JOURNAL OF COMPUTER AND SYSTEM SCIENCES 15, 328–353 (1977)

IO and OI. I

JOOST ENGELFRIET[†] AND ERIK MEINECHE SCHMIDT

Aarhus University, Aarhus, Denmark

Received August 6, 1975; revised May 1, 1977

A fixed-point characterization of the inside-out (IO) and outside-in (OI) context-free tree languages is given. This characterization is used to obtain a theory of nondeterministic systems of context-free equations with parameters. Several "Mezei-and-Wright-like" results are obtained which relate the context-free tree languages to recognizable tree languages and to nondeterministic recursive program(scheme)s (called by value and called by name). Closure properties of the context-free tree languages are discussed. Hierarchies of higher level equational subsets of an algebra are considered.

1. INTRODUCTION

In theoretical computer science there are two basic ways of describing the meaning of a syntactical object: operational and equational. Operational semantics is defined by some effective (eventually nondeterministic) stepwise process which, from the syntactical object, generates its meaning. Equational semantics is defined by interpreting the syntactical object as a system of equations to be solved in some space of meanings. Usually the solution of the system of equations is obtained as the minimal fixed-point of a continuous mapping between partially ordered sets, and therefore equational semantics is also referred to as fixed-point semantics. An equation is of the form $A = \tau$, where A is an unknown and τ is a term (or tree) built up from the unknowns by symbols denoting the basic operations on the objects in the space of meanings. Together with the basic operations, this space can be considered as an algebra, and, to allow for solutions of equations, it should also be a partially ordered set such that the basic operations are continuous.

A well-known example of such a syntactical object is a context-free grammar which has a language as meaning. The operational semantics of the grammar is obtained by defining the notion of derivation, whereas the equational semantics is obtained by viewing the grammar as a set of BNF (or ALGOL-like) equations in the intuitively obvious way, and solving this set of equations in the (partially ordered) algebra of languages (with concatenation and union as basic operations). It was shown by Ginsburg and Rice [14] that these two semantics for a context-free grammar coincide. This result might be called a fixed-point characterization of the context-free languages.

[†] Present address: Twente University of Technology, Enschede, Netherlands.

ISSN 0022–0000

Another example of a syntactical object is a recursive program (note, however, that a context-free grammar may also be viewed as a nondeterministic recursive program with parameterless procedures). The operational semantics of a program is obtained by indicating a real or imaginary machine (or "computation rule") on which the program can be executed. The equational or fixed-point semantics is obtained by viewing the recursive program as a set of equations (with the names of the recursive procedures as unknowns), to be solved in an appropriate partially ordered space of functions or relations (with composition and "if–then–else" as basic operations). The fixed-point semantics for programs was first investigated for parameterless procedures (the "monadic case") and then for procedures with parameters (the "polyadic case"). It has been shown for certain classes of recursive programs that the operational semantics and the fixed-point semantics coincide (cf. [18]).

Polyadic procedures were introduced in formal language theory by Fischer [12] who defined macrogrammars, which are basically context-free grammars in which the nonterminals are allowed to have parameters. His "inside-out (IO)" and "outside-in (OI)" modes of derivation are two different operational semantics for macrogrammars corresponding to the two computation rules for recursive programs, "call by value" and "call by name," respectively. A fixed-point characterization of the OI macro-languages was given by Downey [8] and Nivat [23], whereas one for the IO macro-languages can be found in this paper (see also [43]).

It might now be asked what, in fact, one needs equational semantics for. First, equational semantics facilitates the task of proving correctness of programs or grammars, since it leads to useful and intuitively clear proof rules. Second, it provides a unification and simplification of several results in formal language theory and the theory of programs, like closure results, decidability results, and normal form lemmas. Third, it follows from the equational point of view (the fixed point of view) that a given system of equations can be solved in several different algebras. If there is a "meaning preserving" relationship (i.e., a homomorphism) between an algebra A and an algebra B, then the solution of the system in B is the homomorphic image of the solution in A. It follows from this simple fact that problems concerning equationally defined elements of B can be lifted to A, solved there, and projected down again. We shall give two examples. Mezei and Wright [21] and Thatcher and Wright [36] developed a general theory of equational subsets of an arbitrary algebra (for systems of "regular" equations). They showed that the solutions in the algebra of terms are the regular (recognizable) tree languages. Moreover, they showed that the solution of a system of regular equations in any algebra is the interpretation (i.e., homomorphic image) of its solution in the term algebra. Viewing a context-free grammar as a set of regular equations it then follows that every context-free language is the homomorphic image (yield) of a recognizable tree language. This result can be used to give "tree-oriented" proofs for context-free language results by lifting the problem to the tree level and applying the theory of recognizable tree languages (cf. [30, 35]). The theory of equational subsets of an algebra (in particular the algebra of strings) was developed further in [3, 5, 41]. As a second example, it was shown in [11] that a context-free grammar may be viewed as a nondeterministic monadic (i.e., parameterless) recursive program and vice versa. As a set of equations the grammar may then be solved in any space of relations over some domain (using composition of these relations as basic operation). Since there is a homomorphism from the algebra of string languages into the algebra of relations over a domain, it follows that the fixed-point semantics of any monadic recursive program is the homomorphic image of a context-free language and hence, by the result of Mezei and Wright, ultimately the homomorphic image of a recognizable tree language. This fact can be used to solve problems in the theory of program(scheme)s by lifting them to the theory of context-free languages (see, for instance, [1, 11, 13]).

Thus the existence of homomorphisms between algebras gives rise to "lifting of theories." We shall call such a result a "Mezei-and-Wright-like" result.

In this paper we investigate the equational approach to the (nondeterministic) polyadic case; that is, we investigate fixed-point semantics of IO and OI macrogrammars, and call by value and call by name (nondeterministic) recursive procedures with parameters. In our opinion deterministic recursive programs with tests also fit nicely into the framework of nondeterministic ones without tests, essentially because the "if–then–else" construction is a choice mechanism. In fact, we shall consider context-free tree grammars (IO and OI) which are generalizations of macrogrammars in exactly the same way as recognizable tree languages are a generalization of context-free languages (see above). We shall give an equational semantics for the IO and OI tree grammars and we shall use this fixed-point characterization of context-free tree languages for the goals of equational semantics mentioned above, trying to achieve results similar to those for the context-free languages in the monadic case (in particular Mezei-and-Wright-like results). Several results in this area already exist. As mentioned before, Downey [8] and Nivat [23] have given a fixed-point characterization for the OI tree languages. Nivat [23] and Goguen *et al.* [15, 16] show that the semantics of a deterministic program can be obtained as the homomorphic image of a "schematic OI tree language" or an "infinite context-free tree," respectively. This result can also be applied to the nondeterministic call by name programs by viewing the choice of an alternative as an operation (denoted by, say, $+$) in the algebra. The $+$ then appears as a symbol on the tree(s). Maibaum [17] shows that a context-free tree grammar can be viewed as a system of regular equations (with substitution of trees as a basic operation). Unfortunately all results in [17, Sections 9–12] are wrong, apparently because IO and OI are confused. We hope that this paper contains correct versions of Maibaum's results. Wand [42] shows, similarly to Downey [8], that systems of regular equations solved in the space of functions of languages (with composition and join of functions and concatenation of languages as basic operations) give precisely the OI string languages. Moreover, he shows that in general this process can be iterated, leading to functions of functions of languages, etc. By solving these higher level regular equations in function spaces over languages (using left concatenation with one symbol, and all types of composition of functions, as basic operations) this leads to a hierarchy of language classes starting with the regular languages, the context-free languages, and the OI string languages.

We shall obtain results for the IO and OI cases (which are essentially different in nature), showing the basic differences between these two concepts. On the other hand, a certain symmetry in the results can be detected due to the symmetry in their definition:

in the IO case one first chooses and then computes, whereas in the OI case one first computes and then chooses. The main differences between IO and OI are caused by the combination of nondeterminism (choosing) with the computational facilities of copying and deletion (cf. [9]). These differences are also reflected in the formal properties of the algebraic operations involved in the description of IO and OI. In the case of OI one has the nice property of associativity, leading to nice algebraic proofs (which could eventually be formulated in categorical terms); in the case of IO one has the nice property of "complete distributivity" (continuity), leading to straightforward generalizations of techniques concerning subset algebras.

This paper is divided into two parts and seven sections. Part I contains Sections 1–4; Part II contains Sections 5–7 and a Conclusion. To each of the parts a list of references is added. In Part I we give the fixed-point characterization of both the IO and OI tree languages. We show that a context-free tree grammar can be viewed as a system of regular equations to be solved in an algebra of tree languages. Part I can be read independently of Part II. In Part II the results of Part I are applied and generalized. The contents of Sections 2–7 will now be described.

Section 2 is concerned with terminology and basic facts. Continuous algebras are defined. Several properties of "completely continuous" algebras are shown. The latter type of algebra will be a major tool in the paper. Two kinds of substitution of tree languages are defined: the OI (or usual) substitution and the IO substitution (in which one has to substitute the same tree for all occurrences of one symbol). OI substitution is associative; IO substitution is only associative under certain restrictions.

In Section 3 (which can be read with the terminology of Section 2.1 only, together with some facts from Section 2.4) we present the fixed-point characterization of IO and OI tree languages. It turns out that one can use the algebra of tree languages (with variables) in both cases, with IO substitution as a basic operation in the IO case and OI substitution as basic operation in the OI case. Thus a simple change in basic operation of the underlying algebra explains in equational terms the difference between IO and OI operational semantics.

In Section 4 it is shown that both IO and OI tree grammars can be viewed as systems of regular equations in the tree language substitution algebras, and vice versa. It follows from this that the IO tree languages are the homomorphic images ("YIELDs") of recognizable tree languages (over the alphabet containing substitution operators: the so-called derived alphabet). For the OI case such a result cannot be obtained.

Section 5 is concerned with nondeterministic call by value and call by name recursive programs. They can be viewed as context-free tree grammars which on their turn can be viewed as systems of equations to be solved in the algebra of relations over a domain in the IO case and the algebra of functions of subsets of a domain in the OI case. We show the following Mezei-and-Wright-like results (lifting the fixed-point semantics to tree languages). In the IO case, the call by value relation computed by a program (i.e., IO tree grammar) over some domain is the homomorphic image of the IO tree language generated by the grammar, but only in case the basic operations over the domain are total (this excludes the use of tests). However, this relation can always (i.e., even if the basic operations are relations) be expressed as the homomorphic image of a recognizable tree language over the derived alphabet (the reader is asked to compare this with the monadic case discussed above). In the OI case, the call by name relation computed by the program (grammar) can always (except in the presence of "nonnaturally extended" basic operations) be expressed in terms of the homomorphic image of the OI tree language generated by the grammar (however, no result relating this relation to a recognizable "second level" tree language exists). We finally mention that both the call by value and the call by name relation can be obtained as the homomorphic image of an infinite recognizable tree (with union as a symbol on the tree), and we fit all these results into a diagram which neatly expresses the difference between IO and OI.

In Section 6 we apply the fixed-point characterization of Sections 3 and 4 to prove a closure result of the IO tree languages: they are closed under deterministic bottom-up tree transducer mappings. Two examples are given which show the nonclosure of the IO tree languages under (nondeterministic) relabeling and the nonclosure of the OI tree languages under tree homomorphisms.

In Section 7 we show how to obtain higher level equational hierarchies. We discuss an IO and an OI hierarchy, obtained by iterating the ideas of the previous sections (solving regular equations in algebras of higher level functions over domains). Mezei-and-Wright-like results similar to the simple case are shown. It is proved, using the result of Section 6, that, when starting with the monadic algebra of strings, the IO hierarchy starts with the regular languages, the context-free languages, and the IO languages. An analogous result is indicated for OI.

This paper might have been shorter. The length of the paper was motivated by our wish to be as precise as possible in order to avoid as many mistakes as possible. We hope that the reader will find it reasonably easy to read only the parts in which he is interested.

2. Terminology, Definitions, and Basic Facts

The reader is assumed to be familiar with the basic concepts of tree language theory (see, for instance, [15, 17, 21, 36]) and lattice theory (see, for instance, [31, 32]). For completeness sake we recall a number of them in this section. Moreover, we prove some basic properties of a few, perhaps less well-known, concepts. In particular we call the readers attention to the notion of a derived alphabet in Section 2.2, of a completely continuous algebra in Section 2.3, and the two different notions of tree language substitution in Sections 2.1 and 2.4.

2.1. *Ranked Alphabet, Tree Substitution, Context-Free Tree Grammar*

For any set A, $\mathcal{P}(A)$ denotes the set of all subsets of A. Whenever no confusion arises we shall identify a singleton $\{a\}$ with the element a. In this sense, $A \subseteq \mathcal{P}(A)$.

