns out that recognizing point-tree algebras (homomorphisms) and recognizing tree automata correspond to each other in a natural way (Lemma 1.11).

In Sect. 2 frontier testable tree languages are characterized. Recall that, as for reverse definite word languages, a language of finite trees is called frontier testable if membership to it depends only on the set of subtrees of bounded depth occurring at the frontier of a given tree; see [Heu89], where it was shown that the class of frontier testable languages is decidable. This was achieved by bounding the degree of frontier testability of a given language by the number of states of a recognizing automaton. We complement this by presenting for each degree k of frontier testability (given by the depth k of the considered frontier trees) a finite set of equations such that a tree language is k-frontier testable iff its syntactic point-tree algebra satisfies these equations (Theorem 3). First this allows us to reprove Heuter's results. In addition, we provide an algorithm deciding whether a given regular tree language is frontier testable that runs in time cubic in the number of states of the corresponding minimal tree automaton and quadratic in the cardinality of the underlying alphabet (Proposition 2.12). In the signature of point-tree algebras extended by the (implicit) ω-operation as known from finite semigroup theory, we characterize frontier testability in general by a finite base of equations (Theorem 4). This is a first example for a closed algebraic description of a non-trivial class of regular tree languages and gives evidence of the usefulness of the suggested approach via point-tree algebras.

By means of the close connection between syntactic point-tree algebras and minimal automata the presented equations help to understand the structure of the (minimal) automata of frontier testable languages.

The combinatorial properties of frontier testable languages used in Sect. 2 were, in a weaker form, established jointly with T. Scholz (see [Sch92]).

This paper is the full version of the conference paper [Wil93]. I am grateful to Magnus Steinby for his comments on and corrections of a preliminary version of the paper.

[1]We adopt the convention that bold symbols stand for the whole structure whereas the symbols for the underlying sets are not in bold type.

1 Point-tree algebras

Let A be an arbitrary alphabet. We consider the set $\mathcal{T}(A)$ of binary *trees* over A, that is, the set of all terms over the ranked alphabet $\Gamma(A)$ that contains for every element a of A a nullary and a binary symbol. Furthermore we are interested in the set $\mathcal{S}(A)$ of all *special trees*[2] over A which is obtained from the elements of $\mathcal{T}(A)$ by substituting $\cdot$ (*point*) for exactly one leaf, i.e. $\mathcal{S}(A)$ is the set of all terms over the ranked alphabet $\Gamma'(A)$ that contains for every element a of A a nullary and a binary symbol and the nullary symbol $\cdot$, and where point occurs exactly once. When we want to stress that we are dealing with elements of $\mathcal{T}(A)$ and not of $\mathcal{S}(A)$ we speak of *ordinary trees* (in contrast to special trees).

In the following we shall describe the relationships between the three sets A, $\mathcal{S}(A)$, and $\mathcal{T}(A)$ in terms of six functions: $\iota, \rho, \lambda, \sigma, \tau, \kappa$, which we now introduce. (For a graphical illustration see Fig. 2).

[$\iota: A \to \mathcal{T}(A)$] The function ι identifies each letter a with the one-node tree a.

[$\rho: A \times \mathcal{T}(A) \to \mathcal{S}(A)$] The function ρ takes a letter a and a tree t as arguments and maps them on the special tree with root a and right subtree t, i.e. $\rho(a,t) = a(\cdot, t)$.

[$\lambda: A \times \mathcal{T}(A) \to \mathcal{S}(A)$] This is the same function as ρ but placing the given tree to the left of the given letter, i.e. $\lambda(a,t) = a(t, \cdot)$.

[$\kappa: A \times \mathcal{T}(A) \times \mathcal{T}(A) \to \mathcal{T}(A)$] The function κ associates with a letter a and two trees t_0 and t_1 the tree with root a, left subtree t_0 and right subtree t_1, i.e. $\kappa(a, t_0, t_1) = a(t_0, t_1)$.

[$\tau: \mathcal{S}(A) \times \mathcal{T}(A) \to \mathcal{T}(A)$] Given a special tree s and a tree t, the function τ substitutes t for point in s, i.e. $\tau(s,t) = s[\cdot/t]$.

[$\sigma: \mathcal{S}(A) \times \mathcal{S}(A) \to \mathcal{S}(A)$] Given special trees s_0, s_1, the function σ substitutes s_1 for point in s_0, i.e. $\sigma(s_0, s_1) = s_0[\cdot/s_1]$.

Example 1.1 Let $A = \{0,1\}$. The tree $t = 0(1(0,1(0,1)),0(0,0))$ can be obtained in several ways starting from the letters $0, 1 \in A$. In Fig. 3 a graphical representation of t is given and some of the possible representations in terms of $\rho, \lambda, \ldots$ are listed.

Obviously the following identities hold for an arbitrary letter $a \in A$, special trees $s_0, s_1, s_2 \in \mathcal{S}(A)$, and ordinary trees $t, t_0, t_1 \in \mathcal{T}(A)$.

$$\left.\begin{array}{ll} \text{(ASS1)} & \sigma(s_0, \sigma(s_1, s_2)) = \sigma(\sigma(s_0, s_1), s_2) \\ \text{(ASS2)} & \tau(s_0, \tau(s_1, t)) = \tau(\sigma(s_0, s_1), t) \\ \text{(COM)} & \tau(\lambda(a, t_0), t_1) = \tau(\rho(a, t_1), t_0) = \kappa(a, t_0, t_1) \end{array}\right\} (*)$$

[2] Special trees were introduced in [Tho84]. They are also known as pointed trees, see [NP89]. In fact, our definition of special trees is a slightly modified version of what is called a pointed tree in [NP89].

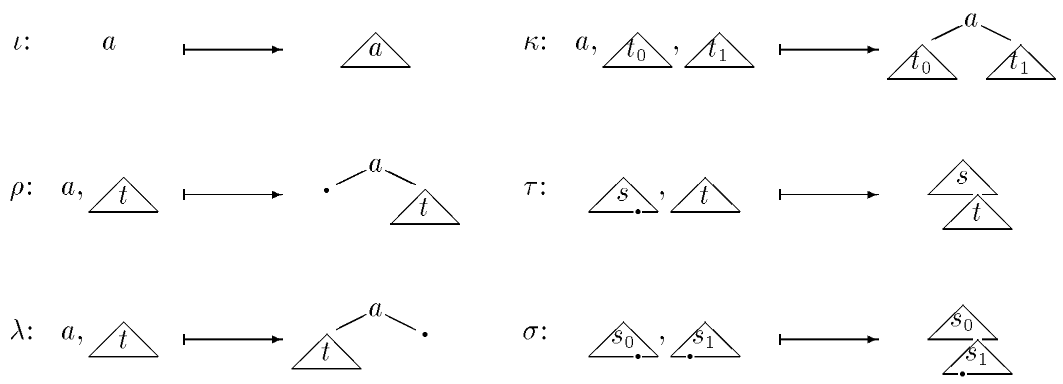
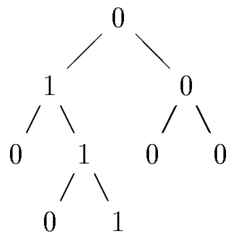


Figure 2: The six operations on A, $\mathcal{T}(A)$, and $\mathcal{S}(A)$



$$= \left\{ \begin{array}{l}
\kappa(0,\kappa(1,\iota(0),\kappa(1,\iota(0),\iota(1))),\kappa(0,\iota(0),\iota(0))) \\
\kappa(0,\kappa(1,\iota(0),\kappa(1,\iota(0),\iota(1))),\tau(\rho(0,\iota(0)),\iota(0))) \\
\tau(\sigma(\sigma(\rho(0,\kappa(0,\iota(0),\iota(0))),\lambda(1,\iota(0))),\rho(1,\iota(1))),\iota(0)) \\
\tau(\sigma(\rho(0,\kappa(0,\iota(0),\iota(0))),\sigma(\lambda(1,\iota(0)),\rho(1,\iota(1)))),\iota(0)) \\
\tau(\tau(\rho(0,\kappa(0,\iota(0),\iota(0))),\tau(\sigma(\lambda(1,\iota(0)),\rho(1,\iota(1))),\iota(0)))) \\
\tau(\lambda(0,\tau(\sigma(\lambda(1,\iota(0)),\rho(1,\iota(1))),\iota(0))),\kappa(0,\iota(0),\iota(0)))
\end{array} \right.$$

Figure 3: Different representations of the same tree

Each of the identities is reflected by at least one of the different presentations given for the tree of Example 1.1. Equations (ASS1) and (ASS2) express the associativity of the substitution operation used in the definition of σ and τ. Equation (COM) takes into account that it is the same whether we put first a tree to the left of a node and then another tree to the right of that node or do it in the reversed order. In some sense ρ and λ 'commute'.

