IO and OI

by

Joost Engelfriet

&

Erik Meineche Schmidt

DAIMI PB-47

July 1975

Matematisk Institut Aarhus Universitet
DATALOGISK AFDELING
Ny Munkegade - 8000 Aarhus C - Danmark
Tlf. 06-12 83 55

IO and OI

Part 1

by

Joost Engelfriet †

&

Erik Meineche Schmidt

Aarhus University, Aarhus, Denmark

† Present address: Twente University of Technology,
Enschede, Netherlands

Mailing Address: E. Meineche Schmidt
Department of Computer Science
Aarhus University
Ny Munkegade
8000 Aarhus C
Denmark

<u>Abstract.</u>

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) build 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.

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 macro grammars, 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 macro grammars 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 one needs in fact equational semantics for. Firstly, equational semantics facilitates the task of proving correctness of programs or grammars, since it leads to useful and intuitively clear proof rules. Secondly 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. Thirdly 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. By Mezei and Wright [21] and Thatcher and Wright [36] a general theory of equational subsets of an arbitrary algebra was developed (for systems of "regular" equations). It was shown 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 operations). 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 macro grammars, and call by value and call by name (nondeterministic) recursive procedures with parameters. In our opinion deterministic recursive programs with tests fit also 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 macro grammars 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 e. a. [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 basic operation). Unfortunately all results in sections 9-12 of [17] are wrong, apparently because IO and OI are confused. We hope that this paper contains correct versions of Maibaums 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 straight forward generalizations of techniques concerning subset algebras.

The paper is divided into two parts and seven sections. Part 1 contains sections 1-4; Part 2 contains sections 5-7 and a conclusion. To each of the parts the list of references is added. In Part 1 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 1 can be read independently from Part 2. In Part 2 the results of Part 1 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 2.1 only, together with some facts from 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 run 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 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 on 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 m 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 2.2, of a completely continuous algebra in 2.3 and the two different notions of tree language substitution in 2.1 and 2.4.

2.1. Ranked alphabet, tree substitution, context-free tree grammar.

For any set A , $\wp(A)$ 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, $A \subseteq \wp(A)$.

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 $\{0, 1, 2, \dots\}$ of nonnegative integers.

A ranked alphabet (or ranked operator demain) 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 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 $n \geq 1$, if $f \in \Sigma_n$ and $t_1, \ldots, t_n \in T_\Sigma$, then $f(t_1 \ldots t_n) \in T_\Sigma$.

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 $n \geq 1$. Thus the elements of Y are added as constants. We shall only be interested in the case that Y consists of "variables".

Let $X = \{x_1, x_2, x_3, \ldots\}$ be a fixed denumerable set of variables. Let $X_0 = \emptyset$ and, for $k \geq 1$, $X_k = \{x_1, \ldots, x_k\}$ (note that X is not meant to be a ranked alphabet, the elements of X are meant to be constants). For $k \geq 0$, $m \geq 0$, $t \in T_\Sigma(X_k)$ and $t_1, \ldots, t_k \in T_\Sigma(X_m)$, we denote by $t[t_1, \ldots, t_k]$ the result of <u>substituting</u> 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 are more than one possible objects 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 fixpoint 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" substitution respectively. Here we shall call them inside-out (IO) and outside-in (OI) substitution respectively.

(2.1.1) <u>Definition.</u> Let $k \geq 0$, $m \geq 0$, $L \in \mathcal{P}(T_\Sigma(X_k))$ and $L_1, \ldots, L_k \in \mathcal{P}(T_\Sigma(X_m))$.

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

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

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

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

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

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

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

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

□

Substitution will be further treated in 2.4. Here we note the obvious fact that, for trees $t, t_1, \ldots, t_k$, $t \underset{IO}{\leftarrow} (t_1, \ldots, t_k) = t \underset{OI}{\leftarrow} (t_1, \ldots, t_k) = t[t_1, \ldots, t_k]$. We also note that for $k = 0$ $L \underset{IO}{\leftarrow} (L_1, \ldots, L_k) = L \underset{OI}{\leftarrow} (L_1, \ldots, L_k) = L$. For $k = 1$ we shall write $L \underset{IO}{\leftarrow} L_1$ rather than $L \underset{IO}{\leftarrow} (L_1)$, 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 macro grammar in [12]. Note that we do not specify an initial nonterminal.

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

Σ is a finite ranked alphabet of terminals,

$\mathfrak{F}$ 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

$F(x_1 \dots x_k) \to \tau$, where $k \geq 0$, $F \in \mathfrak{F}_k$ and $\tau \in T_{\Sigma \cup \mathfrak{F}}(X_k)$.

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

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

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

$\xi_1, \dots, \xi_k \in T_{\Sigma \cup \mathfrak{F}}(X_n)$ such that

$$\sigma_1 = \eta[x_1, \dots, x_n \, , F(\xi_1 \dots \xi_k)] \text{ and}$$

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

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

The definition of $\sigma_1 \underset{IO}{\Rightarrow} \sigma_2$ is the same as that for $\sigma_1 \underset{unr}{\Rightarrow} \sigma_2$ except that the ξ's are required to be terminal trees $(\xi_1, \dots, \xi_k \in T_\Sigma(X_n))$. The definition of $\sigma_1 \underset{OI}{\Rightarrow} \sigma_2$ is the same as that for $\sigma_1 \underset{unr}{\Rightarrow} \sigma_2$ except that η is required to be such that x_{n+1} does not occur in a subtree of η of the form $G(\tau_1 \dots \tau_m)$, 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 $k \geq 0$ and $\sigma \in T_{\Sigma \cup \mathfrak{F}}(X_k)$ 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 <u>tree language m-generated by G from σ</u> . It is well known from [12], and we shall give an alternative proof in section 3, that $L_{OI}(G, \sigma) = L_{unr}(G, \sigma)$. A tree language L over Σ is called an <u>IO (OI) tree language</u> if there is a context-free tree grammar $G = (\Sigma, \mathfrak{F}, P)$ such that $L = L_{IO}(G, S)$ $(L = L_{OI}(G, S))$ for some $S \in \mathfrak{F}_0$. For $k \geq 1$ (and eventually for $k = 0$)

a tree language $L \subseteq T_\Sigma(X_k)$ is called an <u>IO (OI) tree language with variables</u> if there is a context-free tree grammar $G = (\Sigma, \mathcal{F}, P)$ such that $L = L_{IO}(G, F(x_1 \ldots x_k))$ $(L = L_{OI}(G, F(x_1 \ldots x_k)))$ for some $F \in \mathcal{F}_k$. 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 $\sigma \in T_{\Sigma \cup \mathcal{F}}(X_k)$, $L_{IO}(G, \sigma)$ 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 $\underset{IO}{\Rightarrow}$, we say that G is an <u>IO tree grammar</u>. Similarly, if we intend $\underset{OI}{\Rightarrow}$, we say that G is an <u>OI tree grammar</u>.

2.2. <u>Many-sorted alphabet, derived alphabet, Σ-algebra, yield, derived operation.</u>

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 <u>sorts</u>). An <u>S-sorted alphabet</u> (or <u>many-sorted alphabet</u> or <u>S-sorted operator domain</u>) 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 <u>operator</u> of <u>type</u> $\langle w, s \rangle$, <u>arity</u> w , <u>sort</u> s and <u>rank</u> $\lg(w)$. If $w = \lambda$, then f is also called a <u>constant</u> 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) <u>Definition</u>. Let Σ be a ranked alphabet. The <u>derived $\mathbb{N}$-sorted alphabet of Σ</u>, denoted by $D(\Sigma)$ or simply D whenever Σ is understood, is defined as follows. Let, for each $n \geq 0$, $\Sigma'_n = \{f' \mid f \in \Sigma_n\}$ be a new set of symbols; let for each $n \geq 1$ and each i , $1 \leq i \leq n$, π_i^n be a new symbol (the i^{th} <u>projection symbol</u> of sort n); and let, for each $n \geq 0$ and $k \geq 0$, $c_{n,k}$ be a new symbol (the $(n,k)^{th}$ <u>composition symbol</u>). Then

(i) $D_{\lambda,0} = \Sigma'_0$,

(ii) for $n \geq 1$, $D_{\lambda, n} = \Sigma'_n \cup \{\pi^n_i \mid 1 \leq i \leq n\}$,

(iii) for $n, k \geq 0$, $D_{\underbrace{kk\ldots k}_{n \text{ times}}, k} = \{c_{n,k}\}$

(in particular, $D_{0,k} = \{c_{0,k}\}$) , and

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

$\square$

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

$T_{\Sigma, s}$ is called the set of <u>trees of sort s over Σ</u>. 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 <u>Σ-algebra</u> (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 <u>carrier</u> or <u>domain</u> 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., $f_A : A_{s_1} \times A_{s_2} \times \ldots \times A_{s_n} \to A_s$ where $s_1 s_2 \ldots s_n = w$. 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) <u>Example</u>. 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 $f(x_1 \ldots x_n)$ (for $f \in \Sigma_0$, $f' = f$). For $n \geq 1$ and $1 \leq i \leq n$, $\pi_i^n = x_i$.

For $n, k \geq 0$, $t \in T_\Sigma(X_n)$ 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 $n \geq 1$, $f \in \Sigma_n$ and $w_1, \ldots, w_n \in \Sigma_0^*$, $f_A(w_1, \ldots, w_n) = w_1 \ldots w_n$. 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. $f_A : A_{s_1} \times A_{s_2} \times \ldots \ldots \times A_{s_n} \to \mathcal{P}(A_s)$. 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 <u>sub algebra</u> 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 <u>Σ-homomorphism</u> $h : A \to B$ is a family $\langle h_s \rangle_{s \in S}$ of mappings $h_s : A_s \to B_s$ such that (1) if $f \in \Sigma_{\lambda, s}$, then $h_s(f_A) = f_B$, and (2), if $f \in \Sigma_{s_1 \ldots s_n, s}$ and $a_i \in A_{s_i}$, then $h_s(f_A(a_1, \ldots, a_n)) = f_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$. 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 $f_{T_\Sigma(Y)}(t_1, \ldots, t_n) = f(t_1 \ldots t_n)$. It is well known (see for instance [4]) that $T_\Sigma(Y)$ is the <u>free Σ-algebra</u> 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) <u>Example</u>. 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) <u>Example</u>. 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, $\bigcup\limits_{w,s} \Sigma_{w,s}$ is finite; note that S might be infinite). A tree language $L \subseteq T_{\Sigma,s}$ is <u>recognizable</u> 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 <u>recognizable</u> if there is a finite sub alphabet 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 $\overline{\Sigma}$ such that $\overline{\Sigma}_n = \bigcup_{w,s} \{\Sigma_{w,s} \mid lg(w) = n\}$. Obviously $T_{\Sigma, s} \subseteq T_{\overline{\Sigma}}$, and moreover it can easily be seen from the definition of $T_{\Sigma, s}$ that it is a recognizable subset of $T_{\overline{\Sigma}}$. 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 $T_{\overline{\Sigma}}$. 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.

Let, for simplicity, Σ be a ranked alphabet. Any tree t in $DT_{\Sigma}(X)$ is also called a <u>derived operator</u>. Given a Σ-algebra A and $k \geq 0$, each $t \in T_{\Sigma}(X_k)$ can be interpreted as a function $A^k \to A$, called a <u>derived operation</u>, denoted by t_A or $derop_A(t)$, and defined as follows: for $a_1, \dots, a_k \in A$, $t_A(a_1, \dots, a_k) = \overline{a}(t)$ where $\overline{a} : T_{\Sigma}(X_k) \to A$ is the unique Σ-homomorphism such that $\overline{a}(x_i) = a_i$ for $1 \leq i \leq k$. 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 $1 \leq i \leq k$, and consider $T_{\Sigma}(Y)$ where $Y_s = \{x_i \mid s_i = s\}$. Each tree $t \in T_{\Sigma}(Y)_s$

gives then rise to a derived operation $t_A : A_{s_1} \times \ldots \times A_{s_n} \to A_s$ 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 <u>D-algebra of functions over A</u> , denoted by $\mathcal{F}(A)$, is defined as follows. For $n \geq 0$, $\mathcal{F}(A)_n$ is the set of all total functions $A^n \to A$; for $f \in \Sigma_n$, $f' = f_A$; π_i^n is the i^{th} projection $A^n \to A$; and $c_{n,k}$ is composition of functions: $c_{n,k}(f, f_1, \ldots, f_n) = f \circ (f_1, \ldots, f_n)$, where $(f \circ (f_1, \ldots, f_n))(a_1, \ldots, a_k) = f(f_1(a_1, \ldots, a_k), \ldots \ldots, f_n(a_1, \ldots, a_k))$ (for $n = 0$, $c_{0,k}(f)(a_1, \ldots, a_k) = f$, 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 ([15, Proposition 2.4], [6 , III.3 Exercise 4]) that $(t[t_1, \ldots, t_k])_A = t_A \circ (t_{1A}, \ldots, t_{kA})$. 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 $\mathcal{F}(A)$ (in fact the unique one). By restricting $\mathcal{F}(A)_n$ to derived operations one obtains therefore a sub D-algebra of $\mathcal{F}(A)$: the <u>D-algebra of derived operations over A</u> , denoted by $der\mathcal{F}(A)$. Note that $der\mathcal{F}(A)$ is in fact the D-homomorphic image of T_D ; thus it is the smallest sub algebra of $\mathcal{F}(A)$, 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, Chapter III.3] a sub $\tilde{D}$-algebra of $\mathfrak{T}(A)$ is called a <u>clone</u>, and $\mathrm{der}\mathfrak{T}(A)$ 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 $\mathrm{der}\mathfrak{T}(T_{\Sigma})$ are isomorphic.

2.3. <u>Continuous algebra, subset algebra</u>.

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 <u>directed</u> if any two elements of A_1 have an upper bound in A_1. A is called <u>$\bigsqcup$-complete</u> (<u>Δ-complete, $\sqcup$-complete</u>) 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 $f : A \to B$, then f is called <u>$\bigsqcup$-continuous</u> (<u>Δ-continuous</u>) 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) <u>Definition</u>. 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 <u>Z-continuous Σ-algebra</u> if its carriers are Z-complete and each of its operations is Z-continuous in each of its arguments, i.e. if $f_A : A_{s_1} \times \ldots \times A_{s_n} \to A_s$ and $a_i \in A_{s_i}$ (for $i \neq k$), then the function $\lambda x. f_A(a_1, \ldots, a_{k-1}, x, a_{k+1}, \ldots, a_n) : A_{s_k} \to A_s$ is Z-continuous.

$\square$

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

Note that the notion of Δ-continuous Σ-algebra coincides with the one in [15]. In statements about Δ-continuous Σ-algebras we shall mostly assume that they have $\bigsqcup$-complete carriers (and the particular algebras we shall consider, have $\bigsqcup$-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 $\bigsqcup$-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 $\bigsqcup$-continuous homomorphisms rather than Δ-continuous, $\perp$-preserving and $\sqcup$-preserving ones.

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

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

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

□

It was noticed in [21], in the case that A is an ordinary Σ-algebra, that the operations $f_{\mathcal{P}(A)}$ are "completely distributive" (i.e. $\bigsqcup$-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) <u>Lemma</u>. For each nondeterministic Σ-algebra A, $\mathcal{P}(A)$ is a $\bigsqcup$-continuous Σ-algebra (where $\bigsqcup$ 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 $\bigsqcup$-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 $\bigsqcup$-continuous Σ-algebras. A Σ-homomorphism $h = \langle h_s \rangle_{s \in S}$ is called $\bigsqcup$-continuous if all h_s are $\bigsqcup$-continuous functions.

(2.3.4) <u>Lemma</u>. Let A be a Σ-algebra and B a $\bigsqcup$-continuous Σ-algebra. Let h be a Σ-homomorphism from A to B. Then h is uniquely extendable to a $\bigsqcup$-continuous Σ-homomorphism from $\mathcal{P}(A)$ to B.

<u>Proof</u>. Obviously, the only way to extend h is by defining, for $A_1 \subseteq A_s$ $(s \in S)$, $\bar{h}(A_1) = \bigsqcup\{h(a) \mid a \in A_1\}$. Then, clearly, $\bar{h}$ is $\bigsqcup$-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 $A_1, \ldots, A_n$,

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

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

$$\begin{aligned}&f_B(\bigsqcup\{h(a_1) \mid a_1 \in A_1\}, \ldots, \bigsqcup\{h(a_n) \mid a_n \in A_n\}) \\ &= f_B(\bar{h}(A_1), \ldots, \bar{h}(A_n)) \,.\end{aligned}$$

$\square$

(2.3.5) <u>Theorem</u>. $\mathscr{P}(T_\Sigma)$ is free in the class of $\bigsqcup$-continuous Σ-algebras with $\bigsqcup$-continuous Σ-homomorphisms (i.e., for each $\bigsqcup$-continuous Σ-algebra A there is a unique $\bigsqcup$-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 $\bar{h}_A : \mathcal{P}(T_\Sigma) \to A$. Since the restriction to T_Σ of any Σ-homomorphism $\mathcal{P}(T_\Sigma) \to A$ is a Σ-homomorphism, $\bar{h}_A$ is unique.

□

For simplicity we restrict ourselves now 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 $k \geq 1$, $\mathcal{P}(T_\Sigma(X_k))$ is the free $\sqcup$-continuous Σ-algebra with generators X_k .

Proof. Analogous to [15, Proposition 2.2].

□

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 $k \geq 0$. 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 $\mathrm{derop}_A(L)$, and defined as follows: for $a_1, \ldots, a_k \in A$, $L_A(a_1, \ldots, a_k) = \bar{a}(L)$, where $\bar{a} : \mathcal{P}(T_\Sigma(X_k)) \to A$ is the unique $\bigsqcup$-continuous Σ-homomorphism such that $\bar{a}(x_i) = a_i$.

$\square$

It is obvious that $L_A(a_1, \ldots, a_k) = \bigsqcup_{t \in L} t_A(a_1, \ldots, a_k)$, where t_A is the derived operation of t in the Σ-algebra A as defined in 2. 2.

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

(2. 3. 8) Theorem. For each tree language L and $\bigsqcup$-continuous Σ-algebra A , the derived operation L_A is Δ-continuous.

Proof. Completely analogous to the proof of [16, Proposition 4. 13].