For any set S, S^* is the set of all strings over S. λ is the empty string, $\lg(w)$ is the length of w.

$\mathbb{N}$ denotes the set $\{0, 1, 2, \ldots\}$ of nonnegative integers.

A *ranked alphabet* (or *ranked operator domain*) Σ is an indexed family $\langle \Sigma_n \rangle_{n \in \mathbb{N}}$ of disjoint sets. A symbol f in Σ_n is called an *operator of rank n* (the intention being that f denotes an operation of n arguments; see Section 2.2). If $n = 0$, then f is also called a *constant*.

A ranked alphabet $\Sigma = \langle \Sigma_n \rangle_{n \in \mathbb{N}}$ is said to be *finite* if $\bigcup_{n \in \mathbb{N}} \Sigma_n$ is a finite set.

If Σ and Σ' are ranked alphabets, then their *union*, denoted by $\Sigma \cup \Sigma'$, is defined by $(\Sigma \cup \Sigma')_n = \Sigma_n \cup \Sigma'_n$ for all $n \in \mathbb{N}$.

For a ranked alphabet Σ, the set of *trees over* Σ (or Σ*-trees* or *terms over* Σ), denoted by T_Σ, is defined to be the smallest set of strings over $\Sigma \cup \{(,)\}$ such that $\Sigma_0 \subseteq T_\Sigma$ and, for $n \geqslant 1$, if $f \in \Sigma_n$ and $t_1, \ldots, t_n \in T_\Sigma$, then $f(t_1 \cdots t_n) \in T_\Sigma$.

A subset of T_Σ is called a Σ-tree language or a *tree language* over Σ.

If Y is a set (of symbols) disjoint with Σ, then $T_\Sigma(Y)$ denotes the set of trees $T_{\Sigma(Y)}$, where $\Sigma(Y)$ is the ranked alphabet with $\Sigma(Y)_0 = \Sigma_0 \cup Y$ and $\Sigma(Y)_n = \Sigma_n$ for $n \geqslant 1$. Thus the elements of Y are added as constants. We shall only be interested in the case where Y consists of "variables."

Let $X = \{x_1, x_2, x_3, \ldots\}$ be a fixed denumerable set of *variables*. Let $X_0 = \varnothing$ and, for $k \geqslant 1$, $X_k = \{x_1, \ldots, x_k\}$ (note that X is *not* meant to be a ranked alphabet, the elements of X are meant to be constants). For $k \geqslant 0$, $m \geqslant 0$, $t \in T_\Sigma(X_k)$, and $t_1, \ldots, t_k \in T_\Sigma(X_m)$, we denote by $t[t_1, \ldots, t_k]$ the result of *substituting* t_i for x_i in t. Note that $t[t_1, \ldots, t_k]$ is in $T_\Sigma(X_m)$. Note also that for $k = 0$ $t[t_1, \ldots, t_k] = t[\ ] = t$.

We now define substitution of tree languages. In general, whenever there is more than one possible object to substitute for a given symbol, the problem arises whether to substitute the same object for all occurrences of the symbol or to allow different objects to be substituted for different occurrences of the symbol. Although the latter kind of substitution is the usual one in language theory, the former kind has also been studied, in particular in fixed-point characterizations of classes of languages (see, for instance, the extended definable languages in [27] and the bottom-up tree transductions in [9]). In [41] the two notions of substitution are called "call by value" and "call by name" substitutions, respectively. Here we shall call them inside-out (IO) and outside-in (OI) substitutions, respectively.

2.1.1. DEFINITION. Let $k \geqslant 0$, $m \geqslant 0$, $L \in \mathscr{P}(T_\Sigma(X_k))$, and $L_1, \ldots, L_k \in \mathscr{P}(T_\Sigma(X_m))$.

The *IO substitution* of $L_1, \ldots, L_k$ into L, denoted by $L \underset{\text{IO}}{\leftarrow} (L_1, \ldots, L_k)$, is defined to be the tree language $\{t[t_1, \ldots, t_k] \mid t \in L \text{ and } t_i \in L_i \text{ for } 1 \leqslant i \leqslant k\}$.

The *OI substitution* of $L_1, \ldots, L_k$ into L, denoted by $L \underset{\text{OI}}{\leftarrow} (L_1, \ldots, L_k)$, is defined inductively as follows.

(i) For $f \in \Sigma_0$, $f \underset{\text{OI}}{\leftarrow} (L_1, \ldots, L_k) = \{f\}$.

(ii) For $1 \leqslant i \leqslant k$, $x_i \underset{\text{OI}}{\leftarrow} (L_1, \ldots, L_k) = L_i$.

(iii) For $n \geqslant 1$, $f \in \Sigma_n$ and $t_1, \ldots, t_n \in T_\Sigma(X_k)$,

$$f(t_1 \cdots t_n) \underset{\text{OI}}{\leftarrow} (L_1, \ldots, L_k) = \{f(s_1 \cdots s_n) \mid \text{ for } 1 \leqslant i \leqslant n,\ s_i \in t_i \underset{\text{OI}}{\leftarrow} (L_1, \ldots, L_k)\}.$$

(iv) For $L \subseteq T_\Sigma(X_k)$,

$$L \underset{\text{OI}}{\leftarrow} (L_1, ..., L_k) = \bigcup_{t \in L} t \underset{\text{OI}}{\leftarrow} (L_1, ..., L_k). \blacksquare$$

Substitution will be further treated in Section 2.4. Here we note the obvious fact that, for trees $t, t_1, ..., t_k$, $t \underset{\text{IO}}{\leftarrow} (t_1, ..., t_k) = t \underset{\text{OI}}{\leftarrow} (t_1, ..., t_k) = t[t_1, ..., t_k]$. We also note that for $k = 0$ $L \underset{\text{IO}}{\leftarrow} (L_1, ..., L_k) = L \underset{\text{OI}}{\leftarrow} (L_1, ..., L_k) = L$. For $k = 1$ we shall write $L \underset{\text{IO}}{\leftarrow} L_1$ rather than $L \underset{\text{IO}}{\leftarrow} (L_1)$, and similarly for OI.

Next we define the notion of a context-free tree grammar. It is an obvious generalization (but also a special case!) of the notion of a macrogrammar in [12]. Note that we do not specify an initial nonterminal.

A *context-free tree grammar* is a triple $G = (\Sigma, \mathscr{F}, P)$ where

Σ	is a finite ranked alphabet of *terminals*,
$\mathscr{F}$	is a finite ranked alphabet of *nonterminals* or *function symbols*, disjoint with Σ, and
P	is a finite set of productions (or rules) of the form $F(x_1 \cdots x_k) \to \tau$, where $k \geqslant 0, F \in \mathscr{F}_k$, and $\tau \in T_{\Sigma \cup \mathscr{F}}(X_k)$.

We shall use the convention that for $k = 0$ an expression of the form $F(\tau_1 \cdots \tau_k)$ stands for F. In particular, for $F \in \mathscr{F}_0$, a rule is of the form $F \to \tau$ with $\tau \in T_{\Sigma \cup \mathscr{F}}$.

For $F \in \mathscr{F}_k$, the *set of right-hand sides* of rules for F, denoted by rhs(F), is defined to be $\{\tau \in T_{\Sigma \cup \mathscr{F}}(X_k) \mid F(x_1 \cdots x_k) \to \tau \text{ is in } P\}$.

For a context-free tree grammar $G = (\Sigma, \mathscr{F}, P)$ we now define three direct derivation relations: the unrestricted, the inside-out, and the outside-in one. Let $n \geqslant 0$ and let $\sigma_1, \sigma_2 \in T_{\Sigma \cup \mathscr{F}}(X_n)$. We define $\sigma_1 \underset{\text{unr}}{\Rightarrow} \sigma_2$ if and only if there are a production $F(x_1 \cdots x_k) \to \tau$, a tree $\eta \in T_{\Sigma \cup \mathscr{F}}(X_{n+1})$ containing exactly *one* occurrence of x_{n+1}, and trees $\xi_1, ..., \xi_k \in T_{\Sigma \cup \mathscr{F}}(X_n)$ such that

$$\sigma_1 = \eta[x_1, ..., x_n, F(\xi_1 \cdots \xi_k)]$$

and

$$\sigma_2 = \eta[x_1, ..., x_n, \tau[\xi_1, ..., \xi_k]].$$

In other words, σ_2 is obtained from σ_1 by replacing a (occurrence of a) subtree $F(\xi_1 \cdots \xi_k)$ by the tree $\tau[\xi_1, ..., \xi_k]$.

The definition of $\sigma_1 \underset{\text{IO}}{\Rightarrow} \sigma_2$ is the same as that for $\sigma_1 \underset{\text{unr}}{\Rightarrow} \sigma_2$ except that the ξ's are required to be terminal trees ($\xi_1, ..., \xi_k \in T_\Sigma(X_n)$). The definition of $\sigma_1 \underset{\text{OI}}{\Rightarrow} \sigma_2$ is the same as that for $\sigma_1 \underset{\text{unr}}{\Rightarrow} \sigma_2$ except that η is required to be such that x_{n+1} does not occur in a subtree of η of the form $G(\tau_1 \cdots \tau_m)$; i.e., x_{n+1} does not occur in the argument list of a function symbol.

Let m stand for unr, IO, or OI. As usual, $\overset{*}{\underset{m}{\Rightarrow}}$ denotes the transitive-reflexive closure of $\underset{m}{\Rightarrow}$. For $k \geqslant 0$ and $\sigma \in T_{\Sigma \cup \mathscr{F}}(X_k)$ we define $L_m(G, \sigma) = \{t \in T_\Sigma(X_k) \mid \sigma \overset{*}{\underset{m}{\Rightarrow}} t\}$. $L_m(G, \sigma)$ is called the context-free *tree language m-generated by G from σ*. It is well known from [12], and we shall give an alternative proof in Section 3, that $L_{\text{OI}}(G, \sigma) = L_{\text{unr}}(G, \sigma)$. A tree language L over Σ is called an *IO (OI) tree language* if there is a context-free tree grammar $G = (\Sigma, \mathscr{F}, P)$ such that $L = L_{\text{IO}}(G, S)$ $(L = L_{\text{OI}}(G, S))$ for some $S \in \mathscr{F}_0$. For $k \geqslant 1$ (and eventually for $k = 0$) a tree language $L \subseteq T_\Sigma(X_k)$ is called an *IO (OI) tree language with variables* if there is a context-free tree grammar $G = (\Sigma, \mathscr{F}, P)$ such that $L = L_{\text{IO}}(G, F(x_1 \cdots x_k))$ $(L = L_{\text{OI}}(G, F(x_1 \cdots x_k)))$ for some $F \in \mathscr{F}_k$. It can easily be shown that $L \subseteq T_\Sigma(X_k)$ is an IO (OI) tree language with variables if and only if it is an IO (OI) tree language over the alphabet $\Sigma(X_k)$. Note also that, for any $\sigma \in T_{\Sigma \cup \mathscr{F}}(X_k)$, $L_{\text{IO}}(G, \sigma)$ is an IO tree language with variables (and similarly for OI).

Whenever we want to consider a context-free tree grammar G together with the mode of derivation $\underset{\text{IO}}{\Rightarrow}$, we say that G is an *IO tree grammar*. Similarly, if we intend $\underset{\text{IO}}{\Rightarrow}$, we say that G is an *OI tree grammar*.

2.2. *Many-Sorted Alphabet, Derived Alphabet, Σ-Algebra, Yield, Derived Operation*

In the rest of this section we present the algebraic tools needed in the sequel. For motivation and examples, see [4, 15].

Since we want to make use of many-sorted operator domains, of which the ranked operator domain is a special case, we shall give most of our definitions for the many-sorted case, leaving to the reader the specialization of these definitions to the ranked case.

Let S be a set (of *sorts*). An *S-sorted alphabet* (or *many-sorted alphabet* or *S-sorted operator domain*) Σ is an indexed family $\langle \Sigma_{w,s} \rangle_{\langle w,s \rangle \in S^* \times S}$ of disjoint sets. A symbol f in $\Sigma_{w,s}$ is called an *operator* of *type* $\langle w, s \rangle$, *arity* w, *sort* s, and *rank* $\lg(w)$. If $w = \lambda$, then f is also called a *constant* of sort s.

A ranked alphabet Σ will be considered to be the same as an S-sorted alphabet where S is a singleton, say $S = \{s\}$. The set $\Sigma_{s^n,s}$ is then identified with Σ_n.

We shall in fact mostly be interested in $\mathbb{N}$-sorted alphabets obtained from ranked alphabets as follows.