The equations in $(*)$ are the starting point for the definition of our notion of point-tree algebra. Consider the sort[3] set $\mathrm{S} = \{\mathrm{l}, \mathrm{s}, \mathrm{t}\}$, where we think of l as being the sort of letters, of s as being the sort of special trees, and of t as being the sort of ordinary trees. Besides, consider the typed signature $\Sigma = \{\iota_{(\mathrm{l},\mathrm{t})}, \lambda_{(\mathrm{lt},\mathrm{s})}, \rho_{(\mathrm{lt},\mathrm{s})}, \kappa_{(\mathrm{ltt},\mathrm{t})}, \sigma_{(\mathrm{ss},\mathrm{s})}, \tau_{(\mathrm{st},\mathrm{t})}\}$. Let $\mathrm{T}(A)$ be defined by

$$\mathrm{T}(A) = A_{(\mathrm{l})} \cup (\mathcal{S}(A) \setminus \{\cdot\})_{(\mathrm{s})} \cup \mathcal{T}(A)_{(\mathrm{t})} .$$

Together with the appropriate restrictions of the functions $\kappa, \rho, \lambda, \ldots$ the set $\mathrm{T}(A)$ forms a Σ-Algebra satisfying $(*)$. It is denoted by $\mathbf{T}(A)$. In general a Σ-algebra satisfying $(*)$ will be called a *point-tree algebra.*

The reason for that we do not allow the trivial special tree $\cdot$ as an element of sort s in $\mathbf{T}(A)$ will become clear in the next subsection.

We observe the following.

Remark 1.2 If $\mathbf{B}$ is a point-tree algebra, then the set $B_{(\mathrm{s})}$ together with the binary operation $\sigma^{\mathbf{B}}$ is a semigroup.

We introduce some conventions concerning the notation of terms in $\Gamma'(A)$. Let a stand for l-terms (either constants or variables), ϕ, ϕ_0, and ϕ_1 for s-terms, and ψ, ψ_0, ψ_1 for t-terms. We simply write $\phi_0\phi_1$ instead of $\sigma(\phi_0, \phi_1)$ and $\phi\psi$ instead of $\tau(\phi, \psi)$, since we have associativity for σ and τ. We also replace $\kappa(a, \psi_0, \psi_1)$ by $a(\psi_0, \psi_1)$ and sometimes write $a(\cdot, \psi)$ for $\rho(a, \psi)$ and $a(\psi, \cdot)$ for $\lambda(a, \psi)$. The single letter a stands for $\iota(a)$.

1.1 Free point-tree algebras

We show that the point-tree algebra $\mathbf{T}(A)$ is uniquely determined (up to isomorphism) by $(*)$, that is, $\mathbf{T}(A)$ is freely generated by the set A in the class of point-tree algebras. This is not true for the modified algebra including the special tree $\cdot$ as an element of sort s.

Theorem 1 (free point-tree algebras) *Let A be an arbitrary set (possibly infinite). Then $\mathbf{T}(A)$ is a Σ-algebra freely generated by $A_{(\mathrm{l})}$ in the class of point-tree algebras, i.e. in the class of all Σ-algebras satisfying $(*)$.*

[3]For fundamentals of many-sorted universal algebra we refer the reader to [EM85].

The remainder of this subsection is devoted to the proof of this theorem. We choose a proof using rewrite systems.[4]

Let $\mathbf{F}(A)$ denote the (free) Σ-term algebra generated by $A_{(1)}$, and let θ be the congruence relation generated by the instances of $(*)$ in $\mathbf{F}(A)$. Then, by Birkhoff's theorem[5], we know that the quotient algebra $\mathbf{F}(A)/\theta$ is freely generated by A/θ. Since distinct letters of A are non-equivalent with respect to θ and $\mathbf{T}(A)$ satisfies $(*)$, there is a unique homomorphism $h: \mathbf{F}(A)/\theta \to \mathbf{T}(A)$ with $h(a/\theta) = a$ for every $a \in A$. We have to show that h is an isomorphism. It is not hard to see that h is onto. For the injectivity we construct a rewrite system over $\mathrm{F}(A)$ such that the reflexive-symmetric-transitive closure of the corresponding reduction relation coincides with θ (Lemma 1.3), the system terminates (Lemma 1.4), and the classes of two irreducible elements go to distinct elements of $\mathcal{T}(A)$ under h (Lemma 1.6). Then h is one-to-one and we are done. So we are left with finding the rewrite system and proving the three lemmas about it.

We obtain the rewrite system R by reading $(*)$ from left to right, splitting (COM) into two rules.

$$\left.\begin{array}{lll}
(1) & & \sigma(s_0, \sigma(s_1, s_2)) \to \sigma(\sigma(s_0, s_1), s_2) \\
(2) & & \tau(\sigma(s_0, s_1), t) \to \tau(s_0, \tau(s_1, t)) \\
(3) & & \tau(\lambda(a, t_0), t_1) \to \kappa(a, t_0, t_1) \\
(4) & & \tau(\rho(a, t_1), t_0) \to \kappa(a, t_0, t_1)
\end{array}\right\} (**)$$

Lemma 1.3 *Over* $\mathrm{F}(A)$ *the reflexive-symmetric-transitive closure* $\leftrightarrow^*$ *of the rewrite relation* $\to$ *defined by* $(**)$ *coincides with the congruence* θ *generated by the instances of* $(*)$*, the equations defining the class of point-tree algebras.*

Proof. This is immediate by the construction of R. $\Box$

We need some definitions for the termination proof. We write $||\phi||_\rho$ for the number of occurrences of ρ in the term ϕ; in the same sense we use $||\phi||_\lambda$ and $||\phi||_\sigma$. Furthermore, if u is string over $\Sigma \cup A \cup \{(,)\}$ then $[u]$ is written for the string over $\{0,1\}$ which is obtained from u by reading it from right to left and noting 1 for each σ read, 0 for each element of A, and nothing for any other symbol. For a string v over $\{0,1\}$ let $\langle v \rangle$ designate the number whose binary representation is v.

Lemma 1.4 *The rewrite system* R *terminates over* $\mathrm{F}(A)$.

Proof. Associate with every term ϕ the natural number $\hat{\phi}$ defined by

$$\hat{\phi} = ||\phi||_\rho + ||\phi||_\lambda + ||\phi||_\sigma + \langle [\phi] \rangle \,.$$

[4]For background on rewrite systems, see [DJ90, p. 243–320].
[5]For background on universal algebra, see [BS81].

Since the 'less than' relation $<$ on the natural numbers is well-founded it would be sufficient to show that $\hat{\psi} < \hat{\phi}$ whenever $\phi \to \psi$. We prove this by case distinction.
1. case, $\phi \to^{(1)} \psi$. Then ϕ contains a subterm $\phi_0 = \sigma(\psi_0, \sigma(\psi_1, \psi_2))$ that is replaced by $\phi_1 = \sigma(\sigma(\psi_0, \psi_1), \psi_2)$ and

$$\begin{array}{rcl} [\phi_0] & = & [\psi_2][\psi_1]1[\psi_0]1\,, \\ [\phi_1] & = & [\psi_2][\psi_1][\psi_0]11\,. \end{array}$$

Since ψ_0 is ground we know that a 0 occurs in the string $[\psi_0]$, showing $\langle[\phi_0]\rangle > \langle[\phi_1]\rangle$. This immediately implies $\hat{\psi} < \hat{\phi}$.
2. case, $\phi \to^{(2)} \psi$. Then only the third term in the sum defining $\hat{}$ changes its value, and since the number of σ decreases, we see $\hat{\psi} < \hat{\phi}$.
3. case, $\phi \to^{(3)}$. In this case, only the second term in the definition of $\hat{}$ changes, and since one λ disappears, we also have $\hat{\psi} < \hat{\phi}$ in this case.
4. case, $\phi \to^{(4)}$. The same as in the third case since λ and ρ are treated dually. □

Next we describe the irreducible terms (normal forms) of $\mathrm{F}(A)$. For elements of sort t the situation is as follows: every term of sort t in normal form is built up from the elements of A only using the operation κ; the operations σ, λ and ρ are removed by applications of rule (2), (3), and (4), respectively. For instance, the first representation in Fig. 3 is the only irreducible representation of the sample tree, and any other representation given in the figure can be reduced to that.

Lemma 1.5 *The normal forms of the rewrite system R over* $\mathrm{F}(A)$ *are as follows.*

(1) *The set of normal forms of sort* l *is A.*

(2) *Every term $\iota(a)$, for $a \in A$, is a normal form of sort* t. *If ϕ and ψ are terms of sort* t *in normal form, then $\kappa(a, \phi, \psi)$ is in normal form for every $a \in A$.*

In this way every normal form of sort t *can be obtained.*

(3) *Every term $\rho(a, \psi)$ or $\lambda(a, \psi)$ with $a \in A$ and ψ of sort* t *in normal form is a term of sort* s *in normal form, called a* short term. *If ϕ of sort* s *is in normal form and ψ is a short term, then $\sigma(\phi, \psi)$ is also a term of sort* s *in normal form.*

In this way every normal form of sort s *can be obtained.*

Proof. We have to prove that all terms described above are irreducible (correctness) and that every irreducible term belongs to the terms described (completeness). Both proofs proceed by case distinction on the first symbol of a given term. We will only show the completeness, the proof for the correctness is similar.

Assume that ϕ is an irreducible term. There are six possible cases depending on the first symbol of ϕ.