$\square$

2. 4. Substitution, associativity.

We now characterize tree language substitution (defined in 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 $\mathcal{P}(T_\Sigma(X))_{IO}$, is defined to be the subset D-algebra $\mathcal{P}(DT_\Sigma(X))$.

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

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

The <u>tree language OI substitution algebra</u>, denoted by $\mathcal{P}(T_\Sigma(X))_{OI}$, is defined to be the D-algebra such that

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

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

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

(iv) for $n, k \geq 0$, $L \subseteq T_\Sigma(X_n)$ and $L_1, \ldots, L_n \subseteq T_\Sigma(X_k)$, $c_{n,k}(L, L_1, \ldots, L_n) = L_A(L_1, \ldots, L_n)$, where A is the subset Σ-algebra $\mathcal{P}(T_\Sigma(X_k))$ 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 $c_{n,k}(L, L_1, \ldots, L_n) = L \underset{\mathrm{OI}}{\leftarrow} (L_1, \ldots, L_n)$.

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

The D-algebra $\mathcal{P}(T_\Sigma(X))_{\mathrm{OI}}$ is Δ-continuous (Proof: since $c_{n,k}(L, L_1, \ldots, L_n)$ 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 $\bigsqcup$-continuous in its first argument). It is easy to see that $\mathcal{P}(T_\Sigma(X))_{\mathrm{OI}}$ is in general not $\bigsqcup$-continuous (for instance, $f(x_1x_1) \underset{\mathrm{OI}}{\leftarrow} \{g, h\} \neq (f(x_1x_1) \underset{\mathrm{OI}}{\leftarrow} g) \cup (f(x_1x_1) \underset{\mathrm{OI}}{\leftarrow} h)$ and $f(x_1) \underset{\mathrm{OI}}{\leftarrow} (x_1, \phi) \neq \phi$) .

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 $t \in T_\Sigma(X_n)$, $t_1, \ldots, t_n \in T_\Sigma(X_k)$ and $s_1, \ldots, s_k \in T_\Sigma(X_m)$, $(t[t_1, \ldots, t_n])[s_1, \ldots, s_k] = t[t_1[s_1, \ldots, s_k], \ldots, t_n[s_1, \ldots, s_k]]$. This result was a special case of the fact that, for any Σ-algebra A , $(t[t_1, \ldots, t_n])_A = t_A \circ (t_{1A}, \ldots, t_{nA})$. 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 $\bigsqcup$-continuous Σ-algebra. Let $L \subseteq T_\Sigma(X_n)$ and $L_1, \ldots, L_n \subseteq T_\Sigma(X_k)$. Then $(L \underset{\mathrm{OI}}{\leftarrow} (L_1, \ldots, L_n))_A = L_A \circ (L_{1A}, \ldots, L_{nA})$.

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

$\square$

(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 $M_1, \ldots, M_k \subseteq T_\Sigma(X_p)$,
$(Q \underset{\mathrm{OI}}{\leftarrow} (L_1, \ldots, L_n)) \underset{\mathrm{OI}}{\leftarrow} (M_1, \ldots, M_k) = Q \underset{\mathrm{OI}}{\leftarrow} (L_1 \underset{\mathrm{OI}}{\leftarrow} (M_1, \ldots, M_k), \ldots$
$\ldots, L_n \underset{\mathrm{OI}}{\leftarrow} (M_1, \ldots, M_k))$.

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

$\square$

IO tree language substitution is not associative in general. For instance $(f(x_1x_2) \underset{\mathrm{IO}}{\leftarrow} (x_1, x_1)) \underset{\mathrm{IO}}{\leftarrow} \{a, b\} = f(x_1x_1) \underset{\mathrm{IO}}{\leftarrow} \{a, b\} = \{f(aa), f(bb)\}$.
But $f(x_1x_2) \underset{\mathrm{IO}}{\leftarrow} (x_1 \underset{\mathrm{IO}}{\leftarrow} \{a, b\}\ ,\ x_1 \underset{\mathrm{IO}}{\leftarrow} \{a, b\}) = f(x_1x_2) \underset{\mathrm{IO}}{\leftarrow} (\{a, b\}, \{a, b\})$

$= \{f(aa), f(ab), f(ba), f(bb)\}$. Problems arise in $(Q \underset{IO}{\leftarrow} (L_1,\dots,L_n)) \underset{IO}{\leftarrow}$ $(M_1,\dots,M_k)$ 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 $Q \subseteq T_\Sigma(X_n)$, $L_1,\dots,L_n \subseteq T_\Sigma(X_k)$ and $M_1,\dots,M_k \subseteq T_\Sigma(X_p)$. Suppose that for all i , $1 \le i \le k$, x_i occurs at most in the trees of one of the $L_1,\dots,L_n$ or M_i is a singleton.

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

Proof. Let t be in the left hand side of the above equation. Then $t = (q[l_1,\dots,l_n])\ [m_1,\dots,m_k]$ with $q \in Q$, $l_i \in L_i$ and $m_i \in M_i$. Hence, by associativity of tree substitution, $t = q[l_1[m_1,\dots,m_k],\dots,l_n[m_1,\dots,m_k]]$ and thus t is in the right hand side of the equation. Now let t be in the right hand side. Then $t = q[l_1[m_1^1,\dots,m_k^1],\dots,l_n[m_1^n,\dots,m_k^n]]$ 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

$t = q[l_1[m_1,\dots,m_k],\dots,l_n[m_1,\dots,m_k]]$. Hence by associativity of tree substitution it follows that t is in the left hand side of the equation.

$\square$

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 $\mathcal{P}(T_\Sigma(X))_{OI}$ this law holds by the definition of $\overset{\leftarrow}{OI}$. In the D-algebra $\mathcal{P}(T_\Sigma(X))_{IO}$ the law does not hold, for instance $c_{2,k}(\{x_1\}, L, \phi) = \{x_1\} \overset{\leftarrow}{IO} (L, \phi) = \phi$ 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 <u>abstract clone</u> in [6 , III. 3 Exercise 3] if it satisfies the associativity and projection laws. Thus $\mathcal{P}(T_\Sigma(X))_{OI}$ is an abstract clone, whereas $\mathcal{P}(T_\Sigma(X))_{IO}$ 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 language 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 $G = (\Sigma, \mathfrak{F}, P)$ be a context-free tree grammar where $\mathfrak{F} = \{F_1, \ldots, F_q\}$ for some $q \geq 1$ and let r_i be the rank of F_i for $1 \leq i \leq q$. The grammar G will be fixed throughout this section.

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

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

where Π is Cartesian product. Note that $\mathcal{D}$ is a $\bigsqcup$-complete poset.

The ordering is usual set inclusion (componentwise) and the minimal element is $\Omega = (\phi, \ldots, \phi)$. Now let m stand for IO or OI . For all $k \geq 0$ and all σ in $T_{\Sigma \cup \mathcal{F}}(X_k)$ the mapping $M_m(\sigma) : \mathcal{D} \to \mathcal{P}(T_\Sigma(X_k))$ is defined recursively as follows:

for $\underline{d} = (d_1, \ldots, d_q) \in \mathcal{D}$,

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

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

$$M_m(\sigma)(\underline{d}) = \{f(x_1 \ldots x_n)\} \underset{m}{\leftarrow} (M_m(\sigma_1)(\underline{d}), \ldots, M_m(\sigma_n)(\underline{d})) ;$$

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

$$M_m(\sigma)(\underline{d}) = d_i \underset{m}{\leftarrow} (M_m(\sigma_1)(\underline{d}), \ldots, M_m(\sigma_{r_i})(\underline{d})) .$$

Let $\hat{M}_m$ be the extension of M_m to sets of terms L , i.e. for all $\underline{d} \in \mathcal{D}$

$$\hat{M}_m(L)(\underline{d}) = \bigcup_{\sigma \in L} M_m(\sigma)(\underline{d}) .$$

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

for $\underline{d} \in \mathcal{D}$: $M_{G,m}(\underline{d}) = (\hat{M}_m(\text{rhs}(F_1))(\underline{d}), \ldots, \hat{M}_m(\text{rhs}(F_q))(\underline{d}))$.

(3.1) <u>Lemma</u>. $M_{G,m}$ is Δ-continuous.

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

□

The properties of $\mathfrak{D}$ 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. $|G_m| = \bigcup_{i=0}^{\infty} M_{G,m}^{i}(\Omega)$.

□

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

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

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 behaviour of M_m with respect to tree substitution. The OI-part of the lemma is analogous to Lemma 8.2 in [33].

(3.3) Lemma. Let for $n,k \geq 0$, $\sigma \in T_{\Sigma \cup \mathfrak{F}}(X_n)$ and $\tau_1, \ldots$ $\ldots, \tau_n \in T_{\Sigma \cup \mathfrak{F}}(X_k)$.

Then

(1) for all $\underline{d} \in \mathcal{D}$, $M_{OI}(\sigma[\tau_1, \ldots, \tau_n])\,(\underline{d})$

$= M_{OI}(\sigma)\,(\underline{d}) \overset{\leftarrow}{_{OI}} (M_{OI}(\tau_1)\,(\underline{d}), \ldots, M_{OI}(\tau_n)\,(\underline{d}))$;

(2) if for all i , $1 \leq i \leq n$,

x_i occurs exactly once in σ

or τ_i is terminal (i.e. $\tau_i \in T_\Sigma(X_k)$) , then

for all $\underline{d} \in \mathcal{D}$, $M_{IO}(\sigma[\tau_1, \ldots, \tau_n])\,(\underline{d})$

$= M_{IO}(\sigma)\,(\underline{d}) \overset{\leftarrow}{_{IO}} (M_{IO}(\tau_1)\,(\underline{d}), \ldots, M_{IO}(\tau_n)\,(\underline{d}))$.

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 $\underline{d} \in \mathcal{D}$ $M_{IO}(\tau)\,(\underline{d}) = \{\tau\}$.

$\square$

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

(3.4) Theorem. For all $k \geq 0$ and all $\sigma \in T_{\Sigma \cup \mathcal{F}}(X_k)$

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

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

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

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 $p \geq 0$ and for all $t \in T_\Sigma(X_k)$, if $\sigma \overset{p}{\underset{m}{\Rightarrow}} t$ then $t \in M_m(\sigma)(M^p_{G,m}(\Omega))$,

where $\overset{p}{\underset{m}{\Rightarrow}}$ means derivation in p steps. We prove this by induction on p .

Basis of induction.

If $\sigma \overset{0}{\Rightarrow} t \in T_\Sigma(X_k)$ 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 $t \in M_m(\sigma')(M^p_{G,m}(\Omega))$ and we have to prove that $t \in M_m(\sigma)(M^{p+1}_{G,m}(\Omega))$. Therefore it suffices to show that

$$(*) \qquad M_m(\sigma')(M^p_{G,m}(\Omega)) \subseteq M_m(\sigma)(M^{p+1}_{G,m}(\Omega)) .$$

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

$$\sigma = \eta[x_1,\dots,x_k\ ,\ F_j(\sigma_1\dots\sigma_{r_j})]\ \text{and}$$

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

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

$$M_m(\sigma')\ (\overline{M}^p_m) =$$

$$M_m(\eta[x_1,\dots,x_k\ ,\ \tau[\sigma_1,\dots,\sigma_{r_j}]])\ (\overline{M}^p_m) =$$

$$M_m(\eta)\ (\overline{M}^p_m) \underset{m}{\leftarrow} (x_1,\dots,x_k\ ,\ M_m(\tau[\sigma_1,\dots,\sigma_{r_j}])\ (\overline{M}^p_m))$$

and

$$M_m(\sigma)\ (\overline{M}^{p+1}_m) =$$

$$M_m(\eta[x_1,\dots,x_k\ ,\ F_j(\sigma_1\dots\sigma_{r_j})])\ (\overline{M}^{p+1}_m) =$$

$$M_m(\eta)\ (\overline{M}^{p+1}_m) \underset{m}{\leftarrow} (x_1,\dots,x_k\ ,\ M_m(F_j(\sigma_1\dots\sigma_{r_j}))\ (\overline{M}^{p+1}_m))\ .$$

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

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

Another application of Lemma 3.3 gives

$$M_m(\tau[\sigma_1,\dots,\sigma_{r_j}])\ (\overline{M}^p_m) =$$

$$M_m(\tau)\ (\overline{M}^p_m) \underset{m}{\leftarrow} (M_m(\sigma_1)\ (\overline{M}^p_m),\dots,M_m(\sigma_{r_j})\ (\overline{M}^p_m))$$

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

$$M_m(\tau)\ (\overline{M}^p_m) \subseteq \hat{M}_m(\text{rhs}(F_j))\ (\overline{M}^p_m) =$$

$$(M_{G,m}(\overline{M}_m^p))_j = (\overline{M}_m^{p+1})_j ,$$

so we finally have

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

$$(\overline{M}_m{}^{p+1})_j \overset{\leftarrow}{m} (M_m(\sigma_1)(\overline{M}_m^p),\ldots,M_m(\sigma_{r_j})(\overline{M}_m^p)) \subseteq$$

$$(\overline{M}_m{}^{p+1})_j \overset{\leftarrow}{m} (M_m(\sigma_1)(\overline{M}_m^{p+1}),\ldots,M_m(\sigma_{r_j})(\overline{M}_m^p)) =$$

$$M_m(F_j(\sigma_1\ldots\sigma_{r_j}))(\overline{M}_m^{p+1}) .$$

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

(b) Secondly we show that $M_m(\sigma)(|G_m|) \subseteq L_m(G,\sigma)$. It suffices

to prove the following statement by induction on p :

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

<u>Basis of induction</u>.

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

terminal then $M_m(\sigma)(\Omega) = \emptyset$.

<u>Induction step</u>.

Again we shall use $\overline{M}_m^p$ as shorthand for $M_{G,m}^p(\Omega)$. We shall

prove by induction on σ that $M_m(\sigma)(\overline{M}_m^{p+1}) \subseteq L_m(G,\sigma)$.

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

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

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

If $t \in M_m(\sigma)\,(\overline{M}_m^{p+1}) = \{f(x_1 \ldots x_j)\} \underset{m}{\leftarrow} (M_m(\sigma_1)\,(\overline{M}_m^{p+1}), \ldots, M_m(\sigma_j)\,(\overline{M}_m^{p+1}))$,

then there exist $t_i \in M_m(\sigma_i)\,(\overline{M}_m^{p+1})$ for $1 \leq i \leq j$ such that $t = f(t_1 \ldots t_j)$.

By the σ-induction hypothesis $M_m(\sigma_i)\,(\overline{M}_m^{p+1}) \subseteq L_m(G, \sigma_i)$ for $1 \leq i \leq j$.

Hence $\sigma_i \overset{*}{\underset{m}{\Rightarrow}} t_i$ and so $\sigma = f(\sigma_1 \ldots \sigma_j) \overset{*}{\underset{m}{\Rightarrow}} f(t_1 \ldots t_j) = t$ and t is in

$L_m(G, \sigma)$.

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

Let $t \in M_m(F_j(\sigma_1 \ldots \sigma_{r_j}))\,(\overline{M}_m^{p+1})$. From the definition of M_m and $M_{G,m}$

it follows that there is $\tau \in \text{rhs}(F_j)$ such that $t \in M_m(\tau)\,(\overline{M}_m^{p}) \underset{m}{\leftarrow}$

$(M_m(\sigma_1)\,(\overline{M}_m^{p+1}) , \ldots, M_m(\sigma_{r_j})\,(\overline{M}_m^{p+1}))$. From the p-induction hypothesis

we have

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

and from the σ-induction hypothesis

$$M_m(\sigma_i)\,(\overline{M}_m^{p+1}) \subseteq L_m(G, \sigma_i) \quad \text{for} \quad 1 \leq i \leq r_j \ .$$

From the definition of $\underset{m}{\leftarrow}$ it follows that there exists $u \in L_m(G, \tau)$ such

that $t \in \{u\} \underset{m}{\leftarrow} (L_m(G, \sigma_1), \ldots, L_m(G, \sigma_{r_j}))$.

Now we consider the two cases separately.

<u>m = IO.</u>

By the definition of $\underset{IO}{\leftarrow}$ (section 2.1) there exist $t_i \in L_{IO}(G, \sigma_i)$

for $1 \leq i \leq r_j$ such that $t = u[t_1,\ldots,t_{r_j}]$.

But then $\sigma = F_j(\sigma_1 \ldots \sigma_{r_j}) \overset{*}{\underset{IO}{\Rightarrow}} F_j(t_1 \ldots t_{r_j}) \underset{IO}{\Rightarrow} \tau[t_1,\ldots,t_{r_j}] \overset{*}{\underset{IO}{\Rightarrow}}$

$u[t_1,\ldots,t_{r_j}] = t$.

<u>m = OI.</u>

We want to show that

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

because then $\sigma = F_j(\sigma_1 \ldots \sigma_{r_j}) \underset{OI}{\Rightarrow} \tau[\sigma_1,\ldots,\sigma_{r_j}] \overset{*}{\underset{OI}{\Rightarrow}} u[\sigma_1,\ldots,\sigma_{r_j}] \overset{*}{\underset{OI}{\Rightarrow}} t$.

Following the definition of $\underset{OI}{\leftarrow}$ 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 $\{x_i\} \underset{OI}{\leftarrow} (L_{OI}(G,\sigma_1),\ldots,L_{OI}(G,\sigma_{r_j})) = L_{OI}(G,\sigma_i)$,

hence $\sigma_i \overset{*}{\underset{OI}{\Rightarrow}} t$.

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

In this case $t = f(s_1 \ldots s_n)$ where $s_i \in \{u_i\} \underset{OI}{\leftarrow} (L_{OI}(G,\sigma_1)$,

$\ldots, L_{OI}(G,\sigma_{r_j}))$ and by the u-induction hypothesis

$u_i[\sigma_1,\ldots,\sigma_{r_j}] \overset{*}{\underset{OI}{\Rightarrow}} s_i$ for $1 \leq i \leq n$. Now $u[\sigma_1,\ldots,\sigma_{r_j}] =$

$f(u_1 \ldots u_n)[\sigma_1,\ldots,\sigma_{r_j}] = f(u_1[\sigma_1,\ldots,\sigma_{r_j}] \ldots u_n[\sigma_1,\ldots,\sigma_{r_j}])$

$\overset{*}{\underset{OI}{\Rightarrow}} f(s_1 \ldots s_n) = t$.