2.2.1. Definition. Let Σ be a ranked alphabet. The *derived* $\mathbb{N}$-sorted *alphabet of Σ*, denoted by $D(\Sigma)$ or simply D whenever Σ is understood, is defined as follows. Let, for each $n \geqslant 0$, $\Sigma'_n = \{f' \mid f \in \Sigma_n\}$ be a new set of symbols; let for each $n \geqslant 1$ and each i, $1 \leqslant i \leqslant n$, π_i^n be a new symbol (the ith *projection symbol* of sort n); and let, for each $n \geqslant 0$ and $k \geqslant 0$, $c_{n,k}$ be a new symbol (the (n, k)th *composition symbol*). Then

(i) $D_{\lambda,0} = \Sigma_0'$;

(ii) for $n \geqslant 1$, $D_{\lambda,n} = \Sigma'_n \cup \{\pi_i^n \mid 1 \leqslant i \leqslant n\}$;

(iii) for $n, k \geqslant 0$, $D_{\underbrace{nkk\cdots k}_{n\,\text{times}},k} = \{c_{n,k}\}$ (in particular, $D_{0,k} = \{c_{0,k}\}$), and

(iv) $D_{w,s} = \varnothing$ otherwise. ∎

Intuitively, whenever the elements of Σ are interpreted as operations, the c's will be interpreted as composition of these operations (they might therefore be called "second level" operators) and the π's as projections. Another interpretation of the c's will be as substitution of trees or tree languages (the π's are then interpreted as variables). We note that the primes on the elements of Σ in $D(\Sigma)$ are not needed but used to stress the difference between Σ and $D(\Sigma)$. The symbol $c_{0,0}$ is superfluous, but added for notational convenience.

For an S-sorted operator domain Σ we denote by T_Σ the family $\langle T_{\Sigma,s}\rangle_{s\in S}$, where the $T_{\Sigma,s}$ are sets of trees defined inductively as follows:

(i) for $s \in S$, $\Sigma_{\lambda,s} \subseteq T_{\Sigma,s}$,

(ii) for $n \geqslant 1$ and $s, s_1, \ldots, s_n \in S$, if $f \in \Sigma_{s_1\cdots s_n,s}$ and, for $1 \leqslant i \leqslant n$, $t_i \in T_{\Sigma,s_i}$, then $f(t_1 \cdots t_n) \in T_{\Sigma,s}$.

$T_{\Sigma,s}$ is called the set of *trees of sort s over* Σ. For a family $Y = \langle Y_s\rangle_{s\in S}$ of disjoint sets, the family $T_\Sigma(Y)$ is defined to be $T_{\Sigma(Y)}$ where $\Sigma(Y)$ is the S-sorted alphabet with $\Sigma(Y)_{\lambda,s} = \Sigma_{\lambda,s} \cup Y_s$ and, for $w \neq \lambda$, $\Sigma(Y)_{w,s} = \Sigma_{w,s}$. Note that for $S = \mathbb{N}$, Y is a ranked alphabet.

We now turn to interpretations of operator domains: Σ-algebras. A *Σ-algebra* (or many-sorted algebra) A consists of a family $\langle A_s\rangle_{s\in S}$ of (not necessarily disjoint) sets (A_s is called the *carrier* or *domain* of sort s of the Σ-algebra A) and for each $\langle w, s\rangle \in S^* \times S$ and each $f \in \Sigma_{w,s}$ an operation f_A "of type $\langle w, s\rangle$," i.e., f_A: $A_{s_1} \times A_{s_2} \times \cdots \times A_{s_n} \rightarrow A_s$ where $s_1 s_2 \cdots s_n = w$. If $n = 0$, then f_A is a constant, i.e., $f_A \in A_s$. Whenever A is understood, we shall denote f_A simply by f.

2.2.2. Example. Let D be the derived alphabet of the ranked alphabet Σ. We shall denote by $DT_\Sigma(X)$ the D-algebra which is defined as follows. The domain of sort n is $T_\Sigma(X_n)$. For $f \in \Sigma_n$, f' is the tree $f(x_1 \cdots x_n)$ (for $f \in \Sigma_0$, $f' = f$). For $n \geqslant 1$ and $1 \leqslant i \leqslant n$, $\pi_i^n = x_i$. For $n, k \geqslant 0$, $t \in T_\Sigma(X_n)$, and $t_1, \ldots, t_n \in T_\Sigma(X_k)$, $c_{n,k}(t, t_1, \ldots, t_n) = t[t_1, \ldots, t_n]$ (in particular, $c_{0,k}(t) = t$). We shall call $DT_\Sigma(X)$ the *tree substitution D-algebra.* ▌

2.2.3. Example. Let Σ be a ranked alphabet. Σ^*_0 can be made into a Σ-algebra A by defining for $f \in \Sigma_0$, $f_A = f$, and, for $n \geqslant 1$, $f \in \Sigma_n$ and $w_1, \ldots, w_n \in \Sigma^*_0$, $f_A(w_1, \ldots, w_n) = w_1 \cdots w_n$. Thus, every operator in Σ_n is interpreted as the (n-ary) operation of concatenation. ▌

A *nondeterministic Σ-algebra A* differs from an ordinary Σ-algebra in that its operations are "many-valued," i.e., f_A: $A_{s_1} \times A_{s_2} \times \cdots \times A_{s_n} \rightarrow \mathscr{P}(A_s)$. In other words, the f_A are relations rather than total functions (and for $n = 0$, f_A is a set).

Any Σ-algebra is also a nondeterministic Σ-algebra in the obvious way (recall that we identify singletons with their elements). A nondeterministic Σ-algebra for which all operations are partial functions is sometimes called a *partial Σ-algebra.* Nondeterministic Σ-algebras will only be used to construct ordinary Σ-algebras from: the subset algebra (Definition 2.3.2) and the algebra of relations (Definition 5.3).

If A and B are (ordinary) Σ-algebras, then B is a *subalgebra* of A if (1) for all $s \in S$, $B_s \subseteq A_s$, and (2) for every operator f of Σ, f_B equals f_A restricted to B (in particular, if f is a constant, then $f_B = f_A$).

If A and B are Σ-algebras, a *Σ-homomorphism h*: $A \rightarrow B$ is a family $\langle h_s\rangle_{s\in S}$ of mappings h_s: $A_s \rightarrow B_s$ such that (1) if $f \in \Sigma_{\lambda,s}$, then $h_s(f_A) = f_B$, and (2), if $f \in \Sigma_{s_1\cdots s_n,s}$ and

$a_i \in A_{s_i}$, then $h_s(f_A(a_1, \dots, a_n)) = f_B(h_{s_1}(a_1), \dots, h_{s_n}(a_n))$. Whenever s is understood we shall write h rather than h_s.

For a family $Y = \langle Y_s \rangle_{s \in S}$ of disjoint sets, $T_\Sigma(Y)$ is a Σ-algebra in the obvious way: $T_{\Sigma(Y),s}$ is the domain of sort s and $f_{T_\Sigma(Y)}(t_1, \dots, t_n) = f(t_1 \cdots t_n)$. It is well known (see, for instance, [4]) that $T_\Sigma(Y)$ is the *free Σ-algebra* with generators Y, i.e., given a Σ-algebra A and mappings h_s: $Y_s \to A_s$, there is a unique Σ-homomorphism $\bar{h}$: $T_\Sigma(Y) \to A$ extending the h_s (that is, $\bar{h}_s(y) = h_s(y)$ for all $y \in Y_s$). In particular, for each Σ-algebra A, there is a unique homomorphism from T_Σ to A, denoted by h_A.

2.2.4. Example. Let Σ be a ranked alphabet. Consider the Σ-algebra Σ^*_0 defined in Example 2.2.3. The unique Σ-homomorphism from T_Σ to Σ^*_0 is obviously the mapping which associates with each tree in T_Σ its yield (or frontier) in Σ^*_0. ▌

2.2.5. Example. Let D be the derived alphabet of Σ. Consider the tree substitution D-algebra $DT_\Sigma(X)$ of Example 2.2.2. The unique homomorphism from T_D to $DT_\Sigma(X)$ will also be called YIELD (see [17]). Thus YIELD: $T_D \to DT_\Sigma(X)$ associates with each "second level tree" in $T_{D,n}$ a Σ-tree with variables in $T_\Sigma(X_n)$. ▌

Let Σ be a finite S-sorted alphabet (that is, $\bigcup_{w,s} \Sigma_{w,s}$ is finite; note that S might be infinite). A tree language $L \subseteq T_{\Sigma,s}$ is ***recognizable*** if there exist a finite Σ-algebra A (i.e., A_s is finite for all s) and a subset F of A_s such that $L = h_s^{-1}(F)$ where $h_A = \langle h_s \rangle_{s \in S}$ is the unique Σ-homomorphism $T_\Sigma \to A$.

For an infinite S-sorted alphabet Σ, we say that $L \subseteq T_{\Sigma,s}$ is ***recognizable*** if there is a finite subalphabet Ω of Σ (i.e., $\Omega_{w,s} \subseteq \Sigma_{w,s}$ for all $\langle w, s \rangle \in S^* \times S$) such that $L \subseteq T_{\Omega,s}$ and L is recognizable as a subset of $T_{\Omega,s}$.

Now consider, for a finite S-sorted Σ, the ranked alphabet $\bar{\Sigma}$ such that $\bar{\Sigma}_n = \bigcup_{w,s} \{\Sigma_{w,s} \mid \lg(w) = n\}$. Obviously $T_{\Sigma,s} \subseteq T_{\bar{\Sigma}}$, and, moreover, it can easily be seen from the definition of $T_{\Sigma,s}$ that it is a recognizable subset of $T_{\bar{\Sigma}}$. We leave the proof of the following statement as an easy exercise to the reader: a tree language $L \subseteq T_{\Sigma,s}$ is recognizable if and only if it is recognizable as a subset of $T_{\bar{\Sigma}}$. Clearly the same is true for infinite Σ. From this fact it follows that most of the theory of recognizable tree languages can be carried over directly from the ranked case to the many-sorted case.

For simplicity, let Σ be a ranked alphabet. Any tree t in $DT_\Sigma(X)$ is also called a *derived operator*. Given a Σ-algebra A and $k \geqslant 0$, each $t \in T_\Sigma(X_k)$ can be interpreted as a function $A^k \to A$, called a *derived operation*, denoted by t_A or derop$_A(t)$, and defined as follows: for $a_1, \dots, a_k \in A$, $t_A(a_1, \dots, a_k) = \bar{a}(t)$ where $\bar{a}$: $T_\Sigma(X_k) \to A$ is the unique Σ-homomorphism such that $\bar{a}(x_i) = a_i$ for $1 \leqslant i \leqslant k$. Note that for $k = 0$, $t_A = h_A(t)$, where h_A is the unique homomorphism $T_\Sigma \to A$. In the S-sorted case one has to associate a sort s_i with x_i for $1 \leqslant i \leqslant k$, and consider $T_\Sigma(Y)$ where $Y_s = \{x_i \mid s_i = s\}$. Each tree $t \in T_\Sigma(Y)_s$ then gives rise to a derived operation t_A: $A_{s_1} \times \cdots \times A_{s_n} \to A_s$ in the same way as above.

Let Σ be a ranked alphabet with derived alphabet D and let A be a Σ-algebra. Then the *D-algebra of functions over A*, denoted by $\mathscr{F}(A)$, is defined as follows. For $n \geqslant 0$, $\mathscr{F}(A)_n$ is the set of all total functions $A^n \to A$; for $f \in \Sigma_n$, $f' = f_A$; π_i^n is the ith projection $A^n \to A$; and $c_{n,k}$ is composition of functions: $c_{n,k}(f, f_1, ..., f_n) = f \circ (f_1, ..., f_n)$, where $(f \circ (f_1, ..., f_n))(a_1, ..., a_k) = f(f_1(a_1, ..., a_k), ..., f_n(a_1, ..., a_k))$ (for $n = 0$, $c_{0,k}(f)(a_1, ..., a_k) = f$, i.e., $c_{0,k}(f)$ is the constant function f of k arguments; for $k = 0$, $c_{0,0}(f) = f$).

It is well known [6, III.3, Exercise 4; 15, Proposition 2.4] that $(t[t_1, ..., t_k])_A = t_A \circ (t_{1A}, ..., t_{kA})$. From this it easily follows that the mapping derop_A, which associates the derived operation t_A with each tree t, is a D-homomorphism from $DT_\Sigma(X)$ into $\mathscr{F}(A)$ (in fact the unique one). By restricting $\mathscr{F}(A)_n$ to derived operations one obtains therefore a sub-D-algebra of $\mathscr{F}(A)$: the *D-algebra of derived operations over A*, denoted by $\text{der}\mathscr{F}(A)$. Note that $\text{der}\mathscr{F}(A)$ is in fact the D-homomorphic image of T_D; thus it is the smallest subalgebra of $\mathscr{F}(A)$, i.e., the smallest class of functions containing the operations of A and the projections, and closed under composition.