1. case, $\phi = a$. Then ϕ belongs to the normal forms described in (1).
2. case, $\phi = \iota(a)$. Then ϕ belongs to the normal forms described in (2).
3. case, $\phi = \kappa(a, \psi_0, \psi_1)$. Then ψ_0 and ψ_1 must be in normal form, hence ϕ belongs to the normal forms of sort t described in (2).
4. case, $\phi = \tau(\psi_0, \psi_1)$. Then ψ_0 is of sort s, and thus must begin with σ, ρ, or λ — a contradiction: ϕ is not irreducible.
5. case, $\phi = \sigma(\psi_0, \psi_1)$. Then ψ_0 and ψ_1 must be terms of sort s in normal form. If ψ_1 would start with σ, then ϕ were reducible. Thus ψ_1 starts with λ or ρ, hence it is a short term. Then ϕ belongs to the terms of sort s described in (3).
6. case, $\phi = \rho(a, \psi)$ or $\phi = \lambda(a, \psi)$. Then ψ must be in normal form and ϕ is a short term.□

Lemma 1.6 *Let $h\colon \mathbf{F}(A)/\theta \to \mathbf{T}(A)$ be the unique homomorphism with $h(a/\theta) = a$ for every $a \in A$. Then $\phi \neq \psi$ implies $h(\phi\theta) \neq h(\psi\theta)$ whenever ϕ and ψ are normal forms.*

Proof. For elements of sort l the claim is immediate. For terms of sort s and t the claim can be proved by induction on the structure of the normal forms. We will do it only in the case of terms of sort t, the case of terms of sort s is similar.
Induction base. If $\phi = \iota(a)$ and $\psi \neq \phi$, then either $\psi = \iota(b)$ for some $b \neq a$ or $\psi = \kappa(c, \psi_0, \psi_1)$ for some c, ψ_0, ψ_1. In the first case $h(\psi/\theta) = b$, in the second case $h(\psi/\theta) = c(t_0, t_1)$. In both cases we have $h(\psi/\theta) \neq a = h(\phi/\theta)$.
Induction step. Let $\phi = \kappa(a, \phi_0, \phi_1)$ and suppose $\psi \neq \phi$ for a term ψ of sort t. Then either $\psi = \iota(b)$ for some b or $\psi = \kappa(b, \psi_0, \psi_1)$ for some b, ψ_0, ψ_1 where $b \neq a$, or $\psi_0 \neq \phi_0$, or $\psi_1 \neq \phi_1$. The first case was already treated in the induction base. In the second case we have $h(\phi/\theta) = a(h(\phi_0/\theta), h(\phi_1/\theta))$ and similarly $h(\psi/\theta) = b(h(\psi_0/\theta), h(\psi_1/\theta))$. If $a \neq b$ we can immediately derive $h(\phi/\theta) \neq h(\psi/\theta)$; otherwise, if $\psi_0 \neq \phi_0$, or $\psi_1 \neq \phi_1$, we derive the desired inequality from the induction hypothesis: $h(\psi_0/\theta) \neq h(\phi_0/\theta)$ or $h(\psi_1/\theta) \neq h(\phi_1/\theta)$.□

As a consequence of the termination of the rewrite system and the last theorem we obtain the following.

Corollary 1.7 (confluence) *The rewrite system R is confluent over* F(A).

1.2 Tree automata and regular tree languages

This subsection shows the close relation between finite point-tree algebras and tree (semi-) automata. As a consequence we will obtain a characterization of regularity for tree languages in terms of finite point-tree algebras. First, we will present the basic definitions and state the main result of this section.

A *tree semi-automaton* over an alphabet A is a triple $\mathfrak{A} = (Q, \delta, \beta)$ with a finite *state set* Q, a *transition function* $\delta: A \times Q \times Q \to Q$, and an *initial assignment* $\beta: A \to Q$. This assignment is inductively extended to an assignment with $\mathcal{T}(A)$ as domain: $\beta(a(t,t')) = \delta(a, \beta(t), \beta(t'))$ for all trees $t, t' \in \mathcal{T}(A)$ and every letter $a \in A$. We say that $\mathfrak{A}$ *recognizes* a tree language L over A if there is a *final state set* $F \subseteq Q$ such that $L = \{t \mid \beta(t) \in F\}$. A tree language over A is called *regular* if it is recognized by a tree semi-automaton over A.

A homomorphism $h: \mathbf{T}(A) \to \mathbf{B}$ is called a *recognizing homomorphism* over A. Such a homomorphism recognizes a tree language L over A if there exists a set $P \subseteq B_{(t)}$ with the property $h^{-1}(P) = L$, i.e. if L is an inverse image under h.

We can now state the following.

Proposition 1.8 *A tree language L over A is regular iff it is recognized by a homomorphism $h: \mathbf{T}(A) \to \mathbf{B}$ into a finite point-tree algebra.*

The claim is an immediate consequence of Remarks 1.9 and 1.10 of the subsequent paragraph.

The algebra associated with an automaton

In the following we explain in which way an automaton can be viewed as a point-tree algebra. Basically, the state set forms the elements of sort t, the translations on the state set induced by special trees are the elements of sort s, and the alphabet modulo an appropriate equivalence relation forms the set of elements of sort l.

Assume we are given a tree semi-automaton $\mathfrak{A} = (Q, \delta, \beta)$.

If q is a state of $\mathfrak{A}$ and if a is a letter of A, then a function $Q \to Q$ is defined by $p \mapsto \delta(a, q, p)$. Symmetrically, a function is defined by $p \mapsto \delta(a, p, q)$. Functions that are constructed in this way are called *elementary translations* of $\mathfrak{A}$. The set of all *translations* of $\mathfrak{A}$ is the smallest set containing every elementary translation and which is closed under composition (of functions in the usual way). Obviously, a translation can be associated with every special tree, for instance, if $\beta(a) = q$ for some letter a and if b is another letter, then $p \mapsto \delta(b, p, q)$ is associated with the special tree $b(\cdot, a)$.

We now define the point-tree algebra $\mathbf{PTA}(\mathfrak{A})$, the *point-tree algebra associated with* $\mathfrak{A}$. For ease in the definition, we write $\mathbf{B}$ for $\mathbf{PTA}(\mathfrak{A})$.

Let $B_{(\mathrm{l})}$ be the set A modulo the equivalence relation χ defined by $a \chi b$ iff $\beta(a) = \beta(b)$ and $\delta(a, q, q') = \delta(b, q, q')$ for all choices $q, q' \in Q$. Let $B_{(\mathrm{t})} = Q$ and let $B_{(\mathrm{s})}$ be the set of translations of $\mathfrak{A}$. The operations of $\mathbf{B}$ are defined in a natural way. For instance, if f is a translation and if q is an element of sort t (i.e. if q is a state of $\mathfrak{A}$), then $\tau(f, q)$ simply denotes the value of f when applied to q. Thus if the special tree s is associated with f and t is a tree such that $\beta(t) = q$, then $\beta(st) = \tau(f, q)$. The formal definitions are as follows:

$$
\begin{array}{ll}
\iota^{\mathbf{B}}(a/\chi) = \beta(a) & \text{for } a \in A, \\
\kappa^{\mathbf{B}}(a/\chi, q, q') = \delta(a, q, q') & \text{for } a \in A \text{ and } q, q' \in Q, \\
\rho^{\mathbf{B}}(a/\chi, q) = (q' \mapsto \delta(a, q', q)) & \text{for } a \in A \text{ and } q \in Q, \\
\lambda^{\mathbf{B}}(a/\chi, q) = (q' \mapsto \delta(a, q, q')) & \text{for } a \in A \text{ and } q \in Q, \\
\sigma^{\mathbf{B}}(f, f') = f \circ f' & \text{for } f, f' \in B_{(\mathrm{s})}, \\
\tau^{\mathbf{B}}(f, q) = f(q) & \text{for } f \in B_{(\mathrm{s})} \text{ and } q \in Q.
\end{array}
$$

Then, as one can easily check, B together with these operations forms a finite point-tree algebra.

Since $\mathbf{T}(A)$ is free on $A_{(\mathrm{l})}$ (Theorem 1) there is a unique homomorphism $h_{\mathfrak{A}}$ from $\mathbf{T}(A)$ to $\mathbf{PTA}(\mathfrak{A})$ with $h_{\mathfrak{A}}(a) = a/\chi$ for every $a \in A$. Clearly, $h_{\mathfrak{A}}(t) = \beta(t)$ for every $t \in \mathcal{T}(A)$, moreover, if F is an arbitrary subset of Q then a tree language L over A is recognized by $\mathfrak{A}$ together with the final state set F iff $h_{\mathfrak{A}}^{-1}(F) = L$. Thus $h_{\mathfrak{A}}$ recognizes every language recognized by $\mathfrak{A}$. This gives the following remark, which constitutes one direction of the proof of Proposition 1.8.

Remark 1.9 If $\mathfrak{A}$ is a tree semi-automaton and if L is a tree language recognized by $\mathfrak{A}$, then L is also recognized by $h_{\mathfrak{A}}$.

The homomorphism $h_{\mathfrak{A}}$ is called the *homomorphism associated with* $\mathfrak{A}$.