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 $L_{unr}(G, \sigma) \subseteq M_{OI}(\sigma)$ ($|G_{OI}|$) (in the OI-case we did not use the restrictions on η and $\sigma_1, \ldots, \sigma_{r_j}$) . Since obviously $L_{OI}(G, \sigma) \subseteq L_{unr}(G, \sigma)$ the theorem gives an alternative proof of the well-known fact [12] that $L_{unr}(G, \sigma) = L_{OI}(G, \sigma)$. 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 $\mathcal{P}(T_\Sigma(X))$. 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) <u>Definition</u>. Let Σ be an S-sorted alphabet (possibly infinite) and let $\mathcal{F} = \langle \mathcal{F}_s \rangle_{s \in S}$ be a family of disjoint sets such that $\bigcup_{s \in S} \mathcal{F}_s$, also denoted by $\mathcal{F}$, is finite. Assume that $\mathcal{F} = \{F_1, \ldots, F_n\}$ where $F_i \in \mathcal{F}_{s_i}$ for $1 \leq i \leq n$. The elements of $\mathcal{F}$ are called nonterminals.

A <u>system of regular Σ-equations</u> (in $\mathcal{F}$) is a finite set of equations

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

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

□

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

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

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

(iii) for $\sigma = f(\sigma_1 \ldots \sigma_k)$ $\quad M(\sigma)\ (\underline{b}) = f_B(M(\sigma_1)\ (\underline{b}), \ldots, M(\sigma_k)\ (\underline{b}))$.

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

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

It should be clear that $\mathcal{B}$ is $\bigsqcup$-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, E a system of regular Σ-equations in $\{F_1, \ldots, F_n\}$ and B a Δ-continuous Σ-algebra with $\bigsqcup$-complete carriers. The solution of E in B, denoted by $|EB| = (|EB|_1, \ldots, |EB|_n)$, 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 B a Δ-continuous Σ-algebra with $\bigsqcup$-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 $1 \leq i \leq n$.

□

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] and [37]).

Now we show how to transform a contextfree tree grammar into a system of regular equations. First we define a set of mappings $COMB^\Sigma = \langle COMB_k^\Sigma \rangle_{k\geq 0}$ where $COMB_k^\Sigma$ 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) <u>Definition</u>. Let Σ be a ranked alphabet. For $k \geq 0$ $COMB_k^\Sigma$: $T_\Sigma(X_k) \to T_{D(\Sigma),k}$ is the mapping defined by

(i) $COMB_k^\Sigma(x_i) = \pi_i^k$,

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

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

$$COMB_k^\Sigma(f(t_1 \ldots t_m)) = c_{m,k}(f' \, COMB_k^\Sigma(t_1) \ldots COMB_k^\Sigma(t_n)) \; .$$

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

$\square$

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

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

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

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

$\square$

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

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

$$F_1 \to F_2(F_3) \ , \ F_2(x_1) \to f(x_1 x_1) \ ,$$

$$F_3 \to a \ , \ F_3 \to b \ .$$

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

$$F_1' = \{c_{1,0}(F_2' \, c_{0,0}(F_3'))\} ,$$

$$F_2' = \{c_{2,1}(f' \, \pi_1^1 \, \pi_1^1)\} ,$$

$$F_3' = \{c_{0,0}(a) , c_{0,0}(b)\} .$$

□

Let m stand for IO or OI. Recall that $L_m(G,\sigma)$ is the language m-generated from σ by G , and that $\mathcal{P}(T_\Sigma(X))_m$ 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 $1 \le i \le n$. Then, for $1 \le i \le n$,

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

Proof. It is easy to check that the function M_{G^D} (defined with $B = \mathcal{P}(T_\Sigma(X))_m$) 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 $c_{n,k}(c_{n',k}(\ldots)\ldots)$ 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) <u>Lemma</u>. Let Σ be a ranked alphabet and B a Δ-continuous $D(\Sigma)$-algebra with $\bigsqcup$-complete carriers such that for all $k \geq 0$ and all $b \in B_k$:

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

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 $|G^D B|$.

<u>Proof</u>. By a straightforward generalization of Lemma 3.1 in [21] (cf. Theorem II in [3]) 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

(1) $A' \supseteq c_{n,k}(B' D_1' \ldots D_n')$ for n , $k \geq 0$,

(2) $A' \supseteq \pi_1^k$ for $1 \leq i \leq k$,

(3) $A' \supseteq f'$ for $f \in \Sigma_k$ and $k \geq 0$,

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

(1a) $A' = c_{k,k}(B' c_{k,k}(D_1' \pi_1^k \ldots \pi_k^k) \ldots c_{k,k}(D_n' \pi_1^k \ldots \pi_k^k))$

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

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

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

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

$A(x_1 \ldots x_k) \to f(x_1 \ldots x_k)$

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].

□

Let again m stand for IO or OI. We have the following theorem.

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

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

$\square$

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 $k \geq 0$. L is equational in the $D(\Sigma)$-algebra $\mathcal{P}(T_\Sigma(X))_m$ if and only if L is an m tree language with variables.

$\square$

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 $F(\underline{x}) \to \text{YIELD}(\tau)$.

We leave it to the reader to verify that $|E\mathcal{P}(T_\Sigma(X))_{OI}| = |G'_{OI}|$.

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 $\mathfrak{F}_0' = \{F_1', F_3'\}$. 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

$$F_1 \to f(F_3\ F_3) ,$$

$$F_3 \to a ,\ F_3 \to b .$$

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

$$F_1' = \{c_{1,0}(F_2'\ F_3')\} ,$$

$$F_2' = \{c_{2,1}(f'\pi_1^1\ \pi_1^1)\},$$

$$F_3' = \{a', b'\} ,$$

where F_2' is a new nonterminal of sort 1. E_1 corresponds to the grammar G in Example 4.6 and $L_{IO}(G, F_1) = \{f(aa), f(bb)\}$.

□

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 $\mathcal{P}(T_\Sigma(X))_{IO}$ 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 $k \geq 0$. 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, JACM 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. of Combinatorial Theory 8 (1970), 115-133.

5. A. Blikle, Equational languages, Inf. and Control 12 (1972), 134-147.

6. P.M. Cohn, "Universal algebra", Harper and Row, New York, 1965.

7. B. Courcelle, Recursive schemes, algebraic trees and deterministic languages, 15th Annual Symposium on Switching & Automata Theory, 1974, 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. Syst. Theory 9 (1975), 198-231.

10. J. Engelfriet, A note on infinite trees, Inf. Proc. Letters 1 (1972), 229-232.

11. J. Engelfriet, Simple program schemes and formal languages, Lecture Notes in Computer Science 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, JCSS 7 (1973), 119-160.

14. S. Ginsburg and H. G. Rice, Two families of languages related to ALGOL, JACM 9 (1962), 350-371.

15. J. A. Goguen and J. W. Thatcher, Initial algebra semantics, 15th Annual Symposium on Switching and Automata Theory, 1974, 63-77.

16. J. A. Goguen, J. W. Thatcher, E. G. Wagner and J. B. Wright, Initial algebra semantics, JACM 24 (1977), 68-95.

17. T. S. E. Maibaum, A generalized approach to formal languages, JCSS 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, CACM 16 (1973), 491-502.

20. J. McCarthy, A basis for a mathematical theory of computation, in: Computer Programming and Formal Systems (eds. P. Braffort and D. Hirschberg), North-Holland Publ. Co., Amsterdam, 1963, pp. 33-70.

21. J. Mezei and J. B. Wright, Algebraic automata and context-free sets, Inf. and Control 11 (1967), 3-29.

22. M. Nivat, Langages algébriques sur le magma libre et sémantique des schémas de programme, in: Automata, Languages and Programming (ed. M. Nivat), North-Holland Publ. Co., Amsterdam, 1973, pp. 293-307.

23. M. Nivat, On the interpretation of recursive program schemes, Symposia Matematica , Vol. 15 (1975), 255-281.

24. W. F. Ogden and W. C. Rounds, Composition of n tree transducers, 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 (ed. J. Loeckx), Lecture Notes in Computer Science 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, CACM 7 (1964), 52-71.

28. B.K. Rosen, Tree-manipulating systems and Church-Rosser theorems, JACM 20 (1973), 160-188.

29. W.C. Rounds, Mappings and grammars on trees, Math. Syst. Theory 4 (1970), 257-287.

30. W.C. Rounds, Tree-oriented proofs of some theorems on context-free and indexed languages, Second Annual Symposium on Theory of Computing, Northampton, Mass., 1970, pp. 109-116.

31. D. Scott, Data types as lattices, SIAM J. Comp. 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, JCSS 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 (ed. A. V. Aho), Prentice-Hall, 1973, pp. 143-172.

36. J. W. Thatcher and J. B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic, Math. Syst. Theory 2 (1968), 57-81.

37. R. Turner, An infinite hierarchy of term languages - an approach to mathematical complexity, in: Automata, Languages and Programming (ed. M. Nivat), North-Holland Publ. Co., Amsterdam, 1973, pp. 593-608.

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, 12th Annual Symposium on Switching and Automata Theory, 1971, 192-201.

40. M. Wand, A concrete approach to abstract recursive definitions, in: Automata, Languages and Programming (ed. M. Nivat), North-Holland Publ. Co., Amsterdam, 1972, pp. 331-344.

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, Lecture Notes in Computer Science 25, Springer-Verlag, Berlin, 1975, pp. 209-213.

43. G. Boudol, Thèse de 3ème cycle, Paris VII.

44. W. Damm, Higher type program schemes and their tree languages, 4th Colloquium on Automata, Languages and Programming, Turku, 1977.

IO and OI

Part 2

by

Joost Engelfriet [†]

&

Erik Meineche Schmidt

Aarhus University, Aarhus, Denmark

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

Mailing Address: E. Meineche Schmidt
Department of Computer Science
Aarhus University
Ny Munkegade
8000 Aarhus C
Denmark

In Part 1 of this paper (this Journal, ...) we presented a fixed point characterization of the (IO and OI) context-free tree languages. We showed that a context-free tree grammer can be viewed as a system of regular equations over a tree language substitution algebra. In this part we shall use these results to obtain a theory of systems of context-free equations over arbitrary continuous algebras. We refer to the introduction of Part 1 for a description of the contents of this part.

5. Systems of context-free equations over arbitrary Σ-algebras (or: nondeterministic recursive program schemes).

In this section we shall view a context-free tree grammar as a computational device to define subsets and relations over an arbitrary (possibly nondeterministic) Σ-algebra, in other words, as a nondeterministic recursive program scheme with the Σ-algebra playing the role of an interpretation of the program scheme. One can think of these program schemes as similar to the usual ones (see for instance [18, 20, 25, 28]), but without tests and with expressions like τ_1 $\underline{\text{or}}$ τ_2 to be evaluated as τ_1 or as τ_2 nondeterministically. Thus, nondeterministic recursive program schemes compute relations rather than functions. As an example, let $\Sigma_0 = \{a\}$, $\Sigma_1 = \{f\}$ and let G be the context-free tree grammar $(\Sigma, \{S,F\}, P)$ where P consists of the productions $F(x)\to x$, $F(x)\to F(f(x))$ and $S\to F(a)$. Then F can be considered as the recursive program scheme $F(x) = x$ $\underline{\text{or}}$ $F(f(x))$, and S as a call of F with some input a . In the Σ-algebra with domain $\mathbb{N}$ and $f_{\mathbb{N}}(x) = x + 1$, F computes the relation $\{(x, y) \mid y \geq x\}$ and S computes a set depending on the input $a_{\mathbb{N}}$.

However, rather than defining the computation of a context-free tree grammar in a Σ-algebra (using a specific computation rule) and characterizing the computed relation as the solution of a system of equations (as we did in section 3), we take the shortcut of just considering the context-free tree grammar as a system of "context-free" equations, leaving it to the reader to be convinced of the computational naturalness of this definition (thus we rely on fixed point semantics for our nondeterministic recursive program schemes).

The aim of this section is to find "Mezei-and-Wright-like" (abbreviated by <u>MW-like</u>) results for context-free tree grammars, i.e., we want to find a "tree algebra", preferably some $\mathcal{P}(T_\Sigma(X))$, such that the solution of a system of context-free equations over any Σ-algebra is the homomorphic image of its solution in that tree algebra. Intuitively such an MW-like result means that, instead of computing in the Σ-algebra, one may as well do the computation "symbolically", i.e. on trees, and afterwards interpret the result in the Σ-algebra. (The "Herbrand theorem" in program scheme theory is a result in this direction).

There are two well known modes of computation for (nondeterministic) recursive programs: call by value and call by name (see [19]). In a <u>call by value</u> computation the actual parameters of a function call have to be values from the domain of computation. Hence, to obtain the relation computed in the call by value mode by a context-free tree grammar in a Σ-algebra, it is natural to consider the grammar as a system of equations to be solved in the algebra of relations over the Σ-algebra, and to use composition of relations as the basic operation in these equations (see [25]). In a call by name computation the actual parameters of a function call are formal expressions which stand for (possibly empty) sets of values from the domain of computation (each element of such a set being a possible value of the actual parameter). Hence, to model the call by name computation of a context-free tree grammar in a Σ-algebra, it seems natural to solve the grammar in the algebra of functions of subsets of the Σ-algebra, and to use composition of these functions as the basic operation (cf. [8]). The relation computed by the grammar is then obtained by restricting the subset-function to singletons. Note that for deterministic program schemes the set of possible values of an actual parameter is always a singleton or empty. In this case it suffices to add an element ω (standing for "undefined", or the empty set) to the domain and to consider functions over the so extended domain (see [18, 28]).

To define the solution of a context-free tree grammar G as a system of context-free equations in an algebra of relations or functions (over some Σ-algebra) we could proceed along the same lines as in section 3 for tree languages, using composition of relations or functions rather than substitution of tree languages. Then, clearly, the solution of G would equal the solution of the corresponding system G^D of regular $D(\Sigma)$-equations (cf. section 4). Therefore we shall just define the proper (Δ-continuous) $D(\Sigma)$-algebras of relations or functions and consider solutions of regular $D(\Sigma)$-equations in these algebras.

(5.1) Definition. A context-free tree grammar G with terminal alphabet Σ will be called a system of context-free Σ-equations. The solution of G in a Δ-continuous $D(\Sigma)$-algebra A (with $\bigsqcup$-complete carriers) is defined to be the solution of the system G^D of regular $D(\Sigma)$-equations in A (see section 4).

$\square$

Now Lemma 4.8 shows that we may even, without increase of generality, consider arbitrary systems of regular $D(\Sigma)$-equations rather than just those obtained from context-free tree grammars (provided (*) of the lemma holds).

Thus, for a given Σ-algebra A and a system of regular $D(\Sigma)$-equations, depending on whether we solve the system in the algebra of relations over A or the algebra of functions over subsets of A, we obtain a call by value solution and a call by name solution over A respectively. Our MW-like results will relate the call by value solution to an IO tree language (in fact, the solution in $\mathcal{P}(T_\Sigma(X))_{IO}$) or a recognizable tree language (the solution in $\mathcal{P}(T_{D(\Sigma)})$), and the call by name solution to an OI tree language (in fact, the solution in $\mathcal{P}(T_\Sigma(X))_{OI}$) . To obtain these results it suffices to show the existence of a ($\sqcup$-continuous) $D(\Sigma)$-homomorphism from the tree language algebra into the algebra of relations or functions, as shown by the following lemma of which the easy proof is left to the reader (cf. Lemma 5.3 of [21]).

(5.2) <u>Lemma</u>. Let Σ be an S-sorted alphabet. Let A and B be Δ-continuous Σ-algebras with $\sqcup$-complete carriers. Let h be a $\sqcup$-continuous Σ-homomorphism $A \to B$. Then h preserves solutions of regular Σ-equations (i.e., if E is a system of regular Σ-equations, then $h(|EA|_i) = |EB|_i$ for all i) .

$\square$

We note here that we will in general be interested in nondeterministic Σ-algebras, i.e. domains of computation in which the basic operations are possibly partial or many-valued.

The rest of this section is organized as follows. First we discuss the IO case. Then the OI case is treated. Finally we look at some connections with the literature, in particular deterministic program schemes with tests (cf. [18]) and the MW-like results for infinite trees in [15] and [23].

Let us start by considering the inside-out or call by value case. Let Σ be a fixed ranked alphabet and D its derived alphabet.

We first define the algebra of relations over a (nondeterministic) Σ-algebra.

(5.3) Definition. Let A be a nondeterministic Σ-algebra. We define $\mathcal{R}(A)$, the D-algebra of relations over A, as follows:

(i) for $n \geq 0$, $\mathcal{R}(A)_n = \mathcal{P}(A^{n+1})$;

(ii) for $n \geq 0$ and $f \in \Sigma_n$, $f' = f_A$ (more precisely

$f' = \{(a_1, \ldots, a_n, a) \mid a \in f_A(a_1, \ldots, a_n)\}$);

(iii) for $n \geq 1$ and $1 \leq i \leq n$,

$\pi_i^n = \{(a_1, \ldots, a_n, a_i) \mid a_1, \ldots, a_n \in A\}$, i.e.

π_i^n is the i^{th} projection $A^n \to A$;

(iv) for $n, k \geq 0$, $R \subseteq A^{n+1}$ and $R_1, \ldots, R_n \subseteq A^{k+1}$,

$c_{n,k}(R, R_1, \ldots, R_n) = R \circ (R_1, \ldots, R_n)$, i.e.

$c_{n,k}$ is composition of relations (in fact, $R \circ (R_1, \ldots, R_n) =$

$\{(a_1, \ldots, a_k, a) \mid$ there are $b_1, \ldots, b_n \in A$ such that

$(a_1, \ldots, a_k, b_i) \in R_i$ and $(b_1, \ldots, b_n, a) \in R\}$; $c_{0,k}(R) =$

$\{(a_1, \ldots, a_k, a) \mid a \in R\}$) .

$\square$

Note that $\mathfrak{R}(A)_0 = \mathcal{P}(A)$.

The easy proof of the following lemma is left to the reader.

(5.4) <u>Lemma</u>. For every nondeterministic Σ-algebra A , $\mathfrak{R}(A)$ is a $\bigsqcup$-continuous D-algebra (with set inclusion as ordering).