Let $\tilde{D}$ be the $\mathbb{N}$-sorted alphabet consisting of the projection symbols and the composition symbols. In [6, Chap. III.3] a sub-$\tilde{D}$-algebra of $\mathscr{F}(A)$ is called a *clone*, and $\text{der}\mathscr{F}(A)$ is called the "clone of action of Σ on A." The relevance of clones to formal language theory has been shown by Blikle ([5], where clones are called inductive families of functions) and Wand [40].

We finally note that the D-algebras $DT_\Sigma(X)$ and $\text{der}\mathscr{F}(T_\Sigma)$ are isomorphic.

2.3. *Continuous Algebra, Subset Algebra*

In [3, 15, 39, 40] Σ-algebras are investigated which are at the same time posets such that the Σ-operations are continuous. Here we shall consider in particular "completely continuous" Σ-algebras.

Let A be a partially ordered set (poset) with partial ordering $\sqsubseteq$ and minimal element $\perp$. A nonempty subset A_1 of A is called *directed* if any two elements of A_1 have an upper bound in A_1. A is called *$\bigsqcup$-complete* (*Δ-complete*, *$\sqcup$-complete*) if every subset (every directed subset, every finite nonempty subset) A_1 of A has a least upper bound (or join) $\bigsqcup A_1$ in A (for $a_1, a_2 \in A$, $\bigsqcup\{a_1, a_2\}$ is denoted by $a_1 \sqcup a_2$). A $\bigsqcup$-complete poset is usually called a complete lattice. If B is another poset and $f\colon A \to B$, then f is called *$\bigsqcup$-continuous* (*Δ-continuous*) if $f(\bigsqcup A_1) = \bigsqcup f(A_1)$ for all subsets (all directed subsets) A_1 of A, whenever $\bigsqcup A_1$ exists. Note in particular that $\bigsqcup \phi = \perp$ and hence $f(\perp) = \perp$ for every $\bigsqcup$-continuous f (some authors exclude this case). We now define continuous algebras.

2.3.1. Definition. Let Σ be an S-sorted alphabet, and let A be a Σ-algebra such that each carrier A_s is a poset with minimal element. Let Z stand for Δ or $\bigsqcup$. Then A is called a *Z-continuous Σ-algebra* if its carriers are Z-complete and each of its operations is Z-continuous in each of its arguments, i.e., if $f_A\colon A_{s_1} \times \cdots \times A_{s_n} \to A_s$ and $a_i \in A_{s_i}$ (for $i \neq k$), then the function $\lambda x \cdot f_A(a_1, ..., a_{k-1}, x, a_{k+1}, ..., a_n)\colon A_{s_k} \to A_s$ is Z-continuous. ∎

Note that in any $\bigsqcup$-continuous Σ-algebra A, $f_A(a_1, ..., a_k, \perp, a_{k+1}, ..., a_n) = \perp$.

Note that the notion of a Δ-continuous Σ-algebra coincides with the one in [15]. In statements about Δ-continuous Σ-algebras we shall mostly assume that they have

$\sqcup$-complete carriers (and the particular algebras we shall consider, have $\sqcup$-complete carriers). Actually it would suffice for our purposes to assume $\sqcup$-completeness. Observe that if a Δ-complete poset A (with minimal element) has a countable basis (i.e., there is a countable subset of A such that every element of A is the join of elements from that subset, see [32]), then it is $\sqcup$-complete if and only if it is $\sqcup$-complete (we leave the straightforward proof to the reader). By the same argument, we shall often consider $\sqcup$-continuous homomorphisms rather than Δ-continuous, $\perp$-preserving, and $\sqcup$-preserving ones.

The definition of "subalgebra of a continuous algebra" is left to the reader.

We now take a closer look at $\sqcup$-continuous (or: *completely continuous*) Σ-algebras. The most common type of $\sqcup$-continuous Σ-algebra is the "subset algebra."

2.3.2. Definition. Let A be a nondeterministic Σ-algebra. The ***subset algebra of*** A, denoted by $\mathscr{P}(A)$, is the (deterministic) Σ-algebra with $\mathscr{P}(A_s)$ as carrier of sort s and, for $f \in \Sigma_{s_1 \cdots s_n, s}$ and $A_i \in \mathscr{P}(A_{s_i})$, $f_{\mathscr{P}(A)}(A_1, ..., A_n) = \bigcup \{f_A(a_1, ..., a_n) \mid a_i \in A_i$ for $1 \leqslant i \leqslant n\}$. ▌

It was noticed in [21], in the case that A is an ordinary Σ-algebra, that the operations $f_{\mathscr{P}(A)}$ are "completely distributive" (i.e., $\sqcup$-continuous in each of its arguments). The easy generalization of this fact to the nondeterministic (and many-sorted) case is left to the reader.

2.3.3. Lemma. *For each nondeterministic Σ-algebra A, $\mathscr{P}(A)$ is a $\sqcup$-continuous Σ-algebra (where $\sqcup$ is set-union).* ▌

In [11] the notion of a "cslm" is defined (to be used in program scheme theory). A cslm is in fact a $\sqcup$-continuous Σ-algebra A, where Σ is the ranked alphabet with $\Sigma_0 = \{e\}$ and $\Sigma_2 = \{*\}$, such that A is a monoid with respect to e_A and $*_A$. It was shown in [11] that free cslm's exist.

We now prove a lemma which enables us to show the existence of free $\sqcup$-continuous Σ-algebras. A Σ-homomorphism $h = \langle h_s \rangle_{s \in S}$ is called $\sqcup$-continuous if all h_s are $\sqcup$-continuous functions.

2.3.4. Lemma. *Let A be a Σ-algebra and let B be a $\sqcup$-continuous Σ-algebra. Let h be a Σ-homomorphism from A to B. Then h is uniquely extendable to a $\sqcup$-continuous Σ-homomorphism from $\mathscr{P}(A)$ to B.*

Proof. Obviously, the only way to extend h is by defining, for $A_1 \subseteq A_s$ ($s \in S$), $\bar{h}(A_1) = \sqcup \{h(a) \mid a \in A_1\}$. Then, clearly, $\bar{h}$ is $\sqcup$-continuous. It remains to show that $\bar{h}$ is a Σ-homomorphism:

(i) for f of rank 0,

$$\bar{h}(f_{\mathscr{P}(A)}) = \bar{h}(\{f_A\}) = h(f_A) = f_B\,;$$

(ii) for f of rank n and sets $A_1, \dots, A_n$,

$$\begin{aligned} &\bar{h}(f_{\mathscr{P}(A)}(A_1, \dots, A_n)) \\ &\quad = \bar{h}(\{f_A(a_1, \dots, a_n) \mid a_i \in A_i\}) \\ &\quad = \bigsqcup \{h(f_A(a_1, \dots, a_n)) \mid a_i \in A_i\} \\ &\quad = \bigsqcup \{f_B(h(a_1), \dots, h(a_n)) \mid a_i \in A_i\} \end{aligned}$$

and this is, by $\sqcup$-continuity of f_B in each of its arguments, equal to

$$f_B\left(\bigsqcup \{h(a_1) \mid a_1 \in A_1\}, \dots, \bigsqcup \{h(a_n) \mid a_n \in A_n\}\right) = f_B(\bar{h}(A_1), \dots, \bar{h}(A_n)). \quad \blacksquare$$

2.3.5. THEOREM. *$\mathscr{P}(T_\Sigma)$ is free in the class of $\sqcup$-continuous Σ-algebras with $\sqcup$-continuous Σ-homomorphisms (i.e., for each $\sqcup$-continuous Σ-algebra A there is a unique $\sqcup$-continuous Σ-homomorphism from $\mathscr{P}(T_\Sigma)$ to A).*

Proof. By the previous lemma, the unique Σ-homomorphism $h_A: T_\Sigma \to A$ is uniquely extendable to a $\sqcup$-continuous Σ-homomorphism $\bar{h}_A: \mathscr{P}(T_\Sigma) \to A$. Since the restriction to T_Σ of any Σ-homomorphism $\mathscr{P}(T_\Sigma) \to A$ is a Σ-homomorphism, $\bar{h}_A$ is unique. $\blacksquare$

For simplicity we restrict ourselves again to the case of a ranked alphabet Σ and leave the many-sorted case to the reader. From the previous theorem we immediately obtain the following one.

2.3.6. THEOREM. *For $k \geqslant 1$, $\mathscr{P}(T_\Sigma(X_k))$ is the free $\sqcup$-continuous Σ-algebra with generators X_k.*

Proof. Analogous to [15, Proposition 2.2]. $\blacksquare$

From this theorem it follows that any tree language with variables can, in a natural way, be considered as a derived operator for $\sqcup$-continuous Σ-algebras.

2.3.7. DEFINITION. Let $k \geqslant 0$. Any tree language $L \subseteq T_\Sigma(X_k)$ will also be called a *derived operator*. Given a $\sqcup$-continuous Σ-algebra A, L can be interpreted as a function $A^k \to A$, called a *derived operation*, denoted by L_A or $\text{derop}_A(L)$, and defined as follows: for $a_1, \dots, a_k \in A$, $L_A(a_1, \dots, a_k) = \bar{a}(L)$, where $\bar{a}: \mathscr{P}(T_\Sigma(X_k)) \to A$ is the unique $\sqcup$-continuous Σ-homomorphism such that $\bar{a}(x_i) = a_i$. $\blacksquare$

It is obvious that $L_A(a_1, \dots, a_k) = \bigsqcup_{t \in L} t_A(a_1, \dots, a_k)$, where t_A is the derived operation of t in the Σ-algebra A as defined in Section 2.2.

With respect to continuity the "language derived operations" behave as follows.

2.3.8. THEOREM. *For each tree language L and $\sqcup$-continuous Σ-algebra A, the derived operation L_A is Δ-continuous.*

Proof. Completely analogous to the proof of [16, Proposition 4.13]. $\blacksquare$

2.4. *Substitution, Associativity*

We now characterize tree language substitution (defined in Section 2.1) algebraically.

Let Σ be a ranked alphabet and D its derived alphabet. Recall that $DT_\Sigma(X)$ is the tree substitution D-algebra.

The ***tree language IO substitution algebra***, denoted by $\mathscr{P}(T_\Sigma(X))_{\text{IO}}$, is defined to be the subset D-algebra $\mathscr{P}(DT_\Sigma(X))$.

Obviously, for $L \subseteq T_\Sigma(X_n)$ and $L_1, ..., L_n \subseteq T_\Sigma(X_k)$, $c_{n,k}(L, L_1, ..., L_n) = L \underset{\text{IO}}{\leftarrow} (L_1, ..., L_n)$; moreover, $\pi_i^n = \{x_i\}$ and, for $f \in \Sigma_n$, $f' = \{f(x_1 \cdots x_n)\}$.

Note that, by Lemma 2.3.3, $\mathscr{P}(T_\Sigma(X))_{\text{IO}}$ is a $\sqcup$-continuous D-algebra. The unique $\sqcup$-continuous D-homomorphism from $\mathscr{P}(T_D)$ to $\mathscr{P}(T_\Sigma(X))_{\text{IO}}$ will be called YIELD (it is the extension of YIELD: $T_D \to DT_\Sigma(X)$, see Example 2.2.5; for $L \subseteq T_{D,n}$ $(n \in \mathbb{N})$, $\text{YIELD}(L) = \{\text{YIELD}(t) \mid t \in L\}$).

The ***tree language OI substitution algebra***, denoted by $\mathscr{P}(T_\Sigma(X))_{\text{OI}}$, is defined to be the D-algebra such that

(i) the domain of sort n is $\mathscr{P}(T_\Sigma(X_n))$;

(ii) for $f \in \Sigma_n$, $f' = \{f(x_1 \cdots x_n)\}$ (if $n = 0$, then $f' = \{f\}$);

(iii) for $n \geqslant 1$ and $1 \leqslant i \leqslant n$, $\pi_i^n = \{x_i\}$; and

(iv) for $n, k \geqslant 0$, $L \subseteq T_\Sigma(X_n)$ and $L_1, ..., L_n \subseteq T_\Sigma(X_k)$, $c_{n,k}(L, L_1, ..., L_n) = L_A(L_1, ..., L_n)$, where A is the subset Σ-algebra $\mathscr{P}(T_\Sigma(X_k))$ and L_A is the derived operation corresponding to L in this $\sqcup$-continuous Σ-algebra A (as defined in Definition 2.3.7).

It should be clear from the definitions that $c_{n,k}(L, L_1, ..., L_n) = L \underset{\text{OI}}{\leftarrow} (L_1, ..., L_n)$.