The automaton associated with a homomorphism

Now suppose that we are given a recognizing homomorphism $h\colon \mathbf{T}(A) \to \mathbf{B}$ into a finite point-tree algebra $\mathbf{B}$. We define a tree semi-automaton $\mathfrak{A}(h)$ by $\mathfrak{A}(h) = (B_{(\mathrm{t})}, \delta, \iota^{\mathbf{B}})$ with $\delta(a, q, q') = \kappa^{\mathbf{B}}(h(a), q, q')$. Then $\mathfrak{A}(h)$ is in fact a finite tree semi-automaton, which has the property $h^{-1}(P) = \{t \in \mathcal{T}(A) \mid \beta(t) \in P\}$ for every $P \subseteq B_{(\mathrm{t})}$. So we have the following remark, which gives us the other direction in the proof of Proposition 1.8.

Remark 1.10 If h is a recognizing homomorphism into a finite point-tree algebra and if L is recognized by h, then L is also recognized by $\mathfrak{A}(h)$.

The tree semi-automaton $\mathfrak{A}(h)$ is called the *automaton associated with* h.

We conclude this subsection with a description of $\mathfrak{A}(h_{\mathfrak{A}})$ and $h_{\mathfrak{A}(h)}$.

Lemma 1.11 (1) *If* $\mathfrak{A}$ *is a tree semi-automaton, then* $\mathfrak{A} = \mathfrak{A}(h_{\mathfrak{A}})$.

(2) *Let* $h: \mathbf{T}(A) \to \mathbf{B}$ *be a recognizing homomorphism onto a finite point-tree algebra. Then there exists a homomorphism* $g: \mathbf{B} \to \mathbf{PTA}(\mathfrak{A}(h))$ *such that* $h_{\mathfrak{A}(h)} = g \circ h$, *i.e.* $\mathbf{PTA}(\mathfrak{A}(h))$ *is a homomorphic image of* $\mathbf{B}$.

Proof. For (1) let us assume that $\mathfrak{A} = (Q, \delta, \beta)$ and let $\mathfrak{A}' = (Q', \delta', \beta')$ be the automaton associated with $h_{\mathfrak{A}}$. Write $\mathbf{B}$ for $\mathbf{PTA}(\mathfrak{A})$. By the definitions of $\mathfrak{A}'$ and $\mathbf{B}$ we have $Q = Q'$, $\delta'(a, q, q') = \kappa^{\mathbf{B}}(h_{\mathfrak{A}}(a), q, q') = \delta(a, q, q')$, and $\beta'(a) = \iota^{\mathbf{B}}(a) = \beta(a)$. Thus $\mathfrak{A} = \mathfrak{A}(h_{\mathfrak{A}})$.

For the proof of (2) write $\mathbf{B}'$ for $\mathbf{PTA}(\mathfrak{A}_h)$ and h' for $h_{\mathfrak{A}(h)}$; define the equivalence ζ on $B_{(\mathrm{l})}$ by $a \zeta b$ iff $\iota(a) = \iota(b)$ and $\kappa(a, t, t') = \kappa(b, t, t')$ for $t, t' \in B_{(\mathrm{t})}$. Then $B'_{(\mathrm{l})} = A/\zeta$, $B'_{(\mathrm{t})} = B_{(\mathrm{t})}$, and $B'_{(\mathrm{s})} = \{(t \mapsto h(s)t): B_{(\mathrm{t})} \to B_{(\mathrm{t})} \mid s \in \mathcal{S}(A)\}$. Thus a homomorphism g satisfying the requirements of the lemma can be defined in a natural way by setting

$$\begin{array}{lcll} a & \mapsto & a/\zeta & \text{for } a \in B_{(\mathrm{l})}, \\ s & \mapsto & (t \mapsto h(s)t) & \text{for } s \in B_{(\mathrm{s})}, \\ t & \mapsto & t & \text{for } t \in B_{(\mathrm{t})}. \end{array}$$

Here, it is important that h is assumed to be surjective. Otherwise it would not be clear to which elements one should map the elements outside the range of h. $\square$

1.3 Syntactic point-tree algebras

In this subsection we will transform the concept of syntactic congruence resp. syntactic algebra to our situation of regular tree languages and point-tree algebras. We shall see that for every regular tree language L there exists a smallest (up to isomorphism) finite point-tree algebra that recognizes the given language. This algebra is called the syntactic point-tree algebra of L and we will obtain it as the factor algebra of the free point-tree algebra modulo an appropriate congruence $\cong_L$, called the syntactic congruence of L. Furthermore, as a consequence of the correspondence between recognizing homomorphisms and tree-semi automata (Subsection 1.2) the syntactic congruence and the syntactic point-tree algebra will turn out to be effectively computable.

Assume L is a tree language over A. Let $\cong_L$ be the relation on $\mathrm{T}(A)$ defined as follows:

$$\begin{array}{lcll} a & \cong_L & a' & \text{iff} \quad \begin{cases} \forall s \in \mathcal{S}(A) \quad (sa \in L \leftrightarrow sa' \in L) \\ \forall s \in \mathcal{S}(A)\, \forall t, t' \in \mathcal{T}(A) \quad (sa(t, t') \in L \leftrightarrow sa'(t, t') \in L) \end{cases} \\ s & \cong_L & s' & \text{iff} \quad \forall s_0 \in \mathcal{S}(A)\, \forall t \in \mathcal{T}(A) \quad (s_0 st \in L \leftrightarrow s_0 s' t \in L) \\ t & \cong_L & t' & \text{iff} \quad \forall s \in \mathcal{S}(A) \quad (st \in L \leftrightarrow st' \in L) \end{array}$$

(The symbols a, a' stand for letters, s, s' stand for special trees, and t, t' stand for ordinary trees.)

In other words, two elements of the same sort are equivalent if and only if they relate to L the same in every possible context. In fact, this equivalence relation is a congruence on $\mathbf{T}(A)$ and called the *syntactic congruence* of L. The algebra $\mathbf{SA}(L) = \mathbf{T}(A)/\cong_L$ is called the *syntactic point-tree algebra* of L and the factorizing homomorphism $h_L\colon \mathbf{T}(A) \to \mathbf{SA}(L)$ is called the *syntactic homomorphism* of L. As indicated above we have the following.

Lemma 1.12 (syntactic congruence) *Let L be a tree language over A.*

(1) *The relation $\cong_L$ is the greatest congruence on $\mathbf{T}(A)$ such that L is a union of classes.*

(2) *L is regular iff $\cong_L$ is of finite index.*

(3) *If L is regular, then*

- *the syntactic homomorphism h_L is the homomorphism associated with the minimal tree semi-automaton $\mathfrak{A}_L$,*
- *the minimal tree semi-automaton $\mathfrak{A}_L$ is the automaton associated with h_L,*
- *the syntactic point-tree algebra $\mathbf{SA}(L)$ is effectively computable.*

Proof. The proofs for (1) and (2) are standard. For a version concerning regular sets of finite trees see, for instance, the recent contribution [Koz92]. However, the proof has to be adapted to point-tree algebras as recognizing structures.

The correspondence between $\mathfrak{A}_L$ and h_L can be seen immediately from the fact that $\mathfrak{A}_L = (\mathcal{T}(A)/\cong_L, \delta, \beta)$ where $\delta(a, [t], [t']) = [a(t, t')]$ for all $t, t' \in \mathcal{T}(A)$ and where $\beta(a) = [a]$. The effectivity of the construction of $\mathbf{SA}(L)$ follows from the fact that $\mathfrak{A}_L$ is effectively computable from any effective representation of L (regular expression, finite automaton). $\square$

To conclude we want to emphasize that (3) of the above lemma is relevant from the point of view of classification of regular tree languages. This will be explained in what follows.

We say that a class of regular tree languages is *decidable*, if there is a decision procedure that, given a tree semi-automaton $\mathfrak{A}$ and a final state set F, outputs 'yes ', if the language recognized by $\mathfrak{A}$ with F belongs to the given class, and 'no' otherwise. In the same way one can define what it means that a class of finite point-tree algebras is decidable. Then, as a consequence of Lemma 1.12, we obtain the following theorem.

Theorem 2 (decidability) *If $\mathcal{C}$ is a class of regular tree languages and $\mathcal{B}$ is a decidable class of finite point-tree algebras such that a language L belongs to $\mathcal{C}$ iff its syntactic point-tree algebra belongs to $\mathcal{B}$, then $\mathcal{C}$ is also decidable.*$\square$

2 Frontier testable tree languages

A language of finite words is reverse definite if membership is determined by the prefix of fixed maximal length of a given word. In the case of tree languages (reading trees from front to root, as the semi-tree automata introduced in Subsect. 1.2 do) this corresponds to the set of frontier trees of fixed maximal depth.

The set of *frontier trees* of a given tree (either ordinary or special) is defined as follows:

$$\begin{aligned} \mathrm{front}(a) &= a && \text{for } a \in A \cup \{\cdot\}, \\ \mathrm{front}(a(t_0,t_1)) &= \mathrm{front}(t_0) \cup \mathrm{front}(t_1) \cup \{a(t_0,t_1)\}. \end{aligned}$$

The *depth* of a tree is inductively defined by the following rules:

$$\begin{aligned} \mathrm{depth}(a) &= 1 && \text{for } a \in A \cup \{\cdot\}, \\ \mathrm{depth}(a(t_0,t_1)) &= \max\{\mathrm{depth}(t_0), \mathrm{depth}(t_1)\} + 1. \end{aligned}$$

Now the set of frontier trees of depth less than or equal to k of a tree t is defined by $\mathrm{front}_{\leq k}(t) = \{t' \in \mathrm{front}(t) \mid \mathrm{depth}(t') \leq k\}$.