$\square$

Observe that $\mathfrak{R}(A)$ is in fact the subset algebra of the nondeterministic D-algebra with A^{n+1} as domain of sort n and the obvious (partial) $c_{n,k}$ operations on tuples.

In accordance with the discussion at the beginning of this section we now define (context-free) equational relations over a Σ-algebra.

(5.5) Definition. Let A be a nondeterministic Σ-algebra. For $n \geq 0$, a relation $R \subseteq A^{n+1}$ is said to be IO equational if it is equational as an element of the $\bigsqcup$-continuous D-algebra $\mathfrak{R}(A)$ (i.e. iff there is a system E of regular D-equations such that $R = |E\mathfrak{R}(A)|_i$ for some i).

$\square$

Note that in particular $(n=0)$ subsets of A may be IO equational.

The following MW-like result is now immediate.

(5.6) Theorem. Let A be a nondeterministic Σ-algebra and E a system of regular D-equations. Then any component of the solution of E in $\mathfrak{R}(A)$ is the D-homomorphic image of the corresponding component of the solution of E in $\mathcal{P}(T_D)$.

Proof. Since there is a (unique) $\bigsqcup$-continuous D-homomorphism from $\mathcal{P}(T_D)$ into $\mathfrak{R}(A)$ (see Theorem 2.3.5), the theorem follows from Lemma 5.2.

$\square$

(5.7) Corollary. Let A be a nondeterministic Σ-algebra. An element of $\mathfrak{R}(A)$ is IO equational iff it is the D-homomorphic image of a recognizable tree language in T_D.

Proof. Immediate from the previous theorem and the fact that in $\mathcal{P}(T_D)$ the equational elements are the recognizable tree languages (see section 4).

□

Given a system of context-free Σ-equations $G = (\Sigma, \mathcal{F}, P)$ and $F \in \mathcal{F}_n$, the component corresponding to F in the solution of G in $\mathcal{R}(A)_n$ might be called the call by value relation computed by (G,F) over A. Thus, by Theorem 5.6, the call by value relation computed by (G,F) is the D-homomorphic image of the recognizable tree language generated by nonterminal F in T_D (viewing G^D as a regular tree grammar in the obvious way). From the computational point of view this means that, instead of computing in A with some input $(a_1, \dots, a_n)$, one can, nondeterministically, generate a tree in T_D, interpret it as a relation and find an element $(a_1, \dots, a_n, a)$ in this relation. Then the element a is one of the possible outputs.

We now ask whether an IO equational relation can be obtained from the context-free tree grammar by first computing formally with Σ-trees, i.e. generating an IO tree language with variables (where intuitively the variables stand for the input values), and then interpreting the tree language with variables as a relation. The obvious way to interpret such a tree language as a relation is defined as follows.

(5.8) <u>Definition</u>. Let A be a nondeterministic Σ-algebra. Let $n \geq 0$ and $L \subseteq T_\Sigma(X_n)$. The <u>derived relation</u> of L over A, denoted by L^A or $\mathrm{derrel}_A(L)$, is the relation in A^{n+1} defined by $L^A = \{(a_1, \ldots, a_n, b) \mid b \in \bar{a}(L)$, where $\bar{a}$ is the unique $\bigsqcup$-continuous Σ-homomorphism $\mathcal{P}(T_\Sigma(X_n)) \to \mathcal{P}(A)$ such that $\bar{a}(x_i) = \{a_i\}\ \}$.

$\square$

It is easy to see that in the case that A is an ordinary (deterministic) Σ-algebra, $L^A = \{(a_1, \ldots, a_n, a) \mid t_A(a_1, \ldots, a_n) = a$ for some t in $L\}$.

We now show that in the case of an ordinary Σ-algebra an MW-like result, as indicated above, can be obtained.

(5.9) <u>Theorem</u>. Let A be a Σ-algebra. The mapping derrel_A is the unique $\bigsqcup$-continuous D-homomorphism $\mathcal{P}(T_\Sigma(X))_{IO} \to \mathcal{R}(A)$.

<u>Proof</u>.

$$\begin{tikzcd}
T_D \arrow[d, "\mathrm{YIELD}"'] & & \\
DT_\Sigma(X) \arrow[d, "\mathrm{derop}_A"'] & \subseteq & \mathcal{P}(T_\Sigma(X))_{IO} \arrow[d, "\mathrm{derrel}_A"] \\
\mathcal{I}(A) & \subseteq & \mathcal{R}(A)
\end{tikzcd}$$

Consider first the mapping $\mathrm{derop}_A : DT_\Sigma(X) \to \mathcal{R}(A)$, where we identify a function $A^n \to A$ with its graph in $\mathcal{P}(A^{n+1})$. This mapping is a D-homomorphism (see the end of section 2.2). Moreover, since the (unique) D-homomorphism $\mathrm{YIELD} : T_D \to DT_\Sigma(X)$ is onto (in fact, for $t \in T_\Sigma(X)$, $t = \mathrm{YIELD}(\mathrm{COMB}(t))$, where $\mathrm{COMB} : T_\Sigma(X) \to T_D$ is defined in Definition 4.4), derop_A is the unique D-homomorphism $DT_\Sigma(X) \to \mathcal{R}(A)$. Hence by (the proof of) Lemma 2.3.4 and the fact that, for $L \subseteq T_\Sigma(X)$, $L^A = \bigcup_{t \in L} t^A$, derrel_A is the unique $\bigsqcup$-continuous D-homomorphism extending derop_A (recall that $\mathcal{P}(T_\Sigma(X))_{IO}$ is the subset algebra of $DT_\Sigma(X)$) . Since the restriction to $DT_\Sigma(X)$ of any D-homomorphism $\mathcal{P}(T_\Sigma(X))_{IO} \to \mathcal{R}(A)$ is a D-homomorphism, derrel_A is unique.

□

From this theorem and Lemma 5.2 we directly obtain the following MW-like result for IO context-free tree grammars.

(5.10) <u>Theorem</u>. Let A be a Σ-algebra and E a system of regular D-equations. Then any component of the solution of E in $\mathcal{R}(A)$ is the derived relation of the corresponding component of the solution of E in $\mathcal{P}(T_\Sigma(X))_{IO}$ (in formula: $|E\mathcal{R}(A)|_i = \mathrm{derrel}_A(|E\mathcal{P}(T_\Sigma(X))_{IO}|_i)$ for all i).

□

Using Corollary 4.10 we have the following corollary.

(5.11) Corollary. Let A be a Σ-algebra. An element of $\mathfrak{R}(A)$ is IO equational if and only if it is the derived relation of an IO tree language with variables.

$\square$

Note in particular that a subset of A is IO equational iff it is the Σ-homomorphic image of an IO tree language over Σ (thus a subset of T_Σ is IO equational iff it is an IO Σ-tree language).

Corollary 5.11 may be stated more precisely as follows. Let $G = (\Sigma, \mathcal{F}, P)$ be a system of context-free Σ-equations and $F \in \mathcal{F}_n$. Then the call by value relation computed by (G, F) over A is the derived relation of the IO tree language with variables $L_{IO}(G, F(x_1 \ldots x_n))$.

Clearly, when solving D-equations in $\mathfrak{R}(A)$, we may restrict attention to derived relations.

(5.12) Definition. Let A be a Σ-algebra. We define the D-algebra of derived relations over A, denoted by $\mathrm{der}\mathfrak{R}(A)$, to be the image of $\mathcal{P}(T_\Sigma(X))_{IO}$ in $\mathfrak{R}(A)$ under the mapping derrel_A.

$\square$

Thus, for $n \geq 0$, $\text{der}\mathfrak{R}(A)_n = \{L^A \mid L \subseteq T_\Sigma(X_n)\}$.

Obviously $\text{der}\mathfrak{R}(A)$ is a $\bigsqcup$-continuous sub D-algebra of $\mathfrak{R}(A)$ (in fact the smallest one), and therefore the solution of any system of regular D-equations in $\mathfrak{R}(A)$ is equal to its solution in $\text{der}\mathfrak{R}(A)$.

As a special case, let us consider the Σ-algebra T_Σ . Clearly the mapping $\text{derrel}_{T_\Sigma} : \mathcal{P}(T_\Sigma(X))_{IO} \rightarrow \text{der}\mathfrak{R}(T_\Sigma)$ is the identity for sort 0 (i.e. for elements of $\mathcal{P}(T_\Sigma)$) . Hence, given an IO tree grammar $G = (\Sigma, \mathcal{F}, P)$ and an $S \in \mathcal{F}_0$, the component corresponding to S is the same in the solutions in $\mathcal{P}(T_\Sigma(X))_{IO}$ and $\text{der}\mathfrak{R}(T_\Sigma)$. Thus we have obtained a fixed point characterization of IO tree languages in the space $\text{der}\mathfrak{R}(T_\Sigma)$. For $F \in \mathcal{F}_n$ $(n \geq 1)$ it follows from previous remarks that its solution in $\text{der}\mathfrak{R}(T_\Sigma)$ is $\text{derrel}_{T_\Sigma}(L_{IO}(G, F(x_1 \dots x_n)))$ which is (using an obvious property of $\overset{*}{\underset{IO}{\Rightarrow}}$) equal to $\{(t_1, \dots, t_n, t) \mid F(t_1 \dots t_n) \overset{*}{\underset{IO}{\Rightarrow}} t\}$.

Note that, in contrast to the case of trees (see the end of section 2.2), the algebras $\mathcal{P}(T_\Sigma(X))_{IO}$ and $\text{der}\mathfrak{R}(T_\Sigma)$ are <u>not</u> isomorphic. As an example, in $\mathcal{P}(T_\Sigma(X_1))$, $\text{derrel}_{T_\Sigma}(T_\Sigma) = \text{derrel}_{T_\Sigma}(T_\Sigma \cup \{x_1\}) = T_\Sigma \times T_\Sigma$.

Consider now the free Σ-algebra $T_\Sigma(X)$ with generators X . Remarks analogous to those for T_Σ also hold for $T_\Sigma(X)$. Moreover the algebras $\mathcal{P}(T_\Sigma(X))_{IO}$ and $\text{der}\mathfrak{R}(T_\Sigma(X))$ <u>are</u> isomorphic. Thus, by

Theorem 5.9, $T_\Sigma(X)$ is a "universal interpretation": two IO tree grammars G_1 and G_2, with nonterminals F_1 and F_2, compute the same call by value relation over all Σ-algebras iff they do so over $T_\Sigma(X)$ (this was brought to the attention of the authors by M. Nivat).

For a nondeterministic Σ-algebra A both Theorems 5.9 and 5.10 break down in general: there is no homomorphism $\mathcal{P}(T_\Sigma(X))_{IO} \to \mathcal{R}(A)$. As a first example, let $f \in \Sigma_2$ and $b \in \Sigma_0$, and suppose that $b_A = \{a_1, a_2\}$ and f_A is a total function $A^2 \to A$. Consider the IO tree grammar G with productions $S \to F(b)$ and $F(x_1) \to f(x_1 x_1)$. Then the Σ-homomorphic image of the language $\{f(bb)\}$ generated by G from S is in general not equal to the solution of S in $\mathcal{R}(A)$. In fact, $\{f(bb)\}^A = \{f_A(a_1, a_1)$, $f_A(a_1, a_2)$, $f_A(a_2, a_1)$, $f_A(a_2, a_2)\}$, but the solution of S in $\mathcal{R}(A)$ is $\{f_A(a_1, a_1)$, $f_A(a_2, a_2)\}$. The reason for this failure is obviously that during call by value computation of $F(b)$ in A we have to fix a value a_1 or a_2 of b before copying it. This process cannot be mirrored in the derivation of the tree grammar. Thus we cannot compute symbolically on trees, but we have to "consult the interpretation" during computation. As a second example, let $d \in \Sigma_0$ and $p \in \Sigma_1$ and suppose that $d_A = \{a\}$ and $p_A : A \to A$ is a partial function such that $p_A(a)$ is undefined. Consider the IO tree grammar with productions $S \to F(d\ p(d))$ and $F(x_1 x_2) \to x_1$. Then the language generated is $\{d\}$, but the solution of S in $\mathfrak{R}(A)$ is $\emptyset$. Again we cannot just compute with trees because we have to test whether $p_A(d_A)$ has a value before deleting the tree $p(d)$. Thus, in a nondeterministic Σ-algebra, the only way to do a symbolic computation seems to be computing with trees in T_D (see Theorem 5.6), which keep all information about the copying and deletion (which have to be done after the symbolic computation).

One should observe that the above two examples are essentially the same as the failure of the associativity law and the projection laws in $\mathcal{P}(T_\Sigma(X))_{IO}$ (see section 2.4). For the first example,

$$(f(x_1x_2) \underset{IO}{\leftarrow} (b,b)) \underset{IO}{\leftarrow} \{a_1,a_2\} \neq f(x_1x_2) \underset{IO}{\leftarrow} (b \underset{IO}{\leftarrow} \{a_1,a_2\} \; , \; b \underset{IO}{\leftarrow} \{a_1,a_2\}),$$

where b is used to denote x_1 . For the second example, $x_1 \underset{IO}{\leftarrow} (d,\emptyset) \neq d$.

Let us now turn to <u>the outside-in or call by name case</u>. Let Σ again be a fixed ranked alphabet and D its derived alphabet. Instead of considering the subset Σ-algebra $\mathcal{P}(A)$ corresponding to some nondeterministic Σ-algebra A , we shall prove our results for the slightly more general case of a $\bigsqcup$-continuous Σ-algebra. For later use the first few definitions will be given for a Δ-continuous Σ-algebra with $\bigsqcup$-complete carrier.

(5.13) <u>Definition.</u> Let B be a Δ-continuous Σ-algebra with $\bigsqcup$-complete carrier. We define $\mathcal{F}_\Delta(B)$, the <u>D-algebra of Δ-continuous functions over B</u>, as follows:

(i) for $n \geq 0$, $\mathcal{F}_\Delta(B)_n$ is the set of all Δ-continuous total functions $B^n \to B$ (for $n = 0$, $\mathcal{F}_\Delta(B)_0 = B$) ;

(ii) for $n \geq 0$ and $f \in \Sigma_n$, $f' = f_B$;

(iii) for $n \geq 1$, $1 \leq i \leq n$ and $b_1, \ldots, b_n \in B$, $\pi_i^n(b_1, \ldots, b_n) = b_i$, i.e. π_i^n is the i^{th} projection $B^n \to B$;

(iv) for $n, k \geq 0$, $f \in \mathcal{F}_\Delta(B)_n$ and $g_1, \ldots, g_n \in \mathcal{F}_\Delta(B)_k$, $c_{n,k}(f, g_1, \ldots, g_n) = f \circ (g_1, \ldots, g_n)$, i.e. $c_{n,k}$ is composition of functions ($c_{0,k}(a)$ is the constant $a : B^k \to B$, i.e. $c_{0,k}(a)(b_1, \ldots, b_k) = a$ for all $b_1, \ldots, b_k \in B$) .

Moreover, each carrier $\mathcal{F}_\Delta(B)_n$ is ordered in the usual way: $f \sqsubseteq g$ if and only if $f(b_1, \ldots, b_n) \sqsubseteq g(b_1, \ldots, b_n)$ for all $b_1, \ldots, b_n \in B$.

$\square$

It is left to the reader to verify the correctness of the definition (the projections are Δ-continuous and composition preserves Δ-continuity). The straightforward proof of the next lemma is also left to the reader.

(5.14) Lemma. For every Δ-continuous Σ-algebra B with $\bigsqcup$-complete carrier, $\mathfrak{F}_\Delta(B)$ is a Δ-continuous D-algebra with $\bigsqcup$-complete carriers.

□

Note that $\mathfrak{F}_\Delta(B)$ is a sub D-algebra of $\mathfrak{F}(B)$, and hence a "sub clone" of $\mathfrak{F}(B)$ (cf. the end of section 2.2). Thus $\mathfrak{F}_\Delta(B)$, and each of its Δ-complete sub D-algebras, might be called a "Δ-continuous clone". Every Δ-continuous clone is then a μ-clone in the sense of [40], but not vice versa. In fact the smallest μ-clone in $\mathfrak{F}_\Delta(B)$, denoted by $\mu\mathrm{Cl}(B)$ in [40], is equal to $\{L_B \mid L$ is a recognizable tree language with variables$\}$ (see Definition 2.3.7).

We now define (context-free) equational functions over a Δ-continuous Σ-algebra (cf. the discussion in the beginning of this section).

(5.15) Definition. Let B be a Δ-continuous Σ-algebra with $\bigsqcup$-complete carrier. For $n \geq 0$, a (Δ-continuous) function $f: B^n \to B$ is said to be <u>OI equational</u> if it is equational as an element of the Δ-continuous D-algebra $\mathcal{F}_\Delta(B)$.

$\square$

Note that in particular $(n = 0)$ elements of B may be OI equational.

We now turn to an MW-like result for $\bigsqcup$-continuous Σ-algebras, in particular subset algebras of nondeterministic Σ-algebras. It will turn out that an OI equational function can be obtained by interpreting an OI tree language with variables as a function. The obvious way to do this is by taking its derived operation in the $\bigsqcup$-continuous Σ-algebra (see Definition 2.3.7).

(5.16) <u>Theorem</u>. Let B be a $\bigsqcup$-continuous Σ-algebra. The mapping derop_B is the unique $\bigsqcup$-continuous D-homomorphism $\mathcal{P}(T_\Sigma(X))_{OI} \to \mathcal{F}_\Delta(B)$.

<u>Proof</u>. Obviously $\text{derop}_B(f(x_1 \dots x_n)) = f_B$ and $\text{derop}_B(x_i)$ is the i^{th} projection. It now follows from Theorem 2.4.1 that derop_B is a D-homomorphism. Moreover derop_B is clearly $\bigsqcup$-continuous. It is easy to see that there can at most be one $\bigsqcup$-continuous D-homomorphism

from $\mathcal{P}(T_\Sigma(X))_{OI}$ into any continuous algebra.

□

From this theorem and Lemma 5.2 we immediately obtain the following MW-like result for systems of OI context-free equations.

(5.17) Theorem. Let B be a $\bigsqcup$-continuous Σ-algebra and E a system of regular D-equations. Then any component of the solution of E in $\mathcal{I}_\Delta(B)$ is the derived operation of the corresponding component of the solution of E in $\mathcal{P}(T_\Sigma(X))_{OI}$ (in formula: $|E\mathcal{I}_\Delta(B)|_i = \mathrm{derop}_B(|E\mathcal{P}(T_\Sigma(X))_{OI}|_i)$ for all i) .