Note that it would be appropriate to denote $\mathscr{P}(T_\Sigma(X))_{\text{OI}}$ by $D\mathscr{P}(T_\Sigma(X))$ since its elements are derived operators. This would also nicely indicate the difference between IO $(\mathscr{P}(DT_\Sigma(X)))$ and OI $(D\mathscr{P}(T_\Sigma(X)))$. For notational reasons we prefer the chosen denotations.

The D-algebra $\mathscr{P}(T_\Sigma(X))_{\text{OI}}$ is Δ-continuous. (Proof: Since $c_{n,k}(L, L_1, ..., L_n)$ is defined as a derived operation it follows from Theorem 2.3.8 that $c_{n,k}$ is Δ-continuous in the last n arguments; it follows from the very definition of derived operation that $c_{n,k}$ is even $\sqcup$-continuous in its first argument.) It is easy to see that $\mathscr{P}(T_\Sigma(X))_{\text{OI}}$ is in general not $\sqcup$-continuous (for instance, $f(x_1x_1) \underset{\text{OI}}{\leftarrow} \{g, h\} \neq (f(x_1x_1) \underset{\text{OI}}{\leftarrow} g) \cup (f(x_1x_1) \underset{\text{OI}}{\leftarrow} h)$ and $f(x_1) \underset{\text{OI}}{\leftarrow} (x_1, \phi) \neq \phi$).

We now consider the question of associativity of substitution. It was shown in [15, Proposition 2.3] that tree substitution is associative, i.e., for $t \in T_\Sigma(X_n)$, $t_1, ..., t_n \in T_\Sigma(X_k)$, and $s_1, ..., s_k \in T_\Sigma(X_m)$, $(t[t_1, ..., t_n])[s_1, ..., s_k] = t[t_1[s_1, ..., s_k], ..., t_n[s_1, ..., s_k]]$. This result was a special case of the fact that, for any Σ-algebra A, $(t[t_1, ..., t_n])_A = t_A \circ (t_{1A}, ..., t_{nA})$. For OI substitution of tree languages we can prove exactly the same results in the same way. Associativity of OI substitution was proved originally in [33, Lemma 7.8].

2.4.1. Theorem. *Let A be a $\sqcup$-continuous Σ-algebra. Let $L \subseteq T_\Sigma(X_n)$ and $L_1, ..., L_n \subseteq T_\Sigma(X_k)$. Then $(L \underset{\text{OI}}{\leftarrow} (L_1, ..., L_n))_A = L_A \circ (L_{1A}, ..., L_{nA})$.*

Proof. The proof is completely analogous to that of [15, Proposition 2.4], using $\sqcup$-continuous Σ-homomorphisms and the free $\sqcup$-continuous Σ-algebra $\mathscr{P}(T_\Sigma(X_n))$ rather than Σ-homomorphisms and the free Σ-algebra $T_\Sigma(X_n)$, respectively. ▌

2.4.2. Corollary. *OI tree language substitution is associative, i.e., for* $Q \subseteq T_\Sigma(X_n)$, $L_1, \ldots, L_n \subseteq T_\Sigma(X_k)$ *and* $M_1, \ldots, M_k \subseteq T_\Sigma(X_p)$, $(Q \underset{\text{OI}}{\leftarrow} (L_1, \ldots, L_n)) \underset{\text{OI}}{\leftarrow} (M_1, \ldots, M_k) = Q \underset{\text{OI}}{\leftarrow} (L_1 \underset{\text{OI}}{\leftarrow} (M_1, \ldots, M_k), \ldots, L_n \underset{\text{OI}}{\leftarrow} (M_1, \ldots, M_k))$.

Proof. By the previous theorem, using $A = \mathscr{P}(T_\Sigma(X_p))$. ▌

IO tree language substitution is not associative in general. For instance, $(f(x_1x_2) \underset{\text{IO}}{\leftarrow} (x_1, x_1)) \underset{\text{IO}}{\leftarrow} \{a, b\} = f(x_1x_1) \underset{\text{IO}}{\leftarrow} \{a, b\} = \{f(aa), f(bb)\}$. But $f(x_1x_2) \underset{\text{IO}}{\leftarrow} (x_1 \underset{\text{IO}}{\leftarrow} \{a, b\}, x_1 \underset{\text{IO}}{\leftarrow} \{a, b\}) = f(x_1x_2) \underset{\text{IO}}{\leftarrow} (\{a, b\}, \{a, b\}) = \{f(aa), f(ab), f(ba), f(bb)\}$. Problems arise in $(Q \underset{\text{IO}}{\leftarrow} (L_1, \ldots, L_n)) \underset{\text{IO}}{\leftarrow} (M_1, \ldots, M_k)$ if a variable x_i occurs in two different L's and M_i contains at least two elements. If this does not happen, then IO substitution is associative, as shown next.

2.4.3. Lemma. *Let* $Q \subseteq T_\Sigma(X_n)$, $L_1, \ldots, L_n \subseteq T_\Sigma(X_k)$, *and* $M_1, \ldots, M_k \subseteq T_\Sigma(X_p)$. *Suppose that for all* i, $1 \leqslant i \leqslant k$, x_i *occurs at most in the trees of one of the* $L_1, \ldots, L_n$ *or* M_i *is a singleton.*

Then $(Q \underset{\text{IO}}{\leftarrow} (L_1, \ldots, L_n)) \underset{\text{IO}}{\leftarrow} (M_1, \ldots, M_k) = Q \underset{\text{IO}}{\leftarrow} (L_1 \underset{\text{IO}}{\leftarrow} (M_1, \ldots, M_k), \ldots, L_n \underset{\text{IO}}{\leftarrow} (M_1, \ldots, M_k))$.

Proof. Let t be in the left-hand side of the above equation. Then $t = (q[l_1, \ldots, l_n])[m_1, \ldots, m_k]$ with $q \in Q$, $l_i \in L_i$, and $m_i \in M_i$. Hence, by associativity of tree substitution, $t = q[l_1[m_1, \ldots, m_k], \ldots, l_n[m_1, \ldots, m_k]]$ and thus t is in the right-hand side of the equation. Now let t be in the right-hand side. Then $t = q[l_1[m_1^1, \ldots, m_k^1], \ldots, l_n[m_1^n, \ldots, m_k^n]]$ for $q \in Q$, $l_i \in L_i$, and $m_j^i \in M_j$. Define $m_j \in M_j$ as follows: if M_j is a singleton then m_j is its element, else if x_j occurs in l_i (for some i) then $m_j = m_j^i$, else m_j is taken to be an arbitrary element of M_j, say m_j^1. From the hypothesis in the lemma one can see that then $t = q[l_1[m_1, \ldots, m_k], \ldots, l_n[m_1, \ldots, m_k]]$. Hence by associativity of tree substitution it follows that t is in the left-hand side of the equation. ▌

Apart from the "associativity law" discussed above, one can consider the "projection laws." Is it true that $c_{n,k}(\pi_i^n, L_1, \ldots, L_n) = L_i$? In the D-algebra $\mathscr{P}(T_\Sigma(X))_{\text{OI}}$ this law holds by the definition of $\underset{\text{OI}}{\leftarrow}$. In the D-algebra $\mathscr{P}(T_\Sigma(X))_{\text{IO}}$ the law does not hold, for instance, $c_{2,k}(\{x_1\}, L, \phi) = \{x_1\} \underset{\text{IO}}{\leftarrow} (L, \phi) = \phi$ for all L. The law $c_{n,n}(L, \pi_1^n, \ldots, \pi_n^n) = L$ holds in both algebras.

A $\tilde{D}$-algebra (where $\tilde{D} = D - \Sigma'$) is called an *abstract clone* in [6, III.3, Exercise 3] if it satisfies the associativity and projection laws. Thus $\mathscr{P}(T_\Sigma(X))_{\text{OI}}$ is an abstract clone, whereas $\mathscr{P}(T_\Sigma(X))_{\text{IO}}$ is not. In general one can say that the OI-case leads to Δ-continuous (abstract) clones, whereas the IO-case leads to subset algebras of abstract clones, which are not abstract clones themselves.

3. Fixed-Point Characterization of Context-Free Tree Languages

In this section we characterize context-free tree languages as minimal fixed-points of Δ-continuous mappings from tree languages to tree languages. More precisely, we view a context-free tree grammar as a system of equations, where the unknowns (i.e., the nonterminals) range over tree languages with variables. As the basic operation to build up these equations we use either IO or OI tree language substitution. In the former case the solution of the system of equations is shown to be the IO tree language generated by the grammar, whereas in the latter case it is the OI tree language generated by the grammar. Thus the difference between the IO and OI tree languages is characterized as the difference between IO and OI substitution as a basic operation in the context-free system of equations. We note that for OI tree languages the fixed-point characterization of this section can also be found in [23].

Let $G = (\Sigma, \mathscr{F}, P)$ be a context-free tree grammar where $\mathscr{F} = \{F_1, ..., F_q\}$ for some $q \geqslant 1$ and let r_i be the rank of F_i for $1 \leqslant i \leqslant q$. The grammar G will be fixed throughout this section.

With G we associate two mappings $M_{G,\mathrm{IO}}$ and $M_{G,\mathrm{OI}}$ both having as domain and range the set

$$\mathscr{D} = \prod_{i=1}^{q} \mathscr{P}(T_\Sigma(X_{r_i})),$$

where $\prod$ is Cartesian product. Note that $\mathscr{D}$ is a $\sqcup$-complete poset. The ordering is usual set inclusion (componentwise) and the minimal element is $\Omega = (\phi, ..., \phi)$. Now let m stand for IO or OI. For all $k \geqslant 0$ and all σ in $T_{\Sigma \cup \mathscr{F}}(X_k)$ the mapping $M_m(\sigma)$: $\mathscr{D} \to \mathscr{P}(T_\Sigma(X_k))$ is defined recursively as follows:

For $\mathbf{d} = (d_1, ..., d_q) \in \mathscr{D}$,

(i) for $\sigma = x_i$ in X_k, $M_m(\sigma)(\mathbf{d}) = \{x_i\}$;

(ii) for $\sigma = f(\sigma_1 \cdots \sigma_n)$ where $f \in \Sigma_n$ for $n \geqslant 0$

$$M_m(\sigma)(\mathbf{d}) = \{f(x_1 \cdots x_n)\} \xleftarrow[m]{} (M_m(\sigma_1)(\mathbf{d}), ..., M_m(\sigma_n)(\mathbf{d}));$$

(iii) for $\sigma = F_i(\sigma_1 \cdots \sigma_{r_i})$

$$M_m(\sigma)(\mathbf{d}) = d_i \xleftarrow[m]{} (M_m(\sigma_1)(\mathbf{d}), ..., M_m(\sigma_{r_i})(\mathbf{d})).$$

Let $\hat{M}_m$ be the extension of M_m to sets of terms L, i.e., for all $\mathbf{d} \in \mathscr{D}$, $\hat{M}_m(L)(\mathbf{d}) = \bigcup_{\sigma \in L} M_m(\sigma)(\mathbf{d})$.

The mapping from $\mathscr{D}$ to $\mathscr{D}$ associated with G, denoted by $M_{G,m}$, is defined as follows:

$$\text{for} \quad \mathbf{d} \in \mathscr{D}: M_{G,m}(\mathbf{d}) = (\hat{M}_m(\mathrm{rhs}(F_1))(\mathbf{d}), ..., \hat{M}_m(\mathrm{rhs}(F_q))(\mathbf{d})).$$

3.1. Lemma. *$M_{G,m}$ is Δ-continuous.*

Proof. In Section 2.4 it was shown that $\underset{\mathrm{IO}}{\leftarrow}$ is $\sqcup$-continuous and that $\underset{\mathrm{OI}}{\leftarrow}$ is Δ-continuous, and since Δ-continuity is preserved by composition, join and "target tupling," the lemma follows. ▌

The properties of $\mathscr{D}$ and the Δ-continuity of $M_{G,m}$ make it possible to use the fixed-point theorem. We shall denote the minimal fixed-point of $M_{G,m}$ by $| G_m |$.

3.2. LEMMA. $| G_m | = \bigcup_{i=0}^{\infty} M^i_{G,m}(\Omega)$. ▌

The rest of this section is devoted to proving that for any $k \geqslant 0$ and $\sigma \in T_{\Sigma \cup \mathscr{F}}(X_k)$

$$M_m(\sigma)(| G_m |) = L_m(G, \sigma).$$

Recall that $L_m(G, \sigma)$ is the language m-generated from σ. Before we prove this result we state the following useful lemma, which shows the behavior of M_m with respect to tree substitution. The OI-part of the lemma is analogous to [33, Lemma 8.2].