We observe the following.

Remark 2.1 Let t be an arbitrary tree.

(1) The tree has depth $\geq k$ iff there is a tree of depth k in $\mathrm{front}_{\leq k}(t)$.

(2) The tree has depth $< k$ iff there is no tree of depth k in $\mathrm{front}_{\leq k}(t)$. If this is the case, there is a unique tree of maximal depth in $\mathrm{front}_{\leq k}(t)$, namely t itself, and $\mathrm{front}_{\leq k}(t) = \mathrm{front}(t)$.

A tree language L over A is *k-frontier testable* if, for $t, t' \in \mathcal{T}(A)$, either $t, t' \in L$ or $t, t' \notin L$. The language L is *frontier testable* if it is k-frontier testable for some k.

In the first subsection we give a characterization for k-frontier testable languages (with fixed parameter k). In Subsect. 2.2 we give applications of this result, which will then be extended to the general situation (without reference to a parameter) in Subsect. 2.3.

2.1 Characterization of k-frontier testability

Let $k > 0$.

Assume that t_0 is a variable of sort t and that $s_0, s_1, \ldots$ are distinct variables of sort s. Let t^k be a term of sort t inductively defined by $t^1 = t_0$ and $t^k = s_{k-2}t^{k-1}$ for $k > 1$. Then t^k covers exactly all trees of depth at least k.

Theorem 3 (**k-frontier testability**) *Let L be a tree language over the alphabet A. Then the following are equivalent:*

(A) *L is k-frontier testable.*

(B) *The syntactic point-tree algebra of L satisfies the identities*

$$\left.\begin{array}{ll} \text{(Sym)} & a(t^k, t) = b(t, t^k) \\ \text{(Idp)} & a(t^k, t^k) = t^k \\ \text{(Can)} & a(t, b(st, t^k)) = a'(st, t^k) \\ \text{(Rot)} & a(b(t, t^k), t') = a'(t, b'(t^k, t')) \end{array}\right\} (+)$$

for variables a, b, a', b' of sort l, *a variable s of sort* s, *variables t, t' of sort* t, *and where t^k is defined as above.*

(C) *L is recognized by a point-tree algebra satisfying the identities (+).*

An illustration of the four equations is given in Fig. 4. These pictures also explain the names of the equations: (Sym) stands for 'symmetry', (Idp) for 'idempotence', (Can) for 'cancellation', and (Rot) for 'rotation'. Equation (Sym) says that a tree of depth at least k can be turned around its root. Equation (Idp) allows a duplication of a tree of depth at least k, (Can) allows the elimination of the occurrence of a frontier tree which also occurs in the neighbourhood of a tree of depth at least k. Finally (Rot) expresses that trees can be rotated around a tree of depth at least k.

If we had allowed $\cdot$ as an element of $\mathrm{T}(A)_{(\mathrm{s})}$, then the equation

$$(\text{Can}_0) \qquad a(t, b(t, t^k)) = a(t, t^k)$$

would have been a special instance of (Can). Nevertheless, (Can$_0$) is a consequence of the other equations, as we can see from the following remark.

Remark 2.2 The equation (Can$_0$) is a consequence of (Sym), (Idp) and (Rot).

Proof. This assertion is proved by the following chain of equations.

$$\begin{array}{rcll} a(t, t^k) & = & a(a(t, t^k), a(t, t^k)) & \text{by (Idp)} \\ & = & a(t, b(t^k, a(t, t^k))) & \text{by (Rot)} \\ & = & a(t, b(a(t, t^k), t^k)) & \text{by (Sym)} \\ & = & a(t, b(t, a(t^k, t^k))) & \text{by (Rot)} \\ & = & a(t, b(t, t^k)) & \text{by (Idp)} \end{array}$$

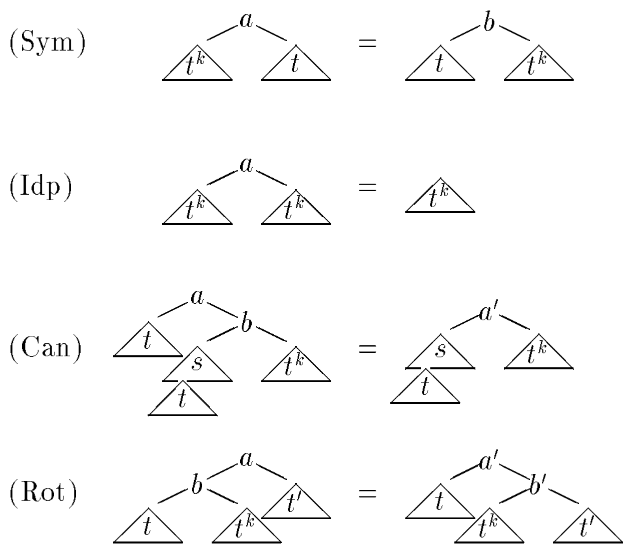


Figure 4: Graphical illustration of (+)

□

The remainder of this subsection is devoted to the proof of Theorem 3. Since equations are preserved under the application of homomorphisms we obtain, by using Lemma 1.12(1) and (2), that (B) and (C) are equivalent. So we are left with the implication from (A) to (C) (correctness) and the implication from (C) to (A) (completeness).

Correctness

It is sufficient to show that for every k-frontier testable tree language L over A there exists a finite point-tree algebra $\mathbf{B}$ satisfying (+) such that a homomorphism $h\colon \mathbf{T}(A) \to \mathbf{B}$ recognizes L. We shall obtain the algebra $\mathbf{B}$ as the quotient of $\mathbf{T}(A)$ modulo the congruence $\approx_k$ defined as follows:

$$\begin{array}{rcll} a & \approx_k & a' & \text{iff} \quad a = a' \\ s & \approx_k & s' & \text{iff} \quad \mathrm{front}_{\leq k}(s) = \mathrm{front}_{\leq k}(s') \\ t & \approx_k & t' & \text{iff} \quad \mathrm{front}_{\leq k}(t) = \mathrm{front}_{\leq k}(t') \end{array}$$

(The symbols a, a' stand for letters, s, s' stand for special trees, and t, t' stand for ordinary trees.)

Notice that we treat, according to the definition, special trees simply as trees (over the extended alphabet $\Gamma'(A)$).

As indicated above we will prove the following.

Lemma 2.3 (1) *The binary relation $\approx_k$ is a congruence of finite index (on $\mathbf{T}(A)$).*

(2) *A tree language over A is k-frontier testable iff it is a union of $\approx_k$-classes.*

(3) *The quotient algebra $\mathbf{T}(A)/\approx_k$ satisfies $(+)$.*

Proof. *Proof of (1) and (2).* Clearly $\approx_k$ is an equivalence relation of finite index and saturates every k-testable language over A, i.e. every k-testable language over A is a union of $\approx_k$-classes. For the rest of (1), namely that $\approx_k$ has the appropriate congruence properties, we have to check that $\approx_k$ is compatible with the functions $\iota, \rho, \lambda, \ldots$ We restrict ourselves to ρ and σ; the arguments in the other cases are similar.

Compatibility with ρ. Suppose $a \approx_k a'$ and $t \approx_k t'$ for $a, a' \in A$ and $t, t' \in \mathcal{T}(A)$. Then $a = a'$ and $\text{front}_{\le k}(t) = \text{front}_{\le k}(t')$, and we have to show $a(\cdot, t) \approx_k a'(\cdot, t')$. We proceed by case distinction on the depth of t.

Case $\text{depth}(t) < k$. Then $t = t'$ by Remark 2.1, thus $a(\cdot, t) = a'(\cdot, t)$, hence $a(\cdot, t) \approx_k a'(\cdot, t')$.

Case $\text{depth}(t) \ge k$. Then also $\text{depth}(t') \ge k$ by Remark 2.1. Therefore we may write $\text{front}_{\le k}(a(\cdot, t))$ as $\text{front}_{\le k}(t) \cup \{\cdot\}$ and also $\text{front}_{\le k}(a'(\cdot, t'))$ as $\text{front}_{\le k}(t') \cup \{\cdot\}$. This proves $a(\cdot, t) \approx_k a'(\cdot, t')$.

Compatibility with σ. Suppose $s \approx_k v$ and $s' \approx_k v'$ for $s, s', v, v' \in \mathcal{S}(A)$. We have to prove that $ss' \approx_k vv'$. In other words, we must show that the equivalence class of ss' depends only on $\text{front}_{\le k}(s)$ and $\text{front}_{\le k}(s')$. This can be seen from the following equation, which can easily be verified.

$$\begin{aligned}\text{front}_{\le k}(ss') = \text{front}_{\le k}(s') \ &\cup \ \text{front}_{\le k}(s) \cap \mathcal{T}(A)\\ &\cup \ \text{front}_{\le k}\left((\text{front}_{\le k}(s) \cap \mathcal{S}(A)) \odot \text{front}_{\le k}(s')\right)\end{aligned}$$

Here $\odot$ is a binary operation combining a set S of special trees with a subset of $\mathcal{S}(A) \cup \mathcal{T}(A)$ to the set $\{st \mid s \in S, t \in T\}$, which is also a subset of $\mathcal{S}(A) \cup \mathcal{T}(A)$. Besides, $\text{front}_{\le k}$ is applied to sets of trees: $\text{front}_{\le k}(M) = \bigcup_{t \in M} \text{front}_{\le k}(t)$ for a set M of trees (special or ordinary).