□

Using Corollary 4.10 we have the following corrollary.

(5.18) Corollary. Let B be a $\bigsqcup$-continuous Σ-algebra. An element of $\mathcal{I}_\Delta(B)$ is OI equational iff it is the derived operation of an OI tree language with variables.

□

Note in particular that an element of B is OI equational iff it is the Σ-homomorphic image of an OI tree language over Σ (thus a subset of T_Σ is OI equational iff it is an OI Σ-tree language).

Note also that, analogous to the IO case, one may restrict attention to <u>der$\mathcal{T}_\Delta$(B)</u>, the <u>D-algebra of derived operations over B</u>.

Note finally that in the case of a Δ-continuous Σ-algebra B, both Theorems 5.16 and 5.17 fail to hold in general: there is no homomorphism $\mathcal{P}(T_\Sigma(X))_{OI} \to \mathcal{T}_\Delta(B)$. As an example, let B have domain $\mathcal{P}(A)$ for some set A. Let $a \in \Sigma_0$ and <u>or</u> $\in \Sigma_2$. Let a_B be some nonempty subset of A and let $\underline{or}_B : B^2 \to B$ be union of sets, i.e. $\underline{or}_B(A_1, A_2) = A_1 \cup A_2$. Note that $\underline{or}_B$ is Δ-continuous, but not $\bigsqcup$-continuous in its arguments (it fails on $\bigsqcup \phi$). Consider the OI tree grammar G with productions $S \to F(Q)$, $F(x_1) \to \underline{or}(a\ x_1)$ and $Q \to Q$. Then the solution of S in $\mathcal{P}(T_\Sigma(X))_{OI}$ is ϕ, but its solution in $\mathcal{T}_\Delta(B)$ is a_B.

For a given system of context-free Σ-equations $G = (\Sigma, \mathcal{F}, P)$ with $F \in \mathcal{F}_n$ and a nondeterministic Σ-algebra A, one is often not interested in the solution of F in $\mathcal{T}_\Delta(\mathcal{P}(A))$ as a function $\mathcal{P}(A)^n \to \mathcal{P}(A)$, but in the restriction of this function to singletons.

(5.19) <u>Definition</u>. Let $G = (\Sigma, \mathcal{F}, P)$ be an OI tree grammar, $F \in \mathcal{F}_n$ and A a nondeterministic Σ-algebra. The <u>call by name relation computed by (G, F) over A</u> is defined to be the relation $\{(a_1, \ldots, a_n, a) \mid$

$a \in f(\{a_1\}, \ldots, \{a_n\})\}$, where f is the component corresponding to F in the solution of G in $\mathcal{F}_\Delta(\mathcal{P}(A))$.

□

It follows easily from the definitions of derived operation and derived relation that, for any nondeterministic Σ-algebra A and any $L \subseteq T_\Sigma(X)$, the restriction of $L_{\mathcal{P}(A)}$ to singletons is L^A . Hence, by Theorem 5.17, the call by name relation computed by (G, F) over A equals the derived relation of $L_{OI}(G,\ F(x_1 \ldots x_n))$ over A .

As a special case, let us consider the Σ-algebra T_Σ . Clearly the mapping $\mathrm{derop}_{\mathcal{P}(T_\Sigma)} : \mathcal{P}(T_\Sigma(X))_{OI} \to \mathcal{F}_\Delta(\mathcal{P}(T_\Sigma))$ is the identity for elements of $\mathcal{P}(T_\Sigma)$. Hence, given an OI tree grammar $G = (\Sigma, \mathcal{F}, P)$, the component corresponding to S is the same for the solutions in $\mathcal{P}(T_\Sigma(X))_{OI}$ and $\mathcal{F}_\Delta(\mathcal{P}(T_\Sigma))$. Thus we obtain an alternative fixed point characterization of the OI tree languages, which is due to Downey [8]. For $F \in \mathcal{F}_n$ $(n \geq 1)$ it follows from previous remarks that the call by name relation computed by (G, F) over T_Σ is $\mathrm{derrel}_{T_\Sigma}(L_{OI}(G,\ F(x_1 \ldots x_n)))$ which is (using an obvious property of $\overset{*}{\underset{OI}{\Rightarrow}}$) equal to $\{(t_1, \ldots, t_n, t) \mid F(t_1 \ldots t_n) \overset{*}{\underset{OI}{\Rightarrow}} t\}$. Note that, by the same example as in the IO case, the algebras $\mathcal{P}(T_\Sigma(X))_{OI}$ and $\mathrm{der}\,\mathcal{F}_\Delta(\mathcal{P}(T_\Sigma))$ are not isomorphic. Consider now the

Σ-algebra $T_\Sigma(X)$. Remarks analogous to those for T_Σ also hold for $T_\Sigma(X)$. The algebras $\mathcal{P}(T_\Sigma(X))_{OI}$ and $\mathrm{der}\mathcal{F}_\Delta(\mathcal{P}(T_\Sigma)))$ are isomorphic. Moreover it is easy to see that $T_\Sigma(X)$ is a "universal interpretation": two OI tree grammars G_1 and G_2 , with nonterminals F_1 and F_2 , compute the same call by name relation over all (nondeterministic) Σ-algebras iff they do so over $T_\Sigma(X)$.

In the rest of this section we look at some connections with the literature. We shall first show how ordinary recursive program(scheme)s (see [18, 5.2]) fit into our formalism in both the call by value and the call by name case (the difference being that ordinary recursive program schemes have tests whereas ours have nondeterminism). The main problem is the representation of the if-then-else construction. We shall use the well known "trick" of representing a conditional expression like if p(x) then f(x) else g(x) by the nondeterministic expression

(if p(x) then f(x) else ω) or (if p(x) then ω else g(x)) ,

where ω stands for "undefined". Moreover, to be able to represent the components of this expression, we introduce firstly a basic function pr_2 which stands for the second projection, and secondly, to replace p(x) , two partial functions $p_Y(x)$ and $p_N(x)$, which are defined iff p(x) is true and false respectively, and, if defined, deliver some arbitrary value (for instance x). Our expression is now representable as $pr_2(p_Y(x), f(x))$ $\underline{\text{or}}$ $pr_2(p_N(x), g(x))$. The recursive program $F(x, y) = \underline{\text{if}}\ x = 0\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ F(x - 1, F(x, y))$ will for instance be represented as follows: let $z(x)$ stand for $x = 0$, $f(x)$ for $x - 1$ and a for 1, then the representation is $F(x, y) = pr_2(z_Y(x), a)$ $\underline{\text{or}}$ $pr_2(z_N(x), F(f(x), F(x, y)))$.

We shall now state this more formally. Let a recursive program scheme S consist of a finite ranked alphabet $\mathfrak{F}$ of function symbols, a finite ranked alphabet Σ of operators or basic function symbols, a finite ranked alphabet Ω of predicate symbols and a finite set of equations of the form

(*) $\quad F(x_1 \dots x_n) = \underline{\text{if}}\ p(\tau_1 \dots \tau_k)\ \underline{\text{then}}\ \alpha\ \underline{\text{else}}\ \beta$,

exactly one equation for each $F \in \mathfrak{F}$. In (*), F is in $\mathfrak{F}_n$ $(n \geq 0)$, p in Ω_k $(k \geq 0)$ and $\tau_1, \dots, \tau_k$, α, β are in $T_{\Sigma \cup \mathfrak{F}}(X_n)$. An "interpretation" A consists of a set A, for each $f \in \Sigma_k$ a partial function $f_A : A^k \to A$, and for each $p \in \Omega_k$ a partial function $p_A : A^k \to \{\text{true}, \text{false}\}$. For such an interpretation one can, in the usual way (see [18]), associate with each $F \in \mathfrak{F}_n$ two partial functions $A^n \to A$ which are the functions computed by (S, F) in A in a call by value or call by name mode.

We now associate with each recursive program scheme S a context-free tree grammar G_S. Define new ranked alphabets Ω^Y and Ω^N such that, for each k, $\Omega_k^Y = \{p_Y \mid p \in \Omega_k\}$ and $\Omega_k^N = \{p_N \mid p \in \Omega_k\}$, and let pr_2 be a new symbol of rank 2. G_S is defined to be $(\Sigma \cup \Omega^Y \cup \Omega^N \cup \{pr_2\}, \mathfrak{F}, R)$, where, corresponding to each equation $F(x_1 \dots x_n) = \underline{\text{if}}\ p(\tau_1 \dots \tau_k)\ \underline{\text{then}}\ \alpha\ \underline{\text{else}}\ \beta$, the set R contains the two rules

$$F(x_1 \dots x_n) \to pr_2(p_Y(\tau_1 \dots \tau_k)\alpha) \quad \text{and}$$

$$F(x_1 \dots x_n) \to pr_2(p_N(\tau_1 \dots \tau_k)\beta)\ .$$

Finally we associate with each interpretation A a partial $\Sigma \cup \Omega^Y \cup \Omega^N \cup \{pr_2\}$ - algebra A' with the same carrier, for $f \in \Sigma_k$ $f_{A'} = f_A$, for $p \in \Omega_k$

$$p_{YA'} = \{(a_1, \dots, a_k, a_1) \mid p_A(a_1, \dots, a_k) = \text{true}\} \quad \text{and}$$

$$p_{NA'} = \{(a_1, \dots, a_k, a_1) \mid p_A(a_1, \dots, a_k) = \text{false}\}\ ,$$

and $pr_{2A'} = \{(a_1, a_2, a_2) \mid a_1, a_2 \in A\}$.

We now state without proof the validness of these translations.

For any recursive program scheme S, function symbol F and interpretation A,

(1) the partial function computed by (S, F) in A in a call by value mode is equal to the component of F in the solution of G_S in

$\mathfrak{R}(A')$ (i.e. the call by value relation computed by (G_S, F) over A'; see the comment following Corollary 5.7) ;

(2) the partial function computed by (S,F) in A in a call by name mode is equal to the restriction to singletons of the component of F in the solution of G_S in $\mathfrak{T}_\Delta(\mathfrak{P}(A'))$ (i.e. the call by name relation computed by (G_S, F) over A'; see Definition 5.19).

In [18], to obtain the call by name function computed by a recursive program scheme for an interpretation A, the domain A is extended to $A \cup \{\omega\}$, where ω is a new element standing for "undefined". It is then shown that a recursive program scheme defines a total function $(A \cup \{\omega\})^n \to A \cup \{\omega\}$, which is the least fixed point of a suitable mapping. In our formalism, the call by name relation is obtained via a total function $(\mathfrak{P}(A))^n \to \mathfrak{P}(A)$. Obviously, in $\mathfrak{P}(A)$, the empty set plays the role of ω. We state without proof:

(3) the total function computed by (S,F), in a call by name mode, in $A \cup \{\omega\}$ is equal to the restriction to singletons and $\emptyset$ (where $\emptyset$ stands for ω) of the F-component of the solution of G_S in $\mathfrak{T}_\Delta(\mathfrak{P}(A'))$.

From correspondences (1) and (2), and Theorems 5.6 and 5.17 respectively, we obtain the following MW-like results for recursive program schemes. For any recursive program scheme S, function symbol F and interpretation A,

(1) the call by value function computed by (S,F) in A is the D-homomorphic image in $\mathfrak{R}(A')$ of a recognizable tree language over $D = D(\tilde{\Sigma})$, where $\tilde{\Sigma} = \Sigma \cup \Omega^{Y} \cup \Omega^{N} \cup \{pr_2\}$; (note that we may delete pr_2 from $\tilde{\Sigma}$ since its homomorphic image is equal to that of π_2^2);

(2) the call by name function computed by (S,F) in A is the derived relation over A' of an OI tree language with variables over the alphabet $\Sigma \cup \Omega^{Y} \cup \Omega^{N} \cup \{pr_2\}$ (see the comment following Definition 5.19).

What can we say about recursive program schemes more general than the ones discussed above? Obviously, we can also handle nondeterminism. Consider for instance the following program (taken from [38]): $F(x) = \underline{\text{if}}\ \text{prime}(x)\ \underline{\text{then}}\ (x\ \underline{\text{or}}\ F(x+1))\ \underline{\text{else}}\ F(x+1)$. This nondeterministic program computes (both in the call by value and call by name case) the relation $\{(x,y) \mid y \text{ is prime and } y \geq x\}$ over $\mathbb{N}$. Let $f(x)$ stand for $x+1$ and $m(x)$ for prime(x). Then a context-free tree grammar computing the same relation has rules

$$F(x) \to pr_2(m_Y(x)\ x)\ ,$$

$$F(x) \to pr_2(m_Y(x)\ F(f(x)))\ \text{ and}$$

$$F(x) \to pr_2(m_N(x)\ F(f(x)))$$

or equivalently

$$F(x) \to pr_2(m_Y(x)\ x)\ \text{ and}$$

$$F(x) \to F(f(x))\ .$$

What about the basic operations? In the presence of tests it does not seem to make much sense to make them multivalued. However in the call by name case it makes some sense to consider basic operations $(A \cup \{\omega\})^n \to A \cup \{\omega\}$, or in other words $\mathcal{P}(A)^n \to \mathcal{P}(A)$, which are Δ-continuous but not $\bigsqcup$-continuous in their arguments (i.e. they cannot be obtained from A by the subset construction, or in the words of [18] they are not "natural extensions" of partial functions $A^n \to A$). We have seen previously that our MW-like result breaks down for non-$\bigsqcup$-continous algebras. It is however not clear whether such operations are needed in the presence of nondeterminism. In [18] the main such operation is <u>if</u>-<u>then</u>-<u>else</u>: $(A \cup \{\omega\})^3 \to A \cup \{\omega\}$. We have seen how to handle <u>if</u>-<u>then</u>-

else with the use of nondeterminism. Another example in [18] is the socalled "parallel multiplication" $* : (\mathbb{N} \cup \{\omega\})^2 \to \mathbb{N} \cup \{\omega\}$ where

$$
\begin{aligned}
*(x,y) &= 0 && \text{if } x = 0 \text{ or } y = 0 \\
&= \omega && \text{if } x = \omega \text{ or } y = \omega \text{ (and } x \neq 0,\ y \neq 0) \\
&= xy && \text{otherwise (where } xy \text{ is the ordinary product} \\
& && \text{of natural numbers } x \text{ and } y\text{).}
\end{aligned}
$$

But, using nondeterminism, parallel multiplication can actually be programmed, as follows

$$
\begin{aligned}
*(xy) \quad = \quad & (\underline{\text{if}}\ x = 0\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ xy) \\
\underline{\text{or}}\ & (\underline{\text{if}}\ y = 0\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ xy)\ ,
\end{aligned}
$$

where the call by name solution is of course intended. Thus it seems that, if such functions are needed, they can as well be programmed rather than considered to be basic.

We conclude this section by connecting our MW-like results to MW-like results obtained in the literature in connection with infinite trees (see [15,22, 23]).

In [15] it is shown that a system of regular Σ-equations in which, for each nonterminal, the right hand side of its equation is a singleton (let us call this a "deterministic" system), can be solved in the algebra

CT_Σ of infinite trees over Σ (we shall call its solution a "recognizable infinite tree"). Let $\perp$ be a new symbol of rank 0 . Then CT_Σ can be viewed as consisting of all $\Sigma \cup \{\perp\}$-labeled finite or infinite trees such that a node labeled with a symbol of rank n has exactly n successors. The finite trees in CT_Σ are there fore those of $T_\Sigma(\perp)$. A natural order, with minimal element $\perp$, is defined on CT_Σ , and it is shown that CT_Σ is free in the class of all Δ-continuous Σ-algebras with Δ-continuous $\perp$-preserving Σ-homomorphisms. Moreover, the solution of a deterministic system of regular Σ-equations in any Δ-continuous Σ-algebra is the homomorphic image of its solution in CT_Σ . A deterministic system of context-free Σ-equations (i.e. a context-free tree grammar with one rule for each nonterminal) can be solved in $CT_\Sigma(X)$ (we shall call its solution a "context-free infinite tree"). In fact, $CT_\Sigma(X)$ is a (Δ-continuous) $D(\Sigma)$-algebra in the obvious way (with $c_{n,k}$ being substitution of infinite trees). For any Δ-continuous Σ-algebra B such a system of context-free Σ-equations can be solved in $\mathfrak{F}_\Delta(B)$ and the solution is the derived operation of the solution in $CT_\Sigma(X)$ (where "derived operation" is the unique Δ-continuous $\perp$-preserving $D(\Sigma)$-homomorphism from $CT_\Sigma(X)$ into $\mathfrak{F}_\Delta(B)$). For more details we refer to [15]. In a different setting these MW-like results have been shown by Nivat [23], who represents an infinite tree by a Δ-complete subset of $T_{\Sigma}(\perp)$.

Obviously, these results can also be applied to "nondeterministic" systems of equations by the introduction of an operator + to represent union. For context-free equations there are two (equivalent) ways of doing this, depending on whether + is viewed as an operator on the Σ level or on the $D(\Sigma)$ level. We shall consider the latter alternative, leaving the former to the reader.

Let, for any S-sorted alphabet Ω , Ω^{+} denote the S-sorted alphabet consisting of $\Omega \cup \{+_s \mid s \in S\}$, where $+_s$ is a new symbol of type $\langle ss, s\rangle$. Whenever s is understood we write + rather than $+_s$. With each system E of regular Ω-equations we now associate a deterministic system E^{+} of regular Ω^{+}-equations as follows.
If $F = \{\tau_1, \ldots, \tau_n\}$ is an equation of E (with $n \geq 1$) , then $F = \{\tau_1 + \tau_2 + \ldots + \tau_n\}$ (in some arbitrary order) is the equation for F in E^{+} , where $\tau_1 + \ldots + \tau_n$ of course stands for $+_s(\tau_1 +_s(\tau_2 \ldots +_s(\tau_{n-1} \tau_n) \ldots))$, s being the sort of F . If $F = \emptyset$ is an equation of E , then $F = \{F\}$ is the equation for F in E^{+} . It should be obvious that, for any Δ-continuous Ω-algebra A with $\bigsqcup$-complete carriers, the solutions of

E and E^+ are equal, if, for each s in S, $+_s$ is interpreted as the $\sqcup$-operation in A_s. However, E^+ can also be solved in Δ-continuous Ω^+-algebras in which $+$ is not interpreted as $\sqcup$, for example in CT_{Ω^+}.