3.3. LEMMA. *For $n, k \geqslant 0$, let $\sigma \in T_{\Sigma \cup \mathscr{F}}(X_n)$ and $\tau_1, \ldots, \tau_n \in T_{\Sigma \cup \mathscr{F}}(X_k)$. Then*

(1) *for all* $\mathbf{d} \in \mathscr{D}$,

$$M_{\mathrm{OI}}(\sigma[\tau_1, \ldots, \tau_n])(\mathbf{d}) = M_{\mathrm{OI}}(\sigma)(\mathbf{d}) \underset{\mathrm{OI}}{\leftarrow} (M_{\mathrm{OI}}(\tau_1)(\mathbf{d}), \ldots, M_{\mathrm{OI}}(\tau_n)(\mathbf{d}));$$

(2) *if for all i, $1 \leqslant i \leqslant n$, x_i occurs exactly once in σ or τ_i is terminal (i.e., $\tau_i \in T_\Sigma(X_k)$), then for all* $\mathbf{d} \in \mathscr{D}$,

$$M_{\mathrm{IO}}(\sigma[\tau_1, \ldots, \tau_n])(\mathbf{d}) = M_{\mathrm{IO}}(\sigma)(\mathbf{d}) \underset{\mathrm{IO}}{\leftarrow} (M_{\mathrm{IO}}(\tau_1)(\mathbf{d}), \ldots, M_{\mathrm{IO}}(\tau_n)(\mathbf{d})).$$

Proof. The proof is by straightforward induction on σ using the associativity results in Section 2.4 (Corollary 2.4.2 and Lemma 2.4.3). Note that in the IO case one uses the fact that if τ is terminal then for all $\mathbf{d} \in \mathscr{D}$, $M_{\mathrm{IO}}(\tau)(\mathbf{d}) = \{\tau\}$. ▌

Now we can prove the fixed-point characterization of context-free tree languages.

3.4. THEOREM. *For all $k \geqslant 0$ and all $\sigma \in T_{\Sigma \cup \mathscr{F}}(X_k)$*

$$L_m(G, \sigma) = M_m(\sigma)(| G_m |).$$

In particular, for $1 \leqslant j \leqslant q$,

$$L_m(G, F_j(x_1 \cdots x_{r_j})) = | G_m |_j .$$

Proof. The proof is in two steps, (a) and (b).

(a) First we show that $L_m(G, \sigma) \subseteq M_m(\sigma)(| G_m |)$. This inclusion can be obtained, by Lemma 3.2, from the following statement: For all $p \geqslant 0$ and for all $t \in T_\Sigma(X_k)$, if $\sigma \underset{m}{\overset{p}{\Rightarrow}} t$ then $t \in M_m(\sigma)(M^p_{G,m}(\Omega))$, where $\underset{m}{\overset{p}{\Rightarrow}}$ means derivation in p steps. We prove this by induction on p.

Basis of induction. If $\sigma \overset{0}{\underset{m}{\Rightarrow}} t \in T_\Sigma(X_k)$ then $\sigma = t$, but since t is terminal $M_m(t)(\Omega) = \{t\}$.

Induction step. Assume that $\sigma \overset{p+1}{\underset{m}{\Rightarrow}} t$, then there exists σ' such that $\sigma \underset{m}{\Rightarrow} \sigma' \overset{p}{\underset{m}{\Rightarrow}} t$. By the induction hypothesis $t \in M_m(\sigma')(M_{G,m}^p(\Omega))$ and we have to prove that $t \in M_m(\sigma)(M_{G,m}^{p+1}(\Omega))$. Therefore it suffices to show that

$$M_m(\sigma')(M_{G,m}^p(\Omega)) \subseteq M_m(\sigma)(M_{G,m}^{p+1}(\Omega)). \tag{*}$$

Assume that the derivation step $\sigma \underset{m}{\Rightarrow} \sigma'$ is obtained by application of the production $F_j(x_1 \cdots x_{r_j}) \to \tau$ where $\tau \in T_{\Sigma \cup \mathscr{F}}(X_{r_j})$. Then there exists $\eta \in T_{\Sigma \cup \mathscr{F}}(X_{k+1})$ with exactly one occurrence of x_{k+1} and there exist $\sigma_1, \ldots, \sigma_{r_j}$ in $T_{\Sigma \cup \mathscr{F}}(X_k)$ such that

$$\sigma = \eta[x_1, \ldots, x_k, F_j(\sigma_1 \cdots \sigma_{r_j})]$$

and

$$\sigma' = \eta[x_1, \ldots, x_k, \tau[\sigma_1, \ldots, \sigma_{r_j}]].$$

Now, writing $\overline{M}_m{}^p$ for $M_{G,m}^p(\Omega)$, we use Lemma 3.3 to get

$$\begin{aligned} M_m(\sigma')(\overline{M}_m{}^p) &= M_m(\eta[x_1, \ldots, x_k, \tau[\sigma_1, \ldots, \sigma_{r_j}]])(\overline{M}_m{}^p) \\ &= M_m(\eta)(\overline{M}_m{}^p) \underset{m}{\leftarrow} (x_1, \ldots, x_k, M_m(\tau[\sigma_1, \ldots, \sigma_{r_j}])(\overline{M}_m{}^p)) \end{aligned}$$

and

$$\begin{aligned} M_m(\sigma)(\overline{M}_m^{p+1}) &= M_m(\eta[x_1, \ldots, x_k, F_j(\sigma_1 \cdots \sigma_{r_j})])(\overline{M}_m^{p+1}) \\ &= M_m(\eta)(\overline{M}_m^{p+1}) \underset{m}{\longleftarrow} (x_1, \ldots, x_k, M_m(F_j(\sigma_1 \cdots \sigma_{r_j}))(\overline{M}_m^{p+1})). \end{aligned}$$

Note that for $m = \mathrm{IO}$ we really use that x_{k+1} occurs exactly once in η. Since $\overline{M}_m{}^p \subseteq \overline{M}_m^{p+1}$ the inclusion $(*)$ will follow from proving that

$$M_m(\tau[\sigma_1, \ldots, \sigma_{r_j}])(\overline{M}_m{}^p) \subseteq M_m(F_j(\sigma_1 \cdots \sigma_{r_j}))(\overline{M}_m^{p+1}).$$

Another application of Lemma 3.3 gives

$$M_m(\tau[\sigma_1, \ldots, \sigma_{r_j}])(\overline{M}_m{}^p) = M_m(\tau)(\overline{M}_m{}^p) \underset{m}{\longleftarrow} (M_m(\sigma_1)(\overline{M}_m{}^p), \ldots, M_m(\sigma_{r_j})(\overline{M}_m{}^p))$$

(observe that for $m = \mathrm{IO}$ all σ's are terminal by the definition of an IO derivation). Now by the definition of $M_{G,m}$

$$M_m(\tau)(\overline{M}_m{}^p) \subseteq \hat{M}_m(\mathrm{rhs}(F_j))(\overline{M}_m{}^p) = (M_{G,m}(\overline{M}_m{}^p))_j = (\overline{M}_m^{p+1})_j,$$

so we finally have

$$\begin{aligned} M_m(\tau[\sigma_1, ..., \sigma_{r_j}])(\bar{M}_m{}^p) &\subseteq (\bar{M}_m^{p+1})_j \underset{m}{\longleftarrow} (M_m(\sigma_1)(\bar{M}_m{}^p), ..., M_m(\sigma_{r_j})(\bar{M}_m{}^p)) \\ &\subseteq (\bar{M}_m^{p+1})_j \underset{m}{\longleftarrow} (M_m(\sigma_1)(\bar{M}_m^{p+1}), ..., M_m(\sigma_{r_j})(\bar{M}_m{}^p)) \\ &= M_m(F_j(\sigma_1 \cdots \sigma_{r_j}))(\bar{M}_m^{p+1}). \end{aligned}$$

Hence $(*)$ is proved and the induction step is completed.

(b) Second, we show that $M_m(\sigma)(|G_m|) \subseteq L_m(G, \sigma)$. It suffices to prove the following statement by induction on p:

$$\text{for all } \ p \geqslant 0,\ k \geqslant 0, \text{ and } \sigma \in T_{\Sigma \cup \mathscr{F}}(X_k),\ M_m(\sigma)(M_{G,m}^p(\Omega)) \subseteq L_m(G, \sigma).$$

Basis of induction. If σ is terminal then $M_m(\sigma)(\Omega) = \{\sigma\} = L_m(G, \sigma)$ and if σ is not terminal then $M_m(\sigma)(\Omega) = \phi$.

Induction step. Again we shall use $\bar{M}_m{}^p$ as shorthand for $M_{G,m}^p(\Omega)$. We shall prove by induction on σ that $M_m(\sigma)(\bar{M}_m^{p+1}) \subseteq L_m(G, \sigma)$.

(i) $\sigma = x_j \in X_k$.

$$M_m(\sigma)(\bar{M}_m^{p+1}) = \{x_j\} = L_m(G, \sigma).$$

(ii) $\sigma = f(\sigma_1 \cdots \sigma_j)$, $f \in \Sigma_j$ for some $j \geqslant 0$.

If $t \in M_m(\sigma)(\bar{M}_m^{p+1}) = \{f(x_1 \cdots x_j)\} \underset{m}{\longleftarrow} (M_m(\sigma_1)(\bar{M}_m^{p+1}), ..., M_m(\sigma_j)(\bar{M}_m^{p+1}))$, then there exist $t_i \in M_m(\sigma_i)(\bar{M}_m^{p+1})$ for $1 \leqslant i \leqslant j$ such that $t = f(t_1 \cdots t_j)$. By the σ-induction hypothesis $M_m(\sigma_i)(\bar{M}_m^{p+1}) \subseteq L_m(G, \sigma_i)$ for $1 \leqslant i \leqslant j$. Hence $\sigma_i \underset{m}{\overset{*}{\Rightarrow}} t_i$ and so $\sigma = f(\sigma_1 \cdots \sigma_j) \underset{m}{\overset{*}{\Rightarrow}} f(t_1 \cdots t_j) = t$ and t is in $L_m(G, \sigma)$.

(iii) $\sigma = F_j(\sigma_1 \cdots \sigma_{r_j})$, $F_j \in \mathscr{F}$.

Let $t \in M_m(F_j(\sigma_1 \cdots \sigma_{r_j}))(\bar{M}_m^{p+1})$. From the definition of M_m and $M_{G,m}$ it follows that there is $\tau \in \text{rhs}(F_j)$ such that $t \in M_m(\tau)(\bar{M}_m{}^p) \underset{m}{\longleftarrow} (M_m(\sigma_1)(\bar{M}_m^{p+1}), ..., M_m(\sigma_{r_j})(\bar{M}_m^{p+1}))$. From the p-induction hypothesis we have

$$M_m(\tau)(\bar{M}_m{}^p) \subseteq L_m(G, \tau)$$

and from the σ-induction hypothesis

$$M_m(\sigma_i)(\bar{M}_m^{p+1}) \subseteq L_m(G, \sigma_i) \qquad \text{for } \ 1 \leqslant i \leqslant r_j.$$

From the definition of $\underset{m}{\longleftarrow}$ it follows that there exists $u \in L_m(G, \tau)$ such that $t \in \{u\} \underset{m}{\longleftarrow} (L_m(G, \sigma_1), ..., L_m(G, \sigma_{r_j}))$.

Now we consider the two cases separately.

$m = IO$. By the definition of $\underset{\text{IO}}{\longleftarrow}$ (Section 2.1) there exist $t_i \in L_{\text{IO}}(G, \sigma_i)$ for $1 \leqslant i \leqslant r_j$ such that $t = u[t_1, ..., t_{r_j}]$. But then $\sigma = F_j(\sigma_1 \cdots \sigma_{r_j}) \underset{\text{IO}}{\overset{*}{\Rightarrow}} F_j(t_1 \cdots t_{r_j}) \underset{\text{IO}}{\Rightarrow} \tau[t_1, ..., t_{r_j}] \underset{\text{IO}}{\overset{*}{\Rightarrow}} u[t_1, ..., t_{r_j}] = t$.

$m = OI$. We want to show that

$$u[\sigma_1, \ldots, \sigma_{r_j}] \overset{*}{\underset{\text{OI}}{\Rightarrow}} t \tag{+}$$

because then $\sigma = F_j(\sigma_1 \cdots \sigma_{r_j}) \underset{\text{OI}}{\Rightarrow} \tau[\sigma_1, \ldots, \sigma_{r_j}] \overset{*}{\underset{\text{OI}}{\Rightarrow}} u[\sigma_1, \ldots, \sigma_{r_j}] \overset{*}{\underset{\text{OI}}{\Rightarrow}} t$. Following the definition of $\underset{\text{OI}}{\leftarrow}$ we prove (+) by induction on u.

(i) $u = a \in \Sigma_0$.

Then $t = u = a$ and (+) follows.

(ii) $u = x_i \in X_k$.