Proof of (3). We have to show: if $\phi = \psi$ is an equation of $(+)$ and t and t' are values of ϕ resp. ψ under an assignment, then $\text{front}_{\le k}(t) = \text{front}_{\le k}(t')$. To prove this one has to take into account that every interpretation of the term t^k yields a tree of depth at least k, and one uses the following relation, which can easily be derived from the definitions: $\text{front}_{\le k}(a(t, t')) = \text{front}_{\le k}(t) \cup \text{front}_{\le k}(t')$ for $t, t' \in \mathcal{T}(A)$ with $\text{depth}(t) \ge k$ or $\text{depth}(t') \ge k$. $\square$

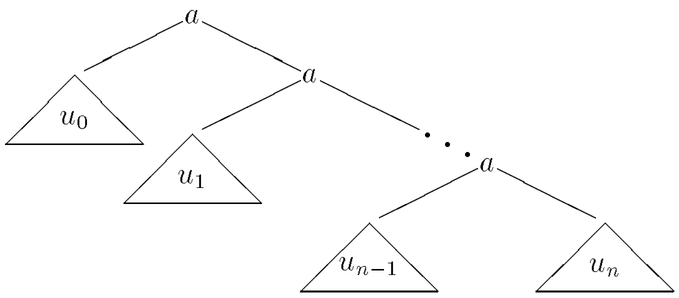


Figure 5: Graphical illustration of a tree in normal form

Completeness

We want to prove the implication from (C) to (A) in Theorem 3. Therefore we fix a recognizing homomorphism $h\colon \mathbf{T}(A) \to \mathbf{B}$ where $\mathbf{B}$ satisfies (+). We have to show $h(t) = h(t')$ whenever $t \approx_k t'$ for $t, t' \in \mathcal{T}(A)$. If $\mathrm{depth}(t) < k$ then $t = t'$ and the claim is trivial. So we are left with trees of depth at least k. We will define a set of trees in so-called normal form that contains for every tree t of depth at least k an $\approx_k$-equivalent tree t' (possibly more than one) such that $t \approx_k t'$ and $h(t) = h(t')$ (Lemma 2.4). We will furthermore show that all $\approx_k$-equivalent trees in normal form have the same image under h (Lemma 2.8). This will then be enough for the claim.

For notational convenience we will often write $t \approx^h t'$ for $h(t) = h(t')$ and in the same sense $t \approx^h_k t'$ iff $t \approx^h t'$ and $t \approx_k t'$, i.e. $\approx^h_k = \approx^h \cap \approx_k$.

Trees in normal form. Let $a \in A$ be a fixed letter. We say that a tree t is a *comb* with *teeth* $u_0, \ldots, u_n$ and width $n+1$ if t has the form

$$a(u_0, a(u_1, \ldots, a(u_{n-1}, u_n)\ldots)).$$

The tree is denoted by $\eta(u_0, \ldots, u_n)$. For a graphical representation see Fig. 5.

We say that a comb $\eta(u_0, \ldots, u_n)$ is in *normal form* if its width is at least 2, $\mathrm{depth}(u_n) = k$ and, for i with $i \leq n$, $\mathrm{depth}(u_i) \leq k$. Furthermore, we demand $\mathrm{front}_{\leq k}(t) = \{u_0, \ldots, u_n\}$. The requirement of u_n being of depth k allows us to reformulate this condition on the frontier trees of t: for every i with $i \leq n$ we need $\mathrm{front}_{\leq k}(u_i) \subseteq \{u_0, \ldots, u_n\}$.

In what follows the reader has to keep in mind that every transformation according to an equation of (+) does not only preserve $\approx^h$-equivalence but also ensures $\approx_k$-equivalence, i.e, $\approx^h_k$-equivalence, because (+) is correct (Lemma 2.3).

The first lemma tells us that every tree of depth at least k is $\approx^h_k$-equivalent to a tree in normal form.

Lemma 2.4 *For every tree t of depth at least k, there exists a $\approx_k^h$-equivalent tree t' in normal form such that $t \approx_k^h t'$.*

Proof. The proof goes by induction on the structure of t.
Induction base, $\text{depth}(t) = k$. By (Idp) the tree t can be transformed into $a(t,t)$. If t' is a proper frontier tree of t then t and $a(t,t)$ can be written as st' and $a(st',t)$, respectively, and an application of (Can) transfers $a(t,t)$ into $a(t', a(t,t)) = \eta(t',t,t)$. Repeated applications of this argument yield a comb $\eta(u_0, u_1, \ldots, u_n, t, t)$ which is $\approx_k^h$-equivalent to t and such that $\text{front}_{\leq k}(t) = \{t\} \cup \{u_0, \ldots, u_n\}$.
Induction step, $\text{depth}(t) > k$. Let $t = a(t_0, t_1)$. We proceed by case distinction on the depth of t_1.
Case 1, $\text{depth}(t_1) \geq k$. By induction hypothesis there exists t'' in normal form with $t'' \approx_k^h t_1$. A case distinction on the depth of t_0 is helpful.
Case 1.a, $\text{depth}(t_0) \leq k$. The tree t is $\approx_k^h$-equivalent to $a(t_0, t'')$, so in a similar way as before a repeated application of (Can) yields the desired tree.
Case 1.b, $\text{depth}(t_0) > k$. By induction hypothesis there exists a tree t' in normal form with $t' \approx_k^h t_0$. Let $t' = \eta(u_0, \ldots, u_m)$ and $t'' = \eta(v_0, \ldots, v_n)$. For the rest it suffices to prove $a(t', t'') \approx_k^h \eta(u_m, \ldots, u_0, v_0, \ldots, v_n)$. (Observe that the tree on the right hand side is in normal form.) This equivalence follows immediately from the lemma subsequent to this proof.
Case 2, $\text{depth}(t_1) < k$. Then $\text{depth}(t_0) \geq k$, and an application of (Sym) yields $t \approx_k^h a(t_1, t_0)$: we are in Case 1. □

Lemma 2.5 *Let $t = \eta(u_0, \ldots, u_m)$ and $t' = \eta(v_0, \ldots, v_n)$ be combs with* $\text{depth}(v_n) =$ *k. Then $a(t,t')$ is $\approx_k^h$-equivalent to $\eta(u_m, \ldots, u_0, v_0, \ldots, v_n)$.'*

Proof. The proof goes by induction on m. The induction base ($m = 0$) is trivial.
Induction step, $m > 0$. Write $\eta(u_0, \ldots, u_m)$ as $a(u_0, \eta(u_1, \ldots, u_m))$. Then

$$\begin{array}{rcll} a(t,t') & \approx_k^h & a(a(u_0, \eta(u_1, \ldots, u_m)), t') & \\ & \approx_k^h & a(a(\eta(u_1, \ldots, u_m), u_0), t') & \text{by (Sym)} \\ & \approx_k^h & a(\eta(u_1, \ldots, u_m), a(u_0, t')). & \text{by (Rot)} \end{array}$$

And an application of the induction hypothesis does the rest. □

We now show that adjacent teeth commute.

Lemma 2.6 *Let $t = \eta(u_0, \ldots, u_n)$ be a tree in normal form. If $i < n-1$, or $i = n-1$ and* $\text{depth}(u_{n-1}) = k$, *then $t \approx_k^h \eta(u_0, \ldots, u_{i-1}, u_{i+1}, u_i, u_{i+2}, \ldots, u_n)$, and the tree on the right hand side is also in normal form.*

Proof. The case $i = n-1$ is an immediate consequence of (Sym). For the other cases write t as $s\eta(u_i, \ldots, u_n)$ for an appropriate special tree s. Then

$$\begin{array}{rcll}
t & = & s\eta(u_i, \ldots, u_n) & \\
 & = & sa(u_i, a(u_{i+1}, \eta(u_{i+2}, \ldots, u_n))) & \\
 & \approx_k^h & sa(a(u_{i+1}, \eta(u_{i+2}, \ldots, u_n), u_i)) & \text{by (Sym)} \\
 & \approx_k^h & sa(u_{i+1}, a(\eta(u_{i+2}, \ldots, u_n), u_i)) & \text{by (Rot)} \\
 & \approx_k^h & sa(u_{i+1}, a(u_i, \eta(u_{i+2}, \ldots, u_n))) & \text{by (Sym)} \\
 & = & s\eta(u_{i+1}, u_i, u_{i+2}, \ldots, u_n).\Box &
\end{array}$$

Next, we observe that multiple occurrences of a teeth can be eliminated.