Let now Σ be a ranked alphabet and D its derived alphabet. Note that $\Sigma^+ = \Sigma \cup \{+\}$ with $+ \in \Sigma_2^+$ and $D^+ = D \cup \{+_n \mid n \in \mathbb{N}\}$ with $+_n \in D^+_{nn,n}$. For a given system E of regular D-equations, we can solve E^+ in the D^+-algebras $\mathcal{P}(T_\Sigma(X))_{IO}$, $\mathcal{P}(T_\Sigma(X))_{OI}$, $\mathcal{R}(A)$ and $\mathcal{F}_\Delta(B)$ (where A is a nondeterministic Σ-algebra and B a Δ-continuous Σ-algebra with $\bigsqcup$-complete carrier) as before, but also in the Δ-continuous D^+-algebras $CT_{\Sigma^+}(X)$ and CT_{D^+} (where, for $t_1, t_2 \in CT_{\Sigma^+}(X_n)$, $+_n(t_1, t_2) = +(t_1 t_2)$).

We now obtain the following diagram.

$$\begin{tikzcd}
 & & CT_{D^+} \arrow[dll, "(1)"'] \arrow[drr, "(5)"] & & \\
\mathcal{P}(T_D) \arrow[dr, "(2)"] \arrow[ddr, bend right, "(4)"'] & & & & CT_{\Sigma^+}(X) \arrow[dl, "(6)"'] \arrow[ddl, bend left, "(8)"] \\
 & \mathcal{P}(T_\Sigma(X))_{IO} \arrow[d, "(3)"] & & \mathcal{P}(T_\Sigma(X))_{OI} \arrow[d, "(7)"'] & \\
 & \mathcal{R}(A) & & \mathcal{F}_\Delta(B) &
\end{tikzcd}$$

In this diagram, if two algebras P and Q are connected by arrows, it means that there is a unique Δ-continuous $\perp$-preserving (and whenever possible $\bigsqcup$-continuous) D^+-homomorphism from P to Q (details are left to the reader). Furthermore, for any deterministic system of regular D^+-equations, its solution in Q is the homomorphic image of its solution in P . We note that one can easily associate with each deterministic system E_1 of regular D^+-equations a system E of regular D-equations such that the solution of E^+ in any Δ-continuous D^+-algebra is a subvector of the solution of E_1 . Hence, as an example, the IO tree languages over Σ are precisely the images under the homomorphism $(2)\circ(1)$ of the recognizable infinite trees in CT_{D^+} . Thus the diagram surveys all MW-like results for systems of regular D-equations discussed up til now.

Although the diagram is meant to be transitive, (4) and (8) are drawn separately because (3) and (7) do not always exist. In fact, as shown previously, (3) only exists if A is a (deterministic) Σ-algebra, whereas (7) only exists if B is a $\bigsqcup$-continuous Σ-algebra.

Let us recall some of the names of the homomorphisms in the diagram and give names to the new ones. (2) is YIELD, (3) is derrel_A

and (7) is derop_B . For any many-sorted alphabet Ω we shall denote by SET the unique Δ-continuous $\perp$-preserving Ω^+-homomorphism $CT_{\Omega^+} \to \mathcal{P}(T_\Omega)$. Thus (1) in the diagram is SET. It is also appropriate to denote (6) by SET. For obvious reasons we shall denote by YIELD the homomorphism (5): $CT_{D^+} \to CT_{\Sigma^+}(X)$. Note that the homomorphism from CT_{D^+} into $\mathcal{P}(T_\Sigma(X))_{IO}$ is YIELD$\circ$SET , whereas the homomorphism from CT_{D^+} into $\mathcal{P}(T_\Sigma(X))_{OI}$ is SET$\circ$YIELD. This expresses in a nice way the basic difference between IO and OI.

Consider a domain of computation A and let $B = \mathcal{P}(A)$. For nondeterministic recursive program schemes we now indicate the "best MW-like result", i.e. the lowest tree algebra in the diagram in which the computation can be done symbolically. There are 6 possibilities, depending on whether the computation is call by value (IO) or call by name (OI) and depending on whether A is an ordinary Σ-algebra, a nondeterministic Σ-algebra or has even nonnaturally extended basic operations.

IO	Σ-algebra	$\mathcal{P}(T_\Sigma(X))_{IO}$
	nondet. Σ-algebra	$\mathcal{P}(T_D)$
	nonnat. ext.	meaningless

OI	Σ-algebra	$\mathcal{P}(T_\Sigma(X))_{OI}$
	nondet. Σ-algebra	$\mathcal{P}(T_\Sigma(X))_{OI}$
	nonnat. ext.	$CT_{\Sigma^+}(X)$

We finally observe that, instead of $+$, one can also consider a ternary operation _if_-_then_-_else_. This would give a similar diagram for deterministic recursive program schemes.

6. A closure property of the IO tree languages

In section 4 (Corollary 4.12) we have shown that every IO tree language over Σ is the YIELD of a recognizable tree language over $D(\Sigma)$ and vice verca. This result can be used to prove properties of IO tree languages by applying well known facts from the theory of recognizable tree languages, in the same way as was done for context-free string languages in [30,35]. Even, since each IO string language ([12]) is obviously the yield of an IO tree language, one can obtain properties of IO string languages from those of recognizable tree languages (cf. section 4).

To illustrate the fruitfulness of the algebraic fixed point approach to language theory we shall in this section use Corollary 4.12 to give an algebraic proof of the intuitively obvious fact that the IO tree languages are closed under deterministic bottom-up tree transducer mappings. Before doing so we shall look at two special cases: closure under intersection with a recognizable tree language and closure under "tree homomorphisms". At the end of the section 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 respectively.

Let us first recall from section 2.2 that, for any many-sorted alphabet Σ, a tree language over Σ is recognizable iff it is recognizable over a finite subalphabet of Σ. We shall also use, without mentioning, the fact that, if $L \subseteq T_{\Sigma, s}$, then it is recognizable in T_{Σ} iff it is recognizable in $T_{\overline{\Sigma}}$, where $\overline{\Sigma}$ is the ranked alphabet with $\overline{\Sigma}_n = \bigcup_{w,s} \{\Sigma_{w,s} \mid lg(w) = n\}$. We shall refer to $\overline{\Sigma}$ as "Σ, viewed as a ranked alphabet".

In order to obtain closure of the IO tree languages under intersection with a recognizable tree language, we prove the next lemma, which is a straightforward generalization of one of Rounds [30] for regular languages.

(6. 1) <u>Lemma</u>. Let Σ be a finite ranked alphabet. If R is a recognizable tree language over Σ, then, for any finite subalphabet D' of $D(\Sigma)$, $YIELD^{-1}(R) \cap T_{D',0}$ is recognizable.

<u>Proof</u>. Let Q be the finite Σ-algebra such that $R = h_Q^{-1}(F)$ for some $F \subseteq Q$, where h_Q is the Σ-homomorphism $T_{\Sigma} \to Q$. Consider now the finite $D(\Sigma)$-algebra $\mathfrak{F}(Q)$ (this algebra, or rather $der\mathfrak{F}(Q)$, might be called "the algebra of state transition functions of the finite tree automaton Q"). By section 2. 2 the mapping $derop_Q$ is a $D(\Sigma)$-homomorp-

hism $DT_\Sigma(X) \to \mathfrak{F}(Q)$. Therefore $\mathrm{derop}_Q \circ \mathrm{YIELD}$ is the unique $D(\Sigma)$-homomorphism from $T_{D(\Sigma)}$ into $\mathfrak{F}(Q)$. Denote this homomorphism by g . Note that, for a tree t of sort 0 , $\mathrm{derop}_Q(t) = h_Q(t)$. Hence, for sort 0, $g_0^{-1}(F) = \mathrm{YIELD}^{-1}(\mathrm{derop}_Q^{-1}(F)) = \mathrm{YIELD}^{-1}(h_Q^{-1}(F)) = \mathrm{YIELD}^{-1}(R)$. Finally, the restriction of g to $T_{D'}$ is the unique D'-homomorphism from $T_{D'}$ into $\mathfrak{F}(Q)$, and the inverse image of F under this mapping is $\mathrm{YIELD}^{-1}(R) \cap T_{D',0}$. Consequently, this set is recognizable.

□

(6.2) <u>Corollary</u>. The class of IO tree languages is closed under intersection with a recognizable tree language.

<u>Proof</u>. Let L be an IO tree language over Σ and R a recognizable tree language over Σ . By Corollary 4.12, $L = \mathrm{YIELD}(R_0)$ for some recognizable tree language R_0 in $T_{D',0}$, where D' is a finite subalphabet of $D(\Sigma)$. Now $L \cap R = \mathrm{YIELD}(R_0 \cap \mathrm{YIELD}^{-1}(R))$. By the previous lemma $\mathrm{YIELD}^{-1}(R) \cap T_{D',0}$ is recognizable, and so, since the class of recognizable tree languages is closed under intersection, $R_0 \cap \mathrm{YIELD}^{-1}(R)$ is recognizable. Hence, by Corollary 4.12, $\mathrm{YIELD}(R_0 \cap \mathrm{YIELD}^{-1}(R))$ is an IO tree language.

□

Note that it follows easily from this corollary and the lemma in [30 , p. 110] that the class of IO string languages is closed under intersection with a regular string language.

Next we shall show that the class of IO tree languages is closed under "tree homomorphisms". Let Σ and Ω be possibly infinite ranked alphabets. Consider a family $h = \langle h_n \rangle_{n \in \mathbb{N}}$ of mappings $h_n : \Sigma_n \to T_\Omega(X_n)$. Such a family determines a mapping $\bar{h} : T_\Sigma \to T_\Omega$, called a <u>tree homomorphism</u>, by the requirements

(i) for $f \in \Sigma_0$, $\bar{h}(f) = h_0(f)$;

(ii) for $f \in \Sigma_n$, $\bar{h}(f(t_1 \dots t_n)) = h_n(f)[\bar{h}(t_1), \dots, \bar{h}(t_n)]$.

Moreover, together with the requirement that $\bar{h}(x_i) = x_i$ for all i , $\bar{h}$ is a mapping from $T_\Sigma(X_n)$ into $T_\Omega(X_n)$ for each $n \geq 0$. Thus $\bar{h}$ may be viewed as a mapping $DT_\Sigma(X) \to DT_\Omega(X)$. Let $\tilde{D}$ be the $\mathbb{N}$-sorted alphabet $D(\Sigma) - \Sigma'$ (thus $\tilde{D} = D(\Omega) - \Omega'$; $\tilde{D}$ consists of all projection symbols and composition symbols). It can easily be shown, and in fact it is a special case of Lemma 3.3(1), that $\bar{h}$ is a $\tilde{D}$-homomorphism from $DT_\Sigma(X)$ into $DT_\Omega(X)$, both viewed as $\tilde{D}$-algebras.

A tree homomorphism $\bar{h}$ called <u>linear</u> if no $h_n(f)$ contains two occurrences of the same variable. In the next lemma we show that each tree homomorphism from T_Σ into T_Ω can be simulated "on the second level" (i.e. on the level of $T_{D(\Sigma)}$ and $T_{D(\Omega)}$) by a linear tree homomorphism.

(6.3) <u>Lemma</u>. Let Σ and Ω be ranked alphabets, and $\bar{h}$ a tree homomorphism from T_Σ into T_Ω. Then there is a linear tree homomorphism $\bar{g}: T_{\overline{D(\Sigma)}} \to T_{\overline{D(\Omega)}}$ (i.e. $D(\Sigma)$ and $D(\Omega)$ are viewed as ranked alphabets), such that $\text{YIELD} \circ \bar{g} = \bar{h} \circ \text{YIELD}$. Moreover, $\bar{g}$ is sort-preserving (for every t, $\bar{g}(t)$ has the same sort as t).

<u>Proof</u>. Intuitively, in order to simulate $\bar{h}$, $\bar{g}$ only has to change the frontiers of the $D(\Sigma)$-trees. Formally $\bar{g}$ is defined as follows (note that $D(\Sigma)$ and $D(\Omega)$ are viewed as ranked alphabets):

(i) for $n \geq 0$ and $f \in \Sigma_n$, $g_0(f') = \text{COMB}_n^\Omega(h_n(f))$

(for the definition of COMB, see Definition 4.4);

(ii) for $1 \leq i \leq n$, $g_0(\pi_i^n) = \pi_i^n$;

(iii) for $n, k \geq 0$, $g_{n+1}(c_{n,k}) = c_{n,k}(x_1 \ldots x_{n+1})$.

Clearly, $\bar{g}$ may be viewed as a $\tilde{D}$-homomorphism from $T_{D(\Sigma)}$ into $T_{D(\Omega)}$, both considered as $\tilde{D}$-algebras (where $\tilde{D} = D(\Sigma) - \Sigma'$). Now we have the following diagram

$$\begin{tikzcd} T_{D(\Sigma)} \arrow[r, "\overline{g}"] \arrow[d, "\text{YIELD}"'] & T_{D(\Omega)} \arrow[d, "\text{YIELD}"] \\ DT_{\Sigma}(X) \arrow[r, "\overline{h}"] & DT_{\Omega}(X) \end{tikzcd}$$

where all sets are $\tilde{D}$-algebras and all mappings $\tilde{D}$-homorphisms. Since $T_{D(\Sigma)}$ is obviously the free $\tilde{D}$-algebra generated by the elements of Σ', the diagram commutes if it does for the generators. For $n \geq 0$ and $f \in \Sigma_n$,

$\text{YIELD}(\overline{g}(f')) = \text{YIELD}(\text{COMB}_n(h_n(f))) = h_n(f) = \overline{h}(f(x_1 \ldots x_n)) = \overline{h}(\text{YIELD}(f'))$.

Hence $\text{YIELD} \circ \overline{g} = \overline{h} \circ \text{YIELD}$.

□

(6.4) Corollary. The class of IO tree languages is closed under tree homomorphisms.

Proof. Let L be an IO tree language over the finite ranked alphabet Σ . Thus $L = \text{YIELD}(R)$ for some recognizable tree language R over some finite subalphabet of $D(\Sigma)$. By the previous lemma, for any tree homomorphism $\overline{h}: T_\Sigma \to T_\Omega$, $\overline{h}(L) = \overline{h}(\text{YIELD}(R)) = \text{YIELD}(\overline{g}(R))$, where $\overline{g}$ is a linear tree homomorphism $T_{D(\Sigma)} \to T_{D(\Omega)}$, which may be restricted to the above mentioned finite subalphabet of $D(\Sigma)$. Therefore the closure of the recognizable tree languages under linear tree homomorphisms [34] implies that $\bar{g}(R)$ is recognizable over $D(\Omega)$. Hence $\mathrm{YIELD}(\bar{g}(R))$ is an IO tree language.

□

We now turn to the slightly more complicated case of a deterministic bottom-up tree transducer, which may be treated by combining the previous two lemmas. We shall show that such a tree transducer may be simulated "on the second level" by a linear (nondeterministic) bottom-up tree transducer. Using the closure of the recognizable tree languages under linear tree transducers, we obtain the desired closure result as in the two previous cases. For background on bottom-up tree transducers, see [2, 9].

Let Σ and Ω be finite ranked alphabets. A <u>bottom-up tree transducer</u> B <u>from Σ to Ω</u> (called a "bottom-up finite state transformation" in [9]) consists of a finite set Q of "states", a subset F of Q (of "final" states) and a family $\langle B_n \rangle_{n \in \mathbb{N}}$ of mappings $B_n : \Sigma_n \to \mathcal{P}(Q^n \times Q \times T_\Omega(X_n))$, such that, for each $f \in \Sigma_n$, $B_n(f)$ is finite.

Intuitively a tuple $(q_1, \ldots, q_n, q, s)$ in $B_n(f)$ corresponds to a rule $f(q_1(x_1) \ldots q_n(x_n)) \to q(s)$, see [9]. We shall be careless with parentheses, thus $(a,(b,c)) = ((a,b),c) = (a,b,c)$, etc.

B is called <u>deterministic</u> if, for all $n \geq 0$ and $f \in \Sigma_n$, $B_n(f)$ is in fact a mapping $Q^n \rightarrow Q \times T_\Omega(X_n)$ (in particular, $B_0(f)$ is a single element). B is called <u>linear</u> if all trees from $T_\Omega(X)$ used in $\langle B_n \rangle_{n \in \mathbb{N}}$ are linear (i.e. each variable occurs at most once in the tree).

B determines a family $\overline{B} = \langle \overline{B}_n \rangle_{n \in \mathbb{N}}$ of mappings $\overline{B}_n : T_\Sigma(X_n) \rightarrow \mathcal{P}(Q^n \times Q \times T_\Omega(X_n))$ as follows (intuitively, $(q_1, \dots, q_n, q, s) \in \overline{B}_n(t)$ iff $t[q_1(x_1), \dots, q_n(x_n)] \overset{*}{\Rightarrow} q(s)$, i.e., when started on t in state q_i at each occurrence of x_i, B can arrive in state q with output s, see [2, 9]):

(i) for $1 \leq i \leq n$, $\overline{B}_n(x_i) = \{(q_1, \dots, q_n, q_i, x_i) \mid q_1, \dots, q_n \in Q\}$;

(ii) for $f \in \Sigma_0$, $\overline{B}_n(f) = \{(q_1, \dots, q_n, q, s) \mid q_i \in Q$ and $(q, s) \in B_0(f)\}$;

(iii) for $f \in \Sigma_k$ and $t_1, \dots, t_k \in T_\Sigma(X_n)$, $\overline{B}_n(f(t_1 \dots t_k)) =$ $\{(q_1, \dots, q_n, q, s) \mid$ there exist $p_1, \dots, p_k$ in Q and $u_1, \dots, u_k, u$ in $T_\Omega(X_n)$ such that, for $1 \leq i \leq k$, $(q_1, \dots, q_n, p_i, u_i) \in \overline{B}_n(t_i)$, $(p_1, \dots, p_k, q, u) \in B_k(f)$ and $s = u[u_1, \dots, u_k]\}$.

In particular, $\overline{B}_0$ is a mapping from T_Σ into $\mathcal{P}(Q \times T_\Omega)$.