Then $u[\sigma_1, \ldots, \sigma_{r_j}] = x_i[\sigma_1, \ldots, \sigma_{r_j}] = \sigma_i$ and $\{x_i\} \underset{\text{OI}}{\leftarrow} (L_{\text{OI}}(G, \sigma_1), \ldots, L_{\text{OI}}(G, \sigma_{r_j})) = L_{\text{OI}}(G, \sigma_i)$, hence $\sigma_i \overset{*}{\underset{\text{OI}}{\Rightarrow}} t$.

(iii) $u = f(u_1 \cdots u_n)$, $f \in \Sigma_n$ for some $n \geqslant 1$.

In this case $t = f(s_1 \cdots s_n)$ where $s_i \in \{u_i\} \underset{\text{OI}}{\leftarrow} (L_{\text{OI}}(G, \sigma_1), \ldots, L_{\text{OI}}(G, \sigma_{r_j}))$ and by the u-induction hypothesis $u_i[\sigma_1, \ldots, \sigma_{r_j}] \overset{*}{\underset{\text{OI}}{\Rightarrow}} s_i$ for $1 \leqslant i \leqslant n$. Now $u[\sigma_1, \ldots, \sigma_{r_j}] = f(u_1 \cdots u_n)[\sigma_1, \ldots, \sigma_{r_j}] = f(u_1[\sigma_1, \ldots, \sigma_{r_j}] \cdots u_n[\sigma_1, \ldots, \sigma_{r_j}]) \overset{*}{\underset{\text{OI}}{\Rightarrow}} f(s_1 \cdots s_n) = t$.

This completes the p-induction step and the second inclusion is proved. The exact statement of the theorem is a direct consequence of the two inclusions. ▌

If we consider the first half of the proof of the theorem we notice that we in fact proved that $L_{\text{unr}}(G, \sigma) \subseteq M_{\text{OI}}(\sigma)(|G_{\text{OI}}|)$ (in the OI-case we did not use the restrictions on η and $\sigma_1, \ldots, \sigma_{r_j}$). Since obviously $L_{\text{OI}}(G, \sigma) \subseteq L_{\text{unr}}(G, \sigma)$ the theorem gives an alternative proof of the well-known fact [12] that $L_{\text{unr}}(G, \sigma) = L_{\text{OI}}(G, \sigma)$. Intuitively Lemma 3.3 plays the role of the "parallel derivation lemma" which is the key to the proof in [12].

4. Context-Free Σ-Tree Grammars and Systems of Regular $D(\Sigma)$-Equations

In this section we show that context-free Σ-tree languages can be characterized as solutions of systems of regular $D(\Sigma)$-equations in tree language substitution algebras.

First we define systems of regular equations over many-sorted alphabets and their solutions in Δ-continuous algebras. Then we show that the class of context-free Σ-tree languages is contained in the class of languages obtained as solutions to systems of regular $D(\Sigma)$-equations in $\mathscr{P}(T_\Sigma(X))$. Using a slight generalization of a normal form lemma in [21] we can show that these two classes are in fact equal. From this correspondence and the main result in [21] it follows that in the IO case the class of context-free Σ-tree languages is exactly the class of YIELDs of recognizable $D(\Sigma)$-tree languages.

We now define systems of regular equations over many-sorted alphabets (the reader is referred to Section 2 for notions and definitions).

4.1. Definition. Let Σ be an S-sorted alphabet (possibly infinite) and let $\mathscr{F} = \langle \mathscr{F}_s \rangle_{s \in S}$ be a family of disjoint sets such that $\bigcup_{s \in S} \mathscr{F}_s$, also denoted by $\mathscr{F}$, is finite.

Assume that $\mathscr{F} = \{F_1, ..., F_n\}$, where $F_i \in \mathscr{F}_{s_i}$ for $1 \leqslant i \leqslant n$. The elements of $\mathscr{F}$ are called nonterminals.

A *system of regular Σ-equations* (in $\mathscr{F}$) is a finite set of equations

$$\{F_i = R_i\}_{i=1}^n,$$

where R_i is a finite subset of $T_\Sigma(\mathscr{F})_{s_i}$. ▌

We want to define solutions to systems of regular equations in Δ-continuous Σ-algebras with $\sqcup$-complete carriers and the approach we take is exactly the same as the one leading to $| G_m |$ in Section 3.

Let $E = \{F_i = R_i\}_{i=1}^n$ be a system of regular Σ-equations in $\mathscr{F}$ and let B be a Δ-continuous Σ-algebra with $\sqcup$-complete carriers. Furthermore, let F_i be "of sort s_i" and let $\mathscr{B} = \prod_{i=1}^n B_{s_i}$. With E we associate a mapping M_E: $\mathscr{B} \to \mathscr{B}$ defined in the following way. For $\sigma \in T_\Sigma(\mathscr{F})_s$ we define $M(\sigma)$: $\mathscr{B} \to B_s$ such that for all $\mathbf{b} = (b_1, ..., b_n)$ in $\mathscr{B}$

(i) for $\sigma = a$ in $\Sigma_{\lambda,s}$, $M(\sigma)(\mathbf{b}) = a_B$,

(ii) for $\sigma = F_i$, $M(\sigma)(\mathbf{b}) = b_i$,

(iii) for $\sigma = f(\sigma_1 \cdots \sigma_k)$, $M(\sigma)(\mathbf{b}) = f_B(M(\sigma_1)(\mathbf{b}), ..., M(\sigma_k)(\mathbf{b}))$.

$M(\sigma)$ is extended to sets R by $\hat{M}(R)(\mathbf{b}) = \bigsqcup_{\sigma \in R} M(\sigma)(\mathbf{b})$ and finally M_E is defined such that for all $\mathbf{b} \in \mathscr{B}$

$$M_E(\mathbf{b}) = (\hat{M}(R_1)(\mathbf{b}), ..., \hat{M}(R_n)(\mathbf{b})).$$

It should be clear that $\mathscr{B}$ is $\sqcup$-complete and that M_E is Δ-continuous. Hence the solution of E can be defined as the minimal fixed-point of M_E.

4.2. DEFINITION. Let Σ be an S-sorted alphabet, let E be a system of regular Σ-equations in $\{F_1, ..., F_n\}$, and let B be a Δ-continuous Σ-algebra with $\sqcup$-complete carriers. The *solution of E in B*, denoted by $| EB | = (| EB |_1, ..., | EB |_n)$, is the minimal fixed-point of M_E. ▌

Now the notion of equational element can be defined.

4.3. DEFINITION. Let Σ be an S-sorted alphabet and let B be a Δ-continuous Σ-algebra with $\sqcup$-complete carriers. For $s \in S$ an element $b \in B_s$ is *equational* in B if there exists a system of regular Σ-equations (in say n variables) such that $b = | EB |_i$ for some i with $1 \leqslant i \leqslant n$. ▌

Generalizing the notion of recognizability to the many-sorted case it follows from [21] that if B is the subset algebra of the free Σ-algebra T_Σ then the equational subsets of T_Σ are exactly the recognizable subsets of T_Σ (cf. [17, 37]).

Now we show how to transform a context-free tree grammar into a system of regular equations. First we define a set of mappings $\text{COMB}^\Sigma = \langle \text{COMB}_k{}^\Sigma \rangle_{k \geqslant 0}$, where $\text{COMB}_k{}^\Sigma$ maps a Σ-tree with k variables into a $D(\Sigma)$-tree of sort k (where $D(\Sigma)$ is the derived alphabet of Σ; see Definition 2.2.1).

4.4. Definition. Let Σ be a ranked alphabet. For $k \geq 0$, COMB_k^{Σ}: $T_{\Sigma}(X_k) \to T_{D(\Sigma),k}$ is the mapping defined by

(i) $\mathrm{COMB}_k^{\Sigma}(x_i) = \pi_i^k$,

(ii) for $f \in \Sigma_0$: $\mathrm{COMB}_k^{\Sigma}(f) = c_{0,k}(f')$,

(iii) for $f \in \Sigma_m$ $(m \geq 1)$:

$$\mathrm{COMB}_k^{\Sigma}(f(t_1 \cdots t_m)) = c_{m,k}(f' \, \mathrm{COMB}_k^{\Sigma}(t_1) \cdots \mathrm{COMB}_k^{\Sigma}(t_m)).$$

COMB_k^{Σ} is extended to sets $L \subseteq T_{\Sigma}(X_k)$ by $\mathrm{COMB}_k^{\Sigma}(L) = \{\mathrm{COMB}_k^{\Sigma}(t) \mid t \in L\}$ and COMB^{Σ} is the family of mappings $\langle \mathrm{COMB}_k^{\Sigma} \rangle_{k \geq 0}$ mapping the family $\langle \mathscr{P}(T_{\Sigma}(X_k)) \rangle_{k \geq 0}$ to the family $\langle \mathscr{P}(T_{D(\Sigma)})_k \rangle_{k \geq 0}$. Whenever Σ is understood we write COMB in stead of COMB^{Σ}. ▌

Using COMB we define the system of regular equations G^D associated with a context-free tree grammar G.

4.5. Definition. Let $G = (\Sigma, \mathscr{F}, P)$ be a context-free Σ-tree grammar where $\mathscr{F} = \{F_1, \dots, F_n\}$ and let $\mathscr{F}' = \{F'_1, \dots, F'_n\}$. Then G^D, the system of regular $D(\Sigma)$-equations (in $\mathscr{F}'$) associated with G, is

$$G^D = \{F'_i = \mathrm{COMB}_{k_i}^{\Sigma \cup \mathscr{F}}(\mathrm{rhs}(F_i))\}_{i=1}^{n},$$

where k_i is the rank of F_i for $1 \leq i \leq n$. ▌

Note that, since $T_{D(\Sigma \cup \mathscr{F}),k}$ is the same as $T_{D(\Sigma)}(\mathscr{F}')_k$ where $F'_i \in \mathscr{F}'_{k_i}$ for $1 \leq i \leq n$, G^D is in fact a system of regular $D(\Sigma)$-equations.

4.6. Example. Consider the grammar $G = (\Sigma, \mathscr{F}, P)$ where $\Sigma_0 = \{a, b\}$, $\Sigma_2 = \{f\}$, $\mathscr{F}_0 = \{F_1, F_3\}$, $\mathscr{F}_1 = \{F_2\}$, and P is the set of productions

$$F_1 \to F_2(F_3), \qquad F_2(x_1) \to f(x_1 x_1), \qquad F_3 \to a, \qquad F_3 \to b.$$

Then G^D is the system of regular $D(\Sigma)$-equations

$$F'_1 = \{c_{1,0}(F'_2 c_{0,0}(F'_3))\},$$

$$F'_2 = \{c_{2,1}(f' \pi_1^1 \pi_1^1)\},$$

$$F'_3 = \{c_{0,0}(a'), c_{0,0}(b')\}. \quad ▌$$

Let m stand for IO or OI. Recall that $L_m(G, \sigma)$ is the language m-generated from σ by G, and that $\mathscr{P}(T_{\Sigma}(X))_m$ is the tree language m-substitution algebra.

4.7. Theorem. *Let $G = (\Sigma, \{F_1, \dots, F_n\}, P)$ be a context-free Σ-tree grammar and let k_i be the rank of F_i for $1 \leq i \leq n$. Then, for $1 \leq i \leq n$,*

$$L_m(G, F_i(x_1 \cdots x_{k_i})) = |\, G^D \mathscr{P}(T_{\Sigma}(X))_m \,|_i \,.$$

Proof. It is easy to check that the function M_{GD} (defined with $B = \mathscr{P}(T_\Sigma(X))_m)$ is identical to $M_{G,m}$ of Section 3, so the theorem follows by an application of Theorem 3.4. ▮

Now we want to show that the above theorem holds in the other direction as well. More precisely we want to show that for any system E of regular $D(\Sigma)$-equations we can construct a context-free Σ-tree grammar generating languages, which are equal to the solution of E in the tree language substitution algebras. Since not all systems of regular $D(\Sigma)$-equations "come" from context-free Σ-tree grammars (a $D(\Sigma)$-tree of the form $c_{n,k}(c_{n',k'}(\cdots)\cdots)$ cannot be the COMB-image of any Σ-tree), the first step of the construction is to transform the system E to a system in so called normal form, from which it is easy to obtain one with the property, that it is the image of a context-free tree grammar via COMB.

4.8. LEMMA. *Let Σ be a ranked alphabet and let B be a Δ-continuous $D(\Sigma)$-algebra with $\sqcup$-complete carriers such that for all $k \geqslant 0$ and all $b \in B_k$*

$$c_{k,k}(b, \pi_1^k, \ldots, \pi_k^k) = b. \tag{$*$}$$

To each system E of regular $D(\Sigma)$-equations one can associate a context-free tree grammar G such that $| EB |$ is a subvector of $| G^D B |$.