Remark 2.7 If $t = \eta(u_0, \ldots, u_n)$ is a tree in normal form and if $0 \leq i < n$ and $u_i = u_{i+1}$, then $t \approx_k^h \eta(u_0, \ldots, u_{i-1}, u_i, u_{i+2}, \ldots, u_n)$, and the tree on the right hand side is also in normal form.

This is an immediate consequence of (Can_0) (and (Sym) in the case of combs of width 2).

Combining the foregoing lemma and remark we can conclude the following.

Lemma 2.8 *Two $\approx_k$-equivalent trees in normal form are also $\approx^h$-equivalent.*

Proof. If t and t' are $\approx_k$-equivalent trees in normal form, they have the same set of teeth.

Let $t_0, \ldots, t_l$ be an enumeration of all trees of depth $\leq k$ (over the given alphabet). By Lemma 2.6 we know that the $\approx^h$-class of a tree in normal form is independent from the order of the teeth. So we can assume t and t' given as $\eta(t_{i_0}, \ldots, t_{i_m})$ and $\eta(t_{j_0}, \ldots, t_{j_n})$ such that

- $0 \leq i_0 \leq i_1 \leq \ldots i_m \leq l$,
- $0 \leq j_0 \leq j_1 \leq \ldots j_n \leq l$,
- $\{i_0, \ldots, i_m\} = \{j_0, \ldots, j_n\}$.

Now, several applications of the previous remark, saying that multiple occurrences of a tooth can be reduced to one, yield that t and t' are $\approx^h$-equivalent. $\Box$

Completeness proof. As pointed out at the beginning of this paragraph, for the completeness of (+) we have to show that $\approx_k$-equivalent trees t and t' go to the same element under h: By Lemma 2.4 there exist $\approx_k$-equivalent trees t_0 and t_0' such that $t_0 \approx^h t$ and $t_0' \approx^h t'$; due to Lemma 2.8 we may write $t_0 \approx^h t_0'$, hence $t \approx^h t'$, thus $h(t) = h(t')$.$\Box$

This also completes the proof of Theorem 3.$\Box$

2.2 Decidability

We show that Theorem 3 implies the decidablity of the class of frontier testable tree languages, and outline an efficient algorithm.

We need the following lemma.

Lemma 2.9 *If* $\mathbf{B}$ *is a point-tree algebra satisfying* $(+)$ *for some* k *and if* $l = |B_{(t)}|$, *then* $\mathbf{B}$ *satisfies* $(+)$ *also for* $k = l + 1$.

Proof. Let m be the smallest number such that $\mathbf{B}$ satisfies $(+)$ with $k = m$. If $m \leq l + 1$, there is nothing to show. If $m > l + 1$ we find an equation $\phi = \psi$ in $(+)$ and an assignment α to the variables such that the equation $\phi = \psi$ does not hold for $k = m - 1$. Consider the values $q_1, \ldots, q_{m-1}$ defined by $q_i = \alpha(t^{m-1})$. Since $m - 1 > l$, there exist i and j such that $i < j$ and $q_i = q_j$. Thus, using a pumping argument, we can find an assignment α' that coincides with α on the variables distinct from $t_0, s_0, s_1, \ldots$ and such that $\alpha'(t^{m-1+j-i}) = \alpha(t^{m-1})$. Let $r = m - 1 + j - i$ and let ϕ' and ψ' be the terms obtained from ϕ and ψ, respectively, by replacing t^{m-1} by t^r. By the construction of α' we know $\alpha'(\phi') = \alpha(\phi)$ and $\alpha'(\psi') = \alpha(\psi)$. From $m-1+j-i \geq m$ and the assumption that $\mathbf{B}$ satisfies $(+)$ for $k = m$ we may conclude $\alpha'(\phi') = \alpha'(\psi')$, hence $\alpha(\phi) = \alpha(\psi)$, which is a contradiction. □

Corollary 2.10 [Heu89] *If* L *is a frontier testable tree language and its minimal automaton* $\mathfrak{A}_L$ *has* l *states, then* L *is* $(l + 1)$*-frontier testable.*

Proof. From Lemma 1.12(3) we know that $\mathbf{PTA}(\mathfrak{A}_L)$ is the syntactic point-tree algebra of L, so it satisfies $(+)$ for some k. By construction of $\mathbf{PTA}(\mathfrak{A}_L)$ we know that it has exactly l elements of sort t. So $\mathbf{PTA}(\mathfrak{A}_L)$ satisfies $(+)$ with $k = l + 1$ by the previous lemma, which implies that L is $(l + 1)$-frontier testable by Theorem 3. □

As an immediate consequence of this we obtain the decidability of the class of frontier testable tree languages.

Corollary 2.11 [Heu89] *It is decidable whether a given regular tree language is frontier testable.*

Proof. We sketch a decision procedure: first compute the syntactic point-tree algebra of L (which can be done effectively, Lemma 1.12), then count the numbers of elements of sort t, say l, and check whether $(+)$ holds with $k = l + 1$. This can be done effectively since there are only a finite number of equations. □

A closer analysis of the equations shows the following.

Proposition 2.12 *Let A be a fixed alphabet. There is an $\mathcal{O}(l^3)$-time algorithm deciding whether a regular tree language over A is frontier testable. The tree language L is assumed to be given by its minimal automaton whose number of states is assumed to be l.*

Proof. We have to show that checking (+) for $k = l + 1$ and the syntactic point-tree algebra can be done in the given time. Let $\mathfrak{A}_L = (Q, \beta, \delta)$ be the minimal tree semi-automaton. We say that a state q is of height k if there is a tree with depth at least k such that $\beta(t) = q$. We say that a state q' is reachable from a state q if there is special tree s and a tree t such that $\beta(t) = q$ and $\beta(st) = q'$. Now, checking (Sym) simply amounts to check whether for any choice of letters a, b and states q, q' with q of height $l + 1$, the relation $\delta(a, q, q') = \delta(b, q', q)$ holds. (Idp) and (Rot) can be checked in a similar way. The verification of (Can) is more complicated: one has to check whether for any choice of letters a, b and states q, q', q'' such that q is of height k and q' is reachable from q'', the relation $\delta(a, \delta(b, q, q'), q'') = \delta(a, q, q')$ holds.

The set of states of height $l+1$ can be computed in time $\mathcal{O}(l^3)$ by a straightforward procedure. In the same time the reachability-relation is computable. Finally, it is important to observe that in any equation there are only three state variables that occur, resulting in an $\mathcal{O}(l^3)$-procedure to check the validity of the equations. $\square$

2.3 Characterization of frontier testable tree languages

The aim of this section is to present a (finite) base of identities for frontier testable tree languages (where the parameter k is not fixed). As is the case of reverse definite languages of finite words, it is impossible to find an appropriate set of equations over the given signature $\Gamma(A)$.

Proposition 2.13 *For every set E of equations in the signature Σ, if the syntactic point-tree algebras of all frontier testable tree languages satisfy E then so does also the syntactic point-tree algebra of a non-frontier testable tree language.*

Proof. We distinguish two cases.
1. case, $E \subseteq \{\phi = \phi \mid \phi \text{ term in } \Sigma\}$. Then every finite point-tree algebra satisfies E, in particular, the syntactic point-tree algebra of a regular non-frontier testable tree language.
2. case, $E \subsetneqq \{\phi = \phi \mid \phi \text{ term in } \Sigma\}$. Take an equation $\phi = \psi$ of E such that ϕ and ψ are distinct. W.l.o.g. we can assume that ϕ and ψ are of sort t. Let A be the alphabet that contains the variables occurring in ϕ and ψ. Then we can regard ϕ and ψ as distinct (ordinary) trees over A. Let k be the maximum of depth(ϕ) and depth(ψ) and let L be defined by $L = \{t \in \mathcal{T}(A) \mid \phi \in \text{front}_{\leq k}(t), \psi \notin \text{front}_{\leq k}(\psi)\}$. Then L is a k-frontier testable language but its syntactic point-tree algebra does not

satisfy the equation $\phi = \psi$ (since $\phi \in L$, but $\psi \notin L$), hence it does not satisfy E: a contradiction. $\square$

However, one could modify the notion of 'ultimately defined by an infinite sequence of equations' (e.g., see [Pin86]) known from finite semigroup theory to the tree case. But introducing implicit operations (see [Rei82]) is an even better remedy, for in our case the base of identities turns out to be finite. We do not want to transform the entire machinery of implicit operations and implicit equations (as elaborated in [Alm90]) to tree algebras, but confine ourselves to equations involving only (apart from symbols of Σ) the ω-operation from finite semigroup theory.

As pointed out before, if $\mathbf{B}$ is a finite point-tree algebra, then $B_{(\mathrm{s})}$ together with the binary function $\cdot: B_{(\mathrm{s})} \times B_{(\mathrm{s})} \to B_{(\mathrm{s})}$ defined by $s \cdot s' = \sigma^{\mathbf{B}}(s, s')$ forms a finite semigroup. Thus for every $s \in B_{(\mathrm{s})}$ there is a unique element s^ω in $\{s, s \cdot s, \ldots\}$ with $s^\omega \cdot s^\omega = s^\omega$. This justifies the following convention.
Convention. From now on every finite point-tree algebra $\mathbf{B}$ is viewed as a structure over the signature $\Sigma' = \Sigma \cup \{{}^{\omega}{}_{(\mathrm{s},\mathrm{s})}\}$, where ω maps every element $s \in B_{(\mathrm{s})}$ onto the unique element $e \in \{s, s \cdot s, \ldots\}$ with $\sigma^{\mathbf{B}}(e, e) = e$, in particular, this operation may also be used in equations for characterizing finite point-tree algebras.