B realizes a mapping from T_Σ into $\mathcal{P}(T_\Omega)$, also denoted by B , defined by $B(t) = \{s \mid (q,s) \in \overline{B}_0(t)$ for some q in $F\}$. Moreover, for $L \subseteq T_\Sigma$, we define $B(L) = \bigcup_{t \in L} B(t)$. Note that, for deterministic transducers, B is a partial function $T_\Sigma \to T_\Omega$.

We now show that every deterministic bottom-up tree transducer can be "lifted to the second level".

(6.5) Lemma. Let Σ and Ω be finite ranked alphabets, and B a deterministic bottom-up tree transducer from Σ into Ω . For every finite subalphabet D' of $D(\Sigma)$ there is a linear bottom-up tree transducer U' from D' into $D(\Omega)$ (both viewed as ranked alphabets) such that $YIELD \circ U' = B \circ YIELD$ as mappings $T_{D',0} \to \mathcal{P}(T_\Omega)$ (in fact they are partial functions $T_{D',0} \to T_\Omega$) .

Proof. We shall construct an infinite linear bottom-up tree transducer U from $D(\Sigma)$ into $D(\Omega)$ such that $YIELD \circ U = B \circ YIELD$ as mappings $T_{D(\Sigma),0} \to \mathcal{P}(T_\Omega)$. U will have an infinite number of states and an infinite number of "input symbols" (the elements of $D(\Sigma)$) , but otherwise all previously given definitions also apply to U . It will be left to the reader to see that, for any finite subalphabet D' of $D(\Sigma)$, U can be restricted in an obvious way to an ordinary bottom-up tree transducer

U′ with the required property.

Intuitively, U will be constructed in such a way that, if YIELD(s) = t , then U simulates on s the behavior of B on t by guessing for each occurrence of an $f' \in \Sigma'_n$ at the frontier of s what the rule applied by B at each of the corresponding occurrences of f in t will be, and then checking the consistency of these guesses (note that, since B is deterministic, it will apply the same rule at two occurrences of f in t which correspond to the same occurrence of f′ in s). The states of U will therefore be the state transitions of B . U changes the $D(\Sigma)$-trees only at their frontier.

Formally U is defined as follows. Let B have states Q , final states F and mappings $\langle B_n \rangle_{n \in \mathbb{N}}$. Then the set of states of U is $Q_U = \bigcup_{n \in \mathbb{N}} (Q^n \times Q)$, and the set of final states of U is F . The mappings $\langle U_n \rangle_{n \in \mathbb{N}}$ are defined by

(i) for $f \in \Sigma_n$ $(n \geq 0)$,

$$U_0(f') = \{((q_1, \ldots, q_n, q), \text{COMB}_n^{\Omega}(t)) \mid (q_1, \ldots, q_n, q, t) \in B_n(f)\} \ ;$$

(ii) for $1 \leq i \leq n$, $U_0(\pi_i^n) = \{((q_1, \ldots, q_n, q_i), \pi_i^n) \mid q_1, \ldots, q_n \in Q\}$;

(iii) for $n, k \geq 0$, $U_{n+1}(c_{n,k}) =$

$$\{((q_1, \ldots, q_k, p_1), \ldots, (q_1, \ldots, q_k, p_n), (p_1, \ldots, p_n, p) ,$$

$(q_1, \ldots, q_k, p), \; c_{n,k}(x_1 \circ \ldots \circ x_{n+1})) \mid q_i, p_j, p \in Q\}$.

Consider the mapping $\overline{U}_0 : T_{D(\Sigma)} \to \mathcal{P}(Q_U \times T_{D(\Omega)})$. It is easy to see that, for each $n \geq 0$, $\overline{U}_0$ maps $T_{D(\Sigma),n}$ into $\mathcal{P}(Q^n \times Q \times T_{D(\Omega),n})$. Thus one can draw the following diagram

$$\begin{tikzcd}
T_{D(\Sigma),n} \arrow[r, "\overline{U}_0"] \arrow[d, "\text{YIELD}"'] & \mathcal{P}(Q^n \times Q \times T_{D(\Omega),n}) \arrow[d, "h"] \\
T_\Sigma(X_n) \arrow[r, "\overline{B}_n"'] & \mathcal{P}(Q^n \times Q \times T_\Omega(X_n))
\end{tikzcd}$$

where h transforms each element of $T_{D(\Omega),n}$ into its YIELD in $T_\Omega(X_n)$ (and perhaps "removes some parentheses"). Clearly, to prove our lemma, it suffices to show the commutativity of the above diagram for all n (in particular $n = 0$) . We shall do this, analogously to the case of tree homomorphisms, by finding four $\tilde{D}$-algebras such that the sets in the above diagram are their carriers of sort n , and such that the mappings in the diagram become $\tilde{D}$-homomorphisms (where, as before, $\tilde{D} = D(\Sigma) - \Sigma' = D(\Omega) - \Omega'$) . For the left side of the diagram we can choose the $\tilde{D}$-algebras $T_{D(\Sigma)}$ and $DT_\Sigma(X)$. For the right side of the diagram, let $TUP(Q)$ denote the (partial) $\tilde{D}$-algebra such that $Q^n \times Q$ is the domain of sort n , $\pi_i^n = \{(q_1, \ldots, q_n, q_i) \mid q_1, \ldots, q_n \in Q\}$, and

$c_{n,k}((q_1, \ldots, q_k, p_1), \ldots, (q_1, \ldots, q_k, p_n), (p_1, \ldots, p_n, p) = (q_1, \ldots, q_k, p)$

and undefined otherwise. Then the $\tilde{D}$-algebras $\mathcal{P}(TUP(Q) \times T_{D(\Omega)})$ and $\mathcal{P}(TUP(Q) \times DT_{\Omega}(X))$ have the proper domains (here, $\times$ denotes the obvious product of algebras, and $\mathcal{P}$ the subset algebra operation). Our diagram is now transformed into

$$\begin{tikzcd}
T_{D(\Sigma)} \arrow[r, "\overline{U}_0"] \arrow[d, "YIELD"'] & \mathcal{P}(TUP(Q) \times T_{D(\Omega)}) \arrow[d, "h"] \\
DT_{\Sigma}(X) \arrow[r, "\overline{B}"'] & \mathcal{P}(TUP(Q) \times DT_{\Omega}(X))
\end{tikzcd}$$

Obviously, YIELD and h are both $\tilde{D}$-homomorphisms. It follows easily from the definition of U that $\overline{U}_0$ is a $\tilde{D}$-homomorphism. Finally, it can be shown that $\overline{B}$ is also a $\tilde{D}$-homomorphism (here the determinism of B is essential; the proof is similar to that of Lemma 3.3(2)). We leave it to the reader to check the details. Now, $T_{D(\Sigma)}$ is the free $\tilde{D}$-algebra generated by Σ'. But, for $f \in \Sigma_n$, $h(\overline{U}_0(f')) = h(U_0(f')) =$ $\{(q_1, \ldots, q_n, YIELD(COMB_n^{\Omega}(t))) \mid (q_1, \ldots, q_n, q, t) \in B_n(f)\} = B_n(f) =$ $\overline{B}_n(f(x_1 \ldots x_n)) = \overline{B}_n(YIELD(f'))$. Hence $h \circ \overline{U}_0 = \overline{B} \circ YIELD$, and the lemma is proved. □

As a corollary we obtain the main theorem of this section.

(6.6) <u>Theorem</u>. The class of IO tree languages is closed under deterministic bottom-up tree transducer mappings.

Proof. Let L be an IO tree language over Σ . Thus L = YIELD(R) for some recognizable R over a finite subalphabet D' of $D(\Sigma)$. By the previous lemma, for any deterministic bottom-up tree transducer B , B(L) = B(YIELD(R)) = YIELD(U′(R)) where U′ is a linear bottom-up tree transducer from D' to $D(\Omega)$. Since the class of recognizable tree languages is closed under linear tree transducer mappings ([9 ,34]), U′(R) is recognizable over $D(\Omega)$, and hence its YIELD is an IO tree language.

□

Rounds [29, 30] has shown that the class of OI tree languages is closed under linear top-down tree transducer mappings (and hence under linear bottom-up tree transducer mappings, see [9]). This closure result and that of Theorem 6.6 are optimal in the sense that the OI tree languages are not closed under copying (more precisely, under tree homomorphisms), whereas the IO tree languages are not closed under nondeterminism (more precisely, under nondeterministic relabeling). This can easily be shown from the examples given by Fischer to show the incomparability of the classes of IO and OI string languages (for a definition of these string languages, see Definition 7.8 or [12]).

(6.7) <u>Example</u>. (The OI tree languages are not closed under tree homomorphisms).

The string language $L = \{b^m(ab^m)^{n-1} \mid m \geq 1 \ , \ n = 2^m\}$ is not an OI string language ([12]). Let $G = (\Sigma, \mathcal{F}, P)$ be the OI tree grammar with $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{g\}$, $\Sigma_2 = \{c\}$, $\mathcal{F}_0 = \{S\}$, $\mathcal{F}_1 = \{F\}$ and P consists of the rules

$S \rightarrow F(b)$,

$F(x_1) \rightarrow g(F(c(x_1 \ b)))$ and

$F(x_1) \rightarrow g(x_1)$.

Let h be the tree homomorphism with $h_1(g) = c(x_1 \ c(ax_1))$ and the identity on the other symbols (i.e. $h_0(a) = a$, $h_0(b) = b$ and $h_2(c) = c(x_1x_2))$. Then $\text{yield}(h(L_{OI}(G, S))) = L$. Hence $h(L_{OI}(G, S))$ is not an OI tree language.

$\square$

(6.8) <u>Example</u>. (The IO tree languages are not closed under nondeterministic relabeling).

The string language $L = \{w \in \{a, b\}^* \mid$ the number of symbols b in w is 2^n for some $n \geq 0\}$ is not an IO string language ([12]). Consider the IO tree grammar $G = (\Sigma, \mathcal{F}, P)$ with $\Sigma_0 = \{a, b, e\}$, $\Sigma_2 = \{c\}$,

$\mathcal{F}_0 = \{S,A\}$, $\mathcal{F}_1 = \{F\}$ and P consists of the rules

$$S \to c(A\ F(c(bA)))\ ,$$

$$A \to c(aA)\ ,\ A \to e\ ,$$

$$F(x_1) \to F(c(x_1 x_1))\ ,\ F(x_1) \to x_1\ .$$

(G is obtained from the macro grammar with rules $S \to AF(bA)$, $A \to aA$, $A \to \lambda$, $F(x_1) \to F(x_1 x_1)$, $F(x_1) \to x_1$ by replacing λ by e and writing c for concatenation). It is easy to see that $\text{yield}(L_{IO}(G,S)) =$ $= \{a^m (ba^k)^{2^n} \mid m\ ,\ k\ ,\ n \geq 0\}$ (note that $\text{yield}(e) = \lambda$) . Let h be the nondeterministic relabeling which relabels a by a or e , and leaves the other symbols (b , e and c) the same. Then $\text{yield}(h(L_{IO}(G,S))) = L$. Hence $h(L_{IO}(G,S))$ is not an IO tree language.

We note that the same argument shows that the class of IO string languages is not closed under nondeterministic relabeling: let h' relabel a by a or f , then $h'(\text{yield}(L_{IO}(G,S)))$ is not an IO string language (the IO string languages are closed under string homomorphisms, in particular the one which sends f into λ) .

$\square$

7. Hierarchies.

We have seen in sections 3, 4 and 5 that any system of context-free Σ-equations over some Σ-algebra A may be replaced by a system of regular $D(\Sigma)$-equations over some appropriate $D(\Sigma)$-algebra connected to A. Several authors [17,37,42] have suggested that this process may be iterated, i.e. one may consider systems of context-free $D(\Sigma)$-equations which may then be replaced by regular $D(D(\Sigma))$-equations (note that this requires the generalization of "derived alphabet" and other notions to the many-sorted case). In general one may consider systems of regular $D^n(\Sigma)$-equations over an appropriate $D^n(\Sigma)$-algebra A_n corresponding to A. Roughly speaking, A_n consists of "nondeterministic" functions of functions of .... of functions (up to level n) over A. In particular each A_n contains the subsets of A. Thus, for growing n, one obtains more and more (at least not less) subsets of A which are solutions of systems of equations, i.e. a hierarchy of higher level equational subsets of A.

In this section we shall show that in fact two such hierarchies can be defined over every Σ-algebra: an IO hierarchy and an OI hierarchy. For both hierarchies an MW-like result can be proved, which, approximately, says that the "level n" equational sets can be obtained by applying the mapping $YIELD^n$ to the recognizable tree languages over $D^n(\Sigma)$ (in the IO case), or the recognizable infinite trees over $D^n(\Sigma)^+$ (in the OI case). In the particular case of a "monadic string algebra" the first three steps in the hierarchy are, in the IO case: the regular, context-free and IO string languages, and in the OI case: the regular, context-free and OI string languages. We conjecture that the OI hierarchy is the same as that in [42]. The IO hierarchy is the one suggested in [17].

This section is organized as follows. Firstly we generalize a number of notions to the manysorted case. We state a theorem saying that, as for recognizable tree languages, no new IO and OI tree languages result from this generalization. Secondly we define the level n IO equational and level n OI equational subsets of an algebra. We then prove the above mentioned MW-like result for the IO case, and consider the IO string hierarchy. Finally we briefly treat the OI case.

The reader is now asked to generalize most of the concepts and facts treated sofar to the case of a many-sorted alphabet. In order to assist him, we shall define a number of these generalized concepts and leave it to the reader to check their properties (note that in section 2 and 4 several many-sorted concepts have already been defined; see also [17]).

Let S be a set of sorts. For $w \in S^*$ and $1 \leq i \leq lg(w)$, we shall denote by w_i the i^{th} symbol of w, thus $w = w_1 w_2 \ldots w_n$, where $n = lg(w)$ and $w_1, \ldots, w_n \in S$. Let $\Sigma = \langle \Sigma_{w,s} \rangle$ be an S-sorted alphabet $(\langle w,s \rangle \in S^* \times S)$. First of all we need the generalized notion of derived alphabet.

The <u>derived</u> $(S^* \times S)$-sorted <u>alphabet of Σ</u>, denoted by $D(\Sigma)$, is obtained as follows. Let, for each $\langle w,s \rangle \in S^* \times S$ and each $f \in \Sigma_{w,s}$, f' be a new symbol; let for each $w \in S^*$ $(w \neq \lambda)$ and each i, $1 \leq i \leq lg(w)$, π_i^w be a new symbol; and let for each $w, v \in S^*$ and $s \in S$, $c_{w,v,s}$ be a new symbol. Then $D(\Sigma)$ consists of these new symbols with their types (elements of $(S^* \times S)^* \times (S^* \times S)$) specified as follows:

(i) for $f \in \Sigma_{w,s}$, f' has type $\langle \lambda, \langle w,s \rangle \rangle$;

(ii) π_i^w has type $\langle \lambda, \langle w, w_i \rangle \rangle$; and

(iii) $c_{w,v,s}$ has type $\langle \langle w,s \rangle \langle v, w_1 \rangle \ldots \langle v, w_n \rangle, \langle v,s \rangle \rangle$

(in particular, $c_{\lambda,v,s}$ has type $\langle \langle \lambda, s \rangle, \langle v,s \rangle \rangle$).

In the ranked case $(S = \{s\})$, to remain consistent with Definition 2.2.1, one has to identify π_i^w with $\pi_i^{lg(w)}$, and $c_{w,v,s}$ with $c_{lg(w),lg(v)}$.

The <u>derived alphabet of order n</u>, denoted by $D^n(\Sigma)$, is defined by $D^0(\Sigma) = \Sigma$ and $D^{n+1}(\Sigma) = D(D^n(\Sigma))$.

In the place of X, we shall use the set of (sorted) <u>variables</u> $X_S = \{x_{i,s} \mid i \geq 1 \text{ and } s \in S\}$. The symbol $x_{i,s}$ is meant to be a constant of sort s. Let $X_\lambda = \phi$ and, for every $w \in S^*$ $(w \neq \lambda)$, $X_w = \{x_{i,w_i} \mid 1 \leq i \leq lg(w)\}$. For $w \in S^*$, X_w will also be used to denote the family of disjoint sets $Y = \langle Y_s \rangle_{s \in S}$ where $Y_s = \{x_{i,w_i} \mid w_i = s\}$. Thus $T_\Sigma(X_w)$ denotes $T_\Sigma(Y)$ as defined in section 2. 2. Note that in the ranked case these concepts are the usual ones.

The <u>tree substitution $D(\Sigma)$-algebra</u>, denoted by $DT_\Sigma(X_S)$, or $DT_\Sigma(X)$ if S is understood, is defined analogously to the ranked case. The domain of sort $\langle w, s\rangle$ is $T_\Sigma(X_w)_s$; for $f \in \Sigma_{w,s}$, f' is the tree $f(x_{1,w_1} \ldots x_{n,w_n})$; $\pi_i^w = x_{i,w_i}$ and $c_{w,v,s}(t, t_1, \ldots, t_n) = t[t_1, \ldots, t_n]$, the result of substituting t_i for x_{i,w_i} in t. The unique $D(\Sigma)$-homomorphism $T_{D(\Sigma)} \to DT_\Sigma(X)$ is called YIELD.

Let now A be a Σ-algebra. The <u>$D(\Sigma)$-algebra of functions over A</u>, denoted by $\mathfrak{F}(A)$, has the set of all functions $A_{s_1} \times \ldots \times A_{s_n} \to A_s$ as domain of sort $\langle s_1 \ldots s_n, s\rangle$ (in particular, $\mathfrak{F}(A)_{\langle\lambda, s\rangle} = A_s$); $\pi_i^{s_1 \ldots s_n}$ is the i^{th} projection and $c_{w,v,s}$ is the usual composition of functions. Every $t \in T_\Sigma(X_w)_s$ gives rise to a derived operation t_A in $\mathcal{F}(A)_{<w,s>}$ (see section 2.2). Note that for $w = \lambda$, $t_A = h_A(t)$, where h_A is the Σ-homomorphism $T_\Sigma \to A$. We also denote t_A by $\mathrm{derop}_A(t)$; derop_A is the unique $D(\Sigma)$-homomorphism $DT_\Sigma(X) \to \mathcal{F}(A)$. $\mathcal{F}$ can be iterated and $\mathcal{F}^n(A)$ is clearly a $D^n(\Sigma)$-algebra $(n \geq 0)$. For a Δ-continuous Σ-algebra B with $\bigsqcup$-complete carriers, the <u>$D(\Sigma)$-algebra of Δ-continuous functions over B</u>, denoted by $\mathcal{F}_\Delta(B)$, is defined in the obvious way. By Lemma 5.14, $\mathcal{F}_\Delta$ can be iterated and $\mathcal{F}_\Delta^n(B)$ is a Δ-continuous $D^n(\Sigma)$-algebra with $\bigsqcup$-complete carriers.