Proof. By a straightforward generalization of Lemma 3.1 in [21] (cf. [3, Theorem 11]) it follows that there is effectively a (normal form) system of equations E_1 such that $| EB |$ is a subvector of $| E_1 B |$, and such that all inclusions of E_1 (we call $A' \supseteq \tau$ an inclusion iff $\tau \in R_i$ where $A' = R_i$ is an equation) are of one of the forms

$$A' \supseteq c_{n,k}(B'D'_1 \cdots D'_n) \quad \text{for} \quad n, \quad k \geqslant 0, \tag{1}$$

$$A' \supseteq \pi_1^k \quad \text{for} \quad 1 \leqslant i \leqslant k, \tag{2}$$

$$A' \supseteq f' \quad \text{for} \quad f \in \Sigma_k \text{ and } k \geqslant 0, \tag{3}$$

where primed symbols are nonterminals of E_1. Because of assumption ($*$) there is a system E_2, where the inclusions of types (1) and (3) are replaced by

$$A' = c_{n,k}(B' c_{k,k}(D'_1 \pi_1^k \cdots \pi_k^k) \cdots c_{k,k}(D'_n \pi_1^k \cdots \pi_k^k)), \tag{1a}$$

$$A' = c_{k,k}(f' \pi_1^k \cdots \pi_k^k), \tag{3a}$$

such that $| E_1 B | = | E_2 B |$. The desired grammar is $G = (\Sigma, \mathscr{F}, P)$ where $\mathscr{F}$ is the set of nonterminals of E_2 without primes and the set P of productions contains

$$A(x_1 \cdots x_k) \to B(D_1(x_1 \cdots x_k) \cdots D_n(x_1 \cdots x_k)),$$

$$A(x_1 \cdots x_k) \to x_i \, ,$$

$$A(x_1 \cdots x_k) \to f(x_1 \cdots x_k),$$

corresponding to (1a), (2), and (3a), respectively. Since G^D is obviously identical to E_2 the lemma is proved. Note that G is in OI normal form [12]. ▌

Again let m stand for IO or OI. We have the following theorem.

4.9. THEOREM. *Let Σ be a ranked alphabet and let E be a system of regular $D(\Sigma)$-equations. If G is the grammar constructed in Lemma 4.8 with $\mathscr{F} = \{F_1, \ldots, F_n\}$ then each component of $| E\mathscr{P}(T_\Sigma(X))_m |$ is equal to $L_m(G, F_i(x_1 \cdots x_{k_i}))$ for some i with $1 \leqslant i \leqslant n$ where k_i is the rank of F_i.*

Proof. Since both $\mathscr{P}(T_\Sigma(X))_{\mathrm{IO}}$ and $\mathscr{P}(T_\Sigma(X))_{\mathrm{OI}}$ satisfy condition $(*)$ in Lemma 4.8, the result follows from that lemma and the previous theorem (4.7). ▌

Using the notion of a set being equational we can present the result of Theorems 4.7 and 4.9 in the following corollary.

4.10. COROLLARY. *Let L be a Σ-tree language with variables, i.e., $L \subseteq T_\Sigma(X_k)$ for some $k \geqslant 0$. L is equational in the $D(\Sigma)$-algebra $\mathscr{P}(T_\Sigma(X))_m$ if and only if L is an m-tree language with variables.* ▌

The normal form construction presented above is not needed in the OI case, since we could have defined a grammar G' by associating with every $D(\Sigma)$-inclusion $F' \supseteq \tau$ of E the production $F(\cdots) \rightarrow \mathrm{YIELD}(\tau)$. We leave it to the reader to verify that $| E\mathscr{P}(T_\Sigma(X))_{\mathrm{OI}} | = | G'_{\mathrm{OI}} |$. This approach does not work in the IO case, as is shown by the following example.

4.11. EXAMPLE. Let Σ be the ranked alphabet with $\Sigma_0 = \{a, b\}$ and $\Sigma_2 = \{f\}$, and let $\mathscr{F}'_0 = \{F'_1, F'_3\}$. Consider the system of regular $D(\Sigma)$-equations E:

$$F'_1 = \{c_{1,0}(c_{2,1}(f'\pi_1{}^1\pi_1{}^1)F'_3)\},$$
$$F'_3 = \{a', b'\}.$$

The YIELD-transformation of the equations gives the grammar G' with productions

$$F_1 \rightarrow f(F_3F_3), \qquad F_3 \rightarrow a, \qquad F_3 \rightarrow b.$$

Clearly $| E\mathscr{P}(T_\Sigma(X))_{\mathrm{IO}} |_1 = \{f(aa), f(bb)\}$ but $L_{\mathrm{IO}}(G', F_1) = \{f(aa), f(ab), f(ba), f(bb)\}$. Note that E is equivalent to the system E_1:

$$F'_1 = \{c_{1,0}(F'_2F'_3)\},$$
$$F'_2 = \{c_{2,1}(f'\pi_1{}^1\pi_1{}^1)\},$$
$$F'_3 = \{a', b'\},$$

where F'_2 is a new nonterminal of sort 1. E_1 corresponds to the grammar G in Example 4.6 and $L_{\mathrm{IO}}(G, F_1) = \{f(aa), f(bb)\}$. ▌

The last result in this section follows from an application of Theorem 5.5 in [21] generalized to the many-sorted case (see the comment following Definition 4.3). Since $\mathscr{P}(T_\Sigma(X))_{\text{IO}}$ is a subset algebra, the theorem states that the equational subsets of $T_\Sigma(X)$ are the homomorphic images of the recognizable subsets of the free $D(\Sigma)$-algebra $T_{D(\Sigma)}$. Since YIELD is the unique homomorphism from $T_{D(\Sigma)}$ to $DT_\Sigma(X)$, we obtain the following corollary.

4.12. Corollary. *Let L be a Σ-tree language with variables, i.e., $L \subseteq T_\Sigma(X_k)$ for some $k \geqslant 0$. L is an* IO *tree language if and only if L is the* YIELD *of a recognizable $D(\Sigma)$-tree language (of sort k).* ▮

References

1. E. Ashcroft, Z. Manna, and A. Pnueli, Decidable properties of monadic functional schemas, *J. Assoc. Comput. Mach.* **20** (1973), 489–499.
2. B. S. Baker, "Tree Transductions and Families of Tree Languages," Ph. D. Thesis, Harvard University, Report TR-9-73, 1973.
3. H. Bekić, Definable Operations in General Algebras, and the Theory of Automata and Flowcharts," IBM Laboratory, Vienna, 1969.
4. G. Birkhoff and J. D. Lipson, Heterogeneous algebras, *J. Combinatorial Theory* **8** (1970), 115–133.
5. A. Blikle, Equational languages, *Inform. Contr.* **12** (1972), 134–147.
6. P. M. Cohn, "Universal Algebra," Harper & Row, New York, 1965.
7. B. Courcelle, Recursive schemes, algebraic trees and deterministic languages, *in* "15th Annual Symposium on Switching and Automata Theory, 1974," pp. 52–62.
8. P. J. Downey, "Formal Languages and Recursion Schemes," Harvard University, Report TR-16-74, 1974.
9. J. Engelfriet, Bottom-up and top-down tree transformations—a comparison, *Math. Systems Theory* **9** (1975), 198–231.
10. J. Engelfriet, A note on infinite trees, *Inform. Proc. Lett.* **1** (1972), 229–232.
11. J. Engelfriet, "Simple Program Schemes and Formal Languages," Lecture Notes in Computer Science, Vol. 20, Springer–Verlag, Berlin, 1974.
12. M. J. Fischer, "Grammars with Macro-like Productions," Doctoral Dissertation, Harvard University, 1968 (see also 9th SWAT, pp. 131–142).
13. S. J. Garland and D. C. Luckham, Program schemes, recursion schemes and formal languages, *J. Comput. System Sci.* **7** (1973), 119–160.
14. S. Ginsburg and H. G. Rice, Two families of languages related to ALGOL, *J. Assoc. Comput. Mach.* **9** (1962), 350–371.
15. J. A. Goguen and J. W. Thatcher, Initial algebra semantics, *in* "15th Annual Symposium on Switching and Automata Theory, 1974," pp. 63–77.
16. J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright, Initial algebra semantics, *J. Assoc. Comput. Mach.* **24** (1977), 68–95.
17. T. S. E. Maibaum, A generalized approach to formal languages, *J. Comput. System Sci.* **8** (1974), 409–439.
18. Z. Manna "Mathematical Theory of Computation," McGraw–Hill, New York, 1974.
19. Z. Manna, S. Ness, and J. Vuillemin, Inductive methods for proving properties of programs, *Comm. ACM* **16** (1973), 491–502.
20. J. McCarthy, A basis for a mathematical theory of computation, *in* "Computer Programming and Formal Systems" (P. Braffort and D. Hirschberg, Eds.), pp. 33–70, North–Holland, Amsterdam, 1963.

21. J. Mezei and J. B. Wright, Algebraic automata and context-free sets, *Inform. Contr.* **11** (1967), 3–29.
22. M. Nivat, Languages algébriques sur le magma libre et sémantique des schémas de programme, *in* "Automata, Languages and Programming" (M. Nivat, Ed.), pp. 293–307, North-Holland, Amsterdam, 1973.
23. M. Nivat, On the interpretation of recursive program schemes, *Symposia Mat.* **15** (1975), 255–281.
24. W. F. Ogden and W. C. Rounds, Composition of n tree transducers, *in* "Proceedings of the 4th Annual ACM Symposium on Theory of Computing, Denver, Colorado, 1972," pp. 198–206.
25. W. P. de Roever, Recursion and parameter mechanisms: An axiomatic approach, *in* "Automata, Languages and Programming" (J. Loeckx, Ed.), Lecture Notes in Computer Science, Vol. 14, Springer-Verlag, Berlin, 1974.
26. W. P. de Roever, First order reduction of call-by-name to call-by-value, International Symposium on Proving and Improving Programs, Arc-et-Senans, 1975.
27. G. F. Rose, An extension of ALGOL-like languages, *Comm. ACM* **7** (1964), 52–71.
28. B. K. Rosen, Tree-manipulating systems and Church-Rosser theorems, *J. Assoc. Comput. Mach.* **20** (1973), 160–188.
29. W. C. Rounds, Mappings and grammars on trees, *Math. Systems Theory* **4** (1970), 257–287.
30. W. C. Rounds, Tree-oriented proofs of some theorems on context-free and indexed languages, *in* "Proceedings of the Second Annual Symposium on Theory of Computing, Northampton, Mass., 1970," pp. 109–116.
31. D. Scott, Data types as lattices, *SIAM J. Comput.* **5** (1976), 522–587.
32. D. Scott, "Outline of a Mathematical Theory of Computation," Technical Monograph PRG-2, Oxford University, 1970.
33. J. W. Thatcher, Generalized^2 sequential machine maps, *J. Comput. System Sci.* **4** (1970), 339–367.
34. J. W. Thatcher, "Generalized^2 Sequential Machine Maps," IBM Report RC 2466, 1969.
35. J. W. Thatcher, Tree automata: An informal survey, *in* "Currents in the Theory of Computing" (A. V. Aho, Ed.), pp. 143–172, Prentice-Hall, Englewood Cliffs, N. J., 1973.
36. J. W. Thatcher and J. B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic, *Math. Systems Theory* **2** (1968), 57–81.
37. R. Turner, An infinite hierarchy of term languages—an approach to mathematical complexity, *in* "Automata, Languages and Programming" (M. Nivat, Ed.), pp. 593–608, North-Holland, Amsterdam, 1973.
38. J. Vuillemin, "Syntaxe, sémantique et axiomatique d'un langage de programmation," These, Université Paris VI, 1974.
39. E. G. Wagner, Languages for defining sets in arbitrary algebras, *in* "Proceedings of the 12th Annual Symposium on Switching and Automata Theory, 1971," pp. 192–201.
40. M. Wand, A concrete approach to abstract recursive definitions, *in* "Automata, Languages and Programming" (M. Nivat, Ed.), pp. 331–344, North-Holland, Amsterdam, 1972.
41. M. Wand, "Mathematical Foundations of Formal Language Theory," Massachusetts Institute of Technology, Report MAC TR-108, 1973.
42. M. Wand, An algebraic formulation of the Chomsky hierarchy, *in* "Category Theory Applied to Computation and Control," pp. 209–213, Lecture Notes in Computer Science, Vol. 25, Springer-Verlag, Berlin, 1975.
43. G. Boudol, Thèse de 3ème cycle, Paris VII, 1976.
44. W. Damm, Higher type program schemes and their tree languages, 4th Colloquium on Automata, Languages and Programming, Turku, 1977.