In the context of frontier testable point-tree algebras the ω-operation can be understood as a tool which implicitly introduces a parameter that correlates with the degree of frontier testability. The correlation between the length of a decomposition of a semigroup element and the ω-operation is expressed in the following well-known lemma from finite semigroup theory.

Lemma 2.14 *If S is a finite semigroup of cardinality n and if $m \geq n$, then the following are equivalent for an element $s \in S$:*

(A) *There exist $s_0, \ldots, s_{m-1} \in S$, such that s can be written as $s_0 \ldots s_{m-1}$.*

(B) *There exist $s_0, s_1, s_2 \in S$ such that s can be written as $s_0 s_1^\omega s_2$.*

In our situation this extends to the following result.

Corollary 2.15 *If $\mathbf{B}$ is a finite point-tree algebra with $n = |B_{(\mathrm{s})}|$ and if $m > n$, then the following are equivalent for an element $t \in B_{(\mathrm{t})}$:*

(A) *There exist $t_0 \in B_{(\mathrm{t})}$ and $s_0, \ldots, s_{m-1} \in B_{(\mathrm{s})}$, such that t can be written as t^{m+1} (where t^{m+1} is defined as at the beginning of Subsect. 2.1).*

(B) *There exist $t_0 \in B_{(\mathrm{t})}$ and $s_0, s_1 \in B_{(\mathrm{s})}$ such that t can be written as $s_0 s_1^\omega t_0$.*

From this together with Theorem 3 we get the desired characterization of frontier testability.

Theorem 4 (frontier testability) *Let L be a tree language over A. The following are equivalent:*

(A) *L is frontier testable.*

(B) *The syntactic point-tree algebra of L satisfies the identities*

$$\left.\begin{array}{ll} (\mathrm{Sym}) & a(s_0 s_1^\omega t_0, t) = b(t, s_0 s_1^\omega t_0) \\ (\mathrm{Idp}) & a(s_0 s_1^\omega t_0, s_0 s_1^\omega t_0) = s_0 s_1^\omega t_0 \\ (\mathrm{Can}) & a(t, b(st, s_0 s_1^\omega t_0)) = a'(st, s_0 s_1^\omega t_0) \\ (\mathrm{Rot}) & a(b(t, s_0 s_1^\omega t_0), t') = a'(t, b'(s_0 s_1^\omega t_0, t')) \end{array}\right\} (++)$$

for variables a, b, a', b' of sort l, *s, s_0, s_1 of sort* s, *t_0, t, t' of sort* t.

(C) *L is recognized by a point-tree algebra satisfying the identities under* $(++)$.

□

As a consequence of the last theorem and Theorem 2, using also the fact that the ω-operation is effectively computable in finite semigroups, we obtain a second proof of the decidability of the class of frontier testable tree languages.

Corollary 2.16 [Heu89] *The class of frontier testable tree languages is decidable.*

We conclude with an example showing that within the semigroup approach frontier testability cannot be characterized.

Example 2.17 Let $A = \{\mathrm{a}, \mathrm{b}\}$. Let L be the set of all trees over A such that some leaf is labelled with a, i.e.

$$L = \{t \in \mathcal{T}(A) \mid \mathrm{a} \in \mathrm{front}_1(t)\}\,.$$

Let L' be the set of all trees over A such that some node is labelled with a, i.e.

$$L' = L \cup \{t \in \mathcal{T}(A) \mid \exists s \in \mathcal{S}(A)\; \exists t_0, t_1 \in \mathcal{T}(A) \quad (t = s\mathrm{a}(t_0, t_1))\}$$

On the one hand, L is frontier testable but L' is not. On the other hand, the syntactic semigroup of L' (in the sense of [NP89]) can be embedded in the syntactic semigroup of L. The syntactic semigroups of L and L' are isomorphic to the semigroups S and S', respectively, given by the following multiplication tables:

S	0	x	y
0	0	0	0
x	0	x	y
y	0	x	y

S'	0	x
0	0	0
x	0	x

So the class of syntactic semigroups corresponding to the class of frontier testable tree languages cannot be characterized by a set of equations (rather than by a pseudovariety of semigroups) since equationally defined classes of semigroups (and pseudovarieties) are closed under isomorphic copies and subsemigroups.

Discussion

We have seen that point-tree algebras are an appropriate structure for the characterization of the class of frontier testable tree languages. There is some hope that the results can be viewed as the starting point of a more exhaustive algebraic classification of regular tree languages. First of all it is not hard to extend these results to the more general case of generalized definite tree languages (cf. [Heu89b]). One only needs to combine them with the results on root testable languages presented in [NP89]. This is worked out in [Sch92]. Presumably a transformation of known results [Alm90] from universal algebra could provide an abstract framework for the classification of regular tree languages — we think of an Eilenberg correspondence as known for regular languages of finite [Eil76] and infinite [Wil91] words. (However, working with more than one sort, this might not be straightforward.) It would be desirable to try to characterize other, more complicated, classes of regular tree languages by using point-tree algebras. Perhaps a characterization of locally testable tree languages [Heu89, Ste92], a natural class of regular tree languages, can be obtained along these lines. In the corresponding word case the use of the wreath product is essential. Therefore it would be interesting to know what the appropriate notion of wreath product for point-tree algebras should be.

References

[Alm90] Jorge Almeida. On pseudovarieties, varieties of languages, filters of congruences, pseudoidentities and related topics. *Algebra Universalis*, 27:333–350, 1990.

[BS81] Stanley Burris and H.P. Sankappanavar. *A course in universal algebra*, volume 78 of *Graduate Texts in Mathematics*. Springer, New York, 1981.

[DJ90] N. Dershowitz and J.P. Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoret. Comput. Sci.: Formal Models and Semantics*, volume B, chapter 6. Elsevier, Amsterdam, 1990.

[Eil76] Samuel Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, New York, 1976.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1*, volume 6 of *EATCS Monographs on Theoret. Comput. Sci.* Springer, Berlin, 1985.

[Heu89] Uschi Heuter. Zur Klassifizierung regulärer Baumsprachen. Doctoral Thesis, RWTH Aachen, June 1989.

[Heu89b] Uschi Heuter. Generalized definite tree languages. In A. Kreczmar and G. Mirkowska, editors, *Mathematical Foundation of Computer Science 1989*, volume 379 of *Lecture Notes in Computer Science*, pages 270–280, Porabka-Kozubnik, August/September 1989. European Assoc. Theoret. Comput. Sci., Springer Verlag.

[Koz92] Dexter Kozen. On the Myhill-Nerode theorem for trees. *Bull. European Assoc. Theoret. Comput. Sci.*, 47:170–173, June 1992.

[NP89] Maurice Nivat and Andreas Podelski. Definite tree languages. *Bull. European Assoc. Theoret. Comput. Sci.*, 38:186–190, June 1989.

[Pin86] Jean Eric Pin. *Varieties of Formal Languages.* North Oxford Academic Press, London, 1986.

[PP92] Pierre Péladeau and Andreas Podelski. On reverse and general definite tree languages. In Werner Kuich, editor, *Automata, Languages and Programming: 19th Internat. Coll.*, volume 623 of *Lecture Notes in Computer Science*, pages 150–161, Wien, July 1992. European Assoc. Theoret. Comput. Sci., Springer Verlag.

[Rei82] Jan Reiterman. The Birkhoff Theorem for finite algebras. *Algebra Universalis*, 14:1–10, 1982.

[Sch92] Thorsten Scholz. Charakterisierung definiter Baumsprachen durch Gleichungen in der Termalgebra. Diploma thesis, Universität Kiel, Inst. f. Inform. u. Prakt. Math., Univ. Kiel, Germany, September 1992.

[Ste92] Magnus Steinby. A theory of tree language varieties. In Maurice Nivat and Andreas Podelski, editors, *Tree Automata and Languages*, pages 57–81. Elsevier Science Publishers, 1992.

[Tho84] Wolfgang Thomas. Logical aspects in the study of tree languages. In Bruno Courcelle, editor, *Ninth Coll. on Trees in Algebra and Programming*, pages 31–51. Cambridge Univ. Press, 1984.

[Wil91] Thomas Wilke. An Eilenberg theorem for ∞-languages. In J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, editors, *Automata, Languages and Programming: 18th Internat. Coll.*, volume 510 of *Lecture Notes in Computer Science*, pages 588–599, Madrid, 1991. European Assoc. Theoret. Comput. Sci., Springer Verlag. (Extended version in Internat. J. Algebra and Comput., to appear.)

[Wil93] Thomas Wilke. Algebras for classifying regular tree languages and an application to frontier testability. In A. Lingas, R. Karlsson, and S. Carlsson, editors, *Automata, Languages and Programming: 20th Internat. Coll.*, volume 700 of *Lecture Notes in Computer Science*, Lund, 1993. Europ. Assoc. Theoret. Comput. Science, Springer Verlag.