Finally we define an <u>S-sorted context-free tree grammar</u> to be a triple $G = (\Sigma, \mathcal{F}, P)$, where Σ and $\mathcal{F}$ are disjoint finite S-sorted alphabets and P is a finite set of productions of the form $F(x_{1,w_1} \cdots x_{k,w_k}) \to \tau$, where $k \geq 0$, $F \in \mathcal{F}_{w,s}$ and $\tau \in T_{\Sigma \cup \mathcal{F}}(X_w)_s$ for $w = w_1 \cdots w_k$ and some s in S. The definitions of derivation and generated language are completely analogous to the ranked case.

This ends our list of generalizations.

We shall first show that in the many-sorted case no new IO and OI tree languages are obtained. As before, for any S-sorted alphabet Σ, we shall denote by $\overline{\Sigma}$ the ranked alphabet associated to Σ in a natural way: for $n \geq 0$ $\overline{\Sigma}_n = \bigcup\limits_{w,s} \{\Sigma_{w,s} \mid lg(w) = n\}$, that is, $\overline{\Sigma}_n$ consists of all symbols of Σ of rank n .

(7.1) Theorem. Let Σ be a finite S-sorted alphabet and $\overline{\Sigma}$ its associated (finite) ranked alphabet. Let L be a tree language contained in $T_{\Sigma,s}$ for some $s \in S$. Then L is an IO (OI) tree language over Σ if and only if it is an IO (OI) tree language over $\overline{\Sigma}$.

Proof. Let us note first of all that S may be assumed to be finite. The only-if part of the statement is easy. One simply changes a given many-sorted context-free tree grammar $G = (\Sigma, \mathfrak{F}, P)$ into the ordinary context-free tree grammar $G_1 = (\overline{\Sigma}, \overline{\mathfrak{F}}, P_1)$ where P_1 is obtained from P by "dropping the sorts of the variables" (i.e. replacing each $x_{i,s}$ by x_i in all rules). It is easy to see that $L(G_1, A) = L(G, A)$ for any $A \in \overline{\mathfrak{F}}_0$ in both the IO and OI mode of derivation.

For the if-part, let L be an arbitrary IO (OI) tree language over $\overline{\Sigma}$ (not necessarily contained in $T_{\Sigma,s}$) . It suffices to show that $L \cap T_{\Sigma,s}$ is an IO (OI) tree language over Σ. Clearly $T_{\Sigma,s}$ is a recognizable tree language over $\overline{\Sigma}$ (use the set of sorts together with one "rejecting state" as the elements of the obvious finite Σ-algebra recognizing $T_{\Sigma,s}$) . Now the IO (OI) tree languages are closed under intersecti on with a recognizable treelanguage (for IO,see Corollary 6. 2; for OI, see [29, 30]). Thus, in particular, $L \cap T_{\Sigma,s}$ is an IO (OI) tree language over $\bar{\Sigma}$. However, by inspecting the constructions involved in the above mentioned proofs, it can be seen that one ends up with a grammar for $L \cap T_{\Sigma,s}$ which can easily be changed into a many-sorted grammar by associating types with nonterminals. In fact the only problem is deletion: a nonterminal might produce a "non-sorted" sub tree, which is deleted later in the derivation. In the IO case this is solved by starting with a nondeleting grammar for L (see [12]), and in the OI case by simply not deriving the wrong sub tree.

$\square$

We now turn to the definition of the hierarchies of higher level equational subsets of a Σ-algebra A . First we note that the IO equational subsets of A (see section 5) can also be characterized as the solutions of systems of regular $D(\Sigma)$-equations in the $D(\Sigma)$-algebra $\mathcal{P}(\mathfrak{T}(A))$ rather than in $\mathfrak{R}(A)$ (there is a $\bigsqcup$-continuous $D(\Sigma)$-homomorphism $h : \mathcal{P}(\mathfrak{T}(A)) \to \mathfrak{R}(A)$; in fact, for $Q \subseteq \mathfrak{T}(A)$, $h(Q) = \bigsqcup Q$, where $\mathfrak{T}(A)$ is considered as sub algebra of $\mathfrak{R}(A)$ in the obvious way; hence, since h is the identity on $\mathcal{P}(A)$, the $D(\Sigma)$-equational elements of $\mathcal{P}(A)$ are the same in $\mathcal{P}(\mathcal{I}(A))$ and $\mathcal{L}(A))$. Moreover, from an intuitive point of view, it is perhaps more natural to solve $D(\Sigma)$-equations in $\mathcal{P}(\mathcal{I}(A))$, where a set of derived operators (i.e. $L \subseteq T_\Sigma(X)$) is interpreted as a set of derived operations, rather than a derived relation.

Consider a derived alphabet $D^n(\Sigma)$ of order n. Intuitively, the composition symbols in $D^n(\Sigma)$ should be interpreted as composition of functions of level n over some domain. Two natural choices of a $D^n(\Sigma)$-algebra connected to a Σ-algebra A are $\mathcal{P}(\mathcal{I}^n(A))$ in the IO case and $\mathcal{I}^n_\Delta(\mathcal{P}(A))$ in the OI case (obtained by iterating the $\mathcal{I}$ in $\mathcal{P}(\mathcal{I}(A))$ and $\mathcal{I}_\Delta(\mathcal{P}(A))$ respectively).

(7.2) <u>Definition.</u> Let Σ be an S-sorted alphabet and A a Σ-algebra. Let $n \geq 0$ and $s \in S$.

(i) A subset of A_s is <u>IO(n) equational</u> if it is equational as an element of the $\bigsqcup$-continuous $D^n(\Sigma)$-algebra $\mathcal{P}(\mathcal{I}^n(A))$.

(ii) A subset of A_s is <u>OI(n) equational</u> if it is equational as an element of the Δ-continuous $D^n(\Sigma)$-algebra $\mathcal{I}^n_\Delta(\mathcal{P}(A))$.

More generally, for a Δ-continuous Σ-algebra B with

$\bigsqcup$ -complete carriers, an element of B_s is OI(n) equational if it is equational as an element of the $D^n(\Sigma)$-algebra $\mathcal{T}^n_\Delta(B)$.

$\square$

The sort of the elements of $\mathcal{P}(\mathcal{T}^n(A))$ and $\mathcal{T}^n_\Delta(\mathcal{P}(A))$ in which we are interested (i. e, for every $s \in S$, the subsets of A_s) will be denoted by $t_n(s)$. Thus

$$t_0(s) = s \text{ and } t_{n+1}(s) = \langle\lambda, t_n(s)\rangle .$$

Note that $\mathcal{T}(A)_{\langle\lambda, s\rangle} = A_s$, $\mathcal{P}(A)_s = \mathcal{P}(A_s)$ and $\mathcal{T}_\Delta(B)_{\langle\lambda, s\rangle} = B_s$.

(7. 3) <u>Example</u>. Let Σ be the S-sorted alphabet with $S = \{s\}$, $\Sigma_{\lambda, s} = \{a\}$ and $\Sigma_{ss, s} = \{g\}$ (thus Σ is a ranked alphabet with a constant a and a binary operator g ; for the sake of the example we shall use the many-sorted notation). Consider the $S^* \times S$-sorted context-free tree grammar $G = (D(\Sigma), \mathcal{T}, P)$, where $\mathcal{T}_{\lambda, \langle\lambda, s\rangle} = \{Q\}$, $\mathcal{T}_{\langle s, s\rangle, \langle s, s\rangle} = \{F\}$ and the productions are

$$Q \rightarrow c_{s,\lambda,s} (F(c_{ss,s,s}(g' \pi_1^{\langle s,s\rangle} \pi_1^{\langle s,s\rangle}))a')$$

$$F(x_{1,\langle s,s\rangle}) \rightarrow F(c_{s,s,s}(x_{1,\langle s,s\rangle} \, x_{1,\langle s,s\rangle})) \text{ , and}$$

$$F(x_{1,\langle s,s\rangle}) \rightarrow x_{1,\langle s,s\rangle} .$$

Then G^D is a system of regular $D^2(\Sigma)$-equations (cf. Definition 4. 5).

Thus, for any Σ-algebra A, the solution of Q in the $D^2(\Sigma)$-algebra $\mathcal{P}(\mathfrak{T}^2(A))$ $(\mathfrak{T}^2_\Delta(\mathcal{P}(A)))$ is an IO(2) equational (OI(2) equational) subset of A. Note that the sort of this solution is $t_2(s) = \langle\lambda, \langle\lambda, s\rangle\rangle$. Without all the confusing sub and super scripts the grammar G looks like this:

$$Q \to c(F(d)a') \text{ , where } d = c(g' \pi_1 \pi_1) \text{ ,}$$

$$F(x) \to F(c(xx)) \text{ and}$$

$$F(x) \to x \text{ .}$$

Consider the Σ-algebra A with domain a^* (all strings of symbols a) $a_A = a$ and g_A is concatenation. We shall now solve the above equations in the algebras $\mathcal{P}(\mathfrak{T}^2(A))$ and $\mathfrak{T}^2_\Delta(\mathcal{P}(A))$, using informal but intuitively clear notation. Firstly, in $\mathcal{P}(\mathfrak{T}^2(A))$, $F = \{F_n \mid n \geq 0\}$ where $F_n \in \mathfrak{T}^2(A)$ denotes the function such that, for any $f: A \to A$, $F_n(f) = f^{2^n}$. Since $d: A \to A$ and for every $w \in A$ $d(w) = ww$, $F(d)$ is the set of functions $\{d^{2^n} \mid n \geq 0\}$ and $c(F(d)a') = \{d^{2^n}(a) \mid n \geq 0\} = \{a^{2^{2^n}} \mid n \geq 0\}$.

Secondly, in $\mathfrak{T}^2_\Delta(\mathcal{P}(A))$, $F = \bigsqcup_{n\geq 0} F_n$, where $F_n \in \mathfrak{T}^2_\Delta(\mathcal{P}(A))$ denotes the function such that, for any $f: \mathcal{P}(A) \to \mathcal{P}(A)$, $F_n(f) = f^{2^n}$. Since $d: \mathcal{P}(A) \to \mathcal{P}(A)$ and for every $B \subseteq A$ $d(B) = BB$, $F(d) = \bigsqcup_{n\geq 0} F_n(d) = \bigsqcup_{n\geq 0} d^{2^n}$ and $c(F(d)a') = \bigsqcup_{n\geq 0} d^{2^n}(\{a\}) = \bigcup_{n\geq 0} \{a\}^{2^{2^n}} = \{a^{2^{2^n}} \mid n \geq 0\}$. Hence the language $\{a^{2^{2^n}} \mid n \geq 0\}$ is both an IO(2) equational and OI(2) equational subset of a^*.

□

Some elementary facts are stated briefly in the following lemma.

(7.4) Lemma.

(1) IO(0) equational = OI(0) equational = equational ;

(2) IO(1) equational = IO equational ,

OI(1) equational = OI equational ;

(3) for tree languages,

IO(0) equational = OI(0) equational = recognizable;

IO(1) equational = IO tree language ,

OI(1) equational = OI tree language (where, for an infinite alphabet, only tree languages over finite sub alphabets are considered);

(4) for all $n \geq 0$, IO(n) equational implies IO(n+1) equational, and OI(n) equational implies OI(n+1) equational.

□

In the next theorem we shall prove an MW-like result for IO(n) equational subsets, which shows that the IO hierarchy is the one intended in [17]. From [21] we know that, for a Σ-algebra A and $s \in S$, a subset of A_s is equational iff it is the Σ-homomorphic image of a recognizable Σ-tree language. From sections 4 and 5 we know that a subset of A_s is IO equational iff it is the Σ-homomorphic image of the YIELD of a recognizable $D(\Sigma)$-tree language (of sort $\langle\lambda, s\rangle$) . The general result will be that a subset of A_s is IO(n) equational iff it is the Σ-homomorphic image of the YIELD^n of a recognizable $D^n(\Sigma)$-tree language (of sort $t_n(s)$) . Note that YIELD maps $T_{D(\Sigma), \langle\lambda, s\rangle}$ to $T_{\Sigma, s}$. Hence, by induction, YIELD^n maps $T_{D^n(\Sigma), t_n(s)}$ to $T_{\Sigma, s}$.

(7.5) Theorem. Let Σ be an S-sorted alphabet, A a Σ-algebra and h_A the Σ-homomorphism $T_\Sigma \to A$. For any $s \in S$ and $n \geq 0$, a subset of A_s is IO(n) equational if and only if it is the image under $h_A \circ \mathrm{YIELD}^n$ of a recognizable tree language in $T_{D^n(\Sigma), t_n(s)}$. In particular, a tree language over Σ is IO(n) equational iff it is $\mathrm{YIELD}^n(L)$ for some recognizable tree language L over $D^n(\Sigma)$ (of appropriate sort).

$\square$

Proof. If h_n denotes the unique $D^n(\Sigma)$-homomorphism $T_{D^n(\Sigma)} \to \mathcal{F}^n(A)$, then the solution of any system of regular $D^n(\Sigma)$-equations in $\mathcal{P}(\mathcal{F}^n(A))$ is the h_n image of its solution in $\mathcal{P}(T_{D^n(\Sigma)})$, which is a recognizable $D^n(\Sigma)$-tree language (see Lemma 5.2 and section 2.3). Thus it suffices to show that $h_n = h_A \circ \mathrm{YIELD}^n$ on any sort $t_n(s)$. We show this by induction on n. For $n = 0$ there is nothing to prove. For $n = 1$ it is clear that $h_1 = h_{\mathcal{F}(A)} = \mathrm{derop}_A \circ \mathrm{YIELD}$. Moreover $\mathrm{derop}_A(t) = h_A(t)$ for each t of sort $<\lambda, s>$. Hence $h_1 = h_A \circ \mathrm{YIELD}$. Suppose that the statement is true for n. We now apply the case $n = 1$ to the case $n+1$ by taking $D^n(\Sigma)$ instead of Σ and $\mathcal{F}^n(A)$ instead of A, as follows (note that $h_n = h_{\mathcal{F}^n(A)}$):

$$\begin{aligned} h_{n+1} &= \mathrm{derop}_{\mathcal{F}^n(A)} \circ \mathrm{YIELD} \\ &= h_n \circ \mathrm{YIELD} \qquad \text{on } t_{n+1}(s) \\ &= h_A \circ \mathrm{YIELD}^n \circ \mathrm{YIELD} = h_A \circ \mathrm{YIELD}^{n+1}. \end{aligned}$$

This proves the statement and the theorem.

□

We note that originally the proof of this theorem was much longer. The present short proof is due to Damm [44].

As an immediate consequence of Theorem 7.5 we state the following MW-like corollary.

(7.6) Corollary. Let Σ be an S-sorted alphabet, A a Σ-algebra, $n \geq 0$ and $s \in S$. A subset of A_s is IO(n) equational iff it is the Σ-homomorphic image of an IO(n) equational tree language over Σ (of sort s).

□

(7.7) Example. Consider Example 7.3. It is left to the reader to show that the IO(2) equational subset $\{a^{2^{2^n}} \mid n \geq 0\}$ is $h_A(\mathrm{YIELD}(L))$, where L is the IO $D(\Sigma)$-tree language generated by the nonterminal Q of grammar G. For instance, without sub and super scripts, $Q \overset{*}{\Rightarrow} c(c(dd)a')$ and $h_A(\mathrm{YIELD}(c(c(dd)a'))) = h_A(g(g(aa)g(aa))) = a^4$.

□

As an example we now consider the IO hierarchy of string languages.

Let Ω be an ordinary (finite) alphabet. There are two well known ways of considering Ω^* as a Σ-algebra. Firstly, let Ω^c be the ranked alphabet determined by $\Omega_0^c = \Omega \cup \{e\}$ and $\Omega_2^c = \{c\}$, where e and c are new symbols. Ω^* is viewed as an Ω^c-algebra, denoted by Ω_c^*, by defining e to be the empty string λ, every symbol in Ω to be itself, and

c to be string concatenation. The unique Ω^c-homomorphism $T_{\Omega^c} \to \Omega^*_c$ is called yield (cf. Example 2.2.3; note that "yield" differs from the usual one by the fact that yield(e) = λ). Secondly, let Ω^m be the (monadic) ranked alphabet determined by $\Omega^m_0 = \{e\}$ and $\Omega^m_1 = \Omega$.
Ω^* is viewed as an Ω^m-algebra, denoted by Ω^*_m , by defining $e = \lambda$ and, for $w \in \Omega^*$ and $a \in \Omega$, $a(w) = aw$ (thus $a \in \Omega^m_1$ is interpreted as left concatenation with a) . Since the unique Ω^m-homomorphism $T_{\Omega^m} \to \Omega^*_m$ is clearly an isomorphism, we shall not distinguish between T_{Ω^m} and Ω^*_m as Ω^m-algebras.

Thus an IO hierarchy of string languages over Ω can be build up in two ways: by viewing Ω^* either as an Ω^c-algebra or as an Ω^m-algebra. It is well known that the equational subsets of Ω^*_c are the context-free languages over Ω ([21]; e causes no problem).

Let us now define IO (and OI) string languages. It should be clear to the reader that our definition is equivalent to the usual one in [12].

(7.8) Definition. Let Ω be an alphabet. A language over Ω is called an IO string language (OI string language) over Ω if it is the yield of an IO tree language (OI tree language) over Ω^c .

□

It follows directly from this definition and Corollaries 5.11 and 5.18 that the IO (OI) equational subsets of Ω_c^* are the IO (OI) string languages over Ω. Thus the hierarchy of IO(n) equational languages in Ω_c^* starts with the context-free and the IO string languages. We now consider Ω_m^*.

(7.9) <u>Theorem</u>. Let Ω be an alphabet. For $n = 0$, 1 and 2 the IO(n) equational subsets of Ω_m^* are the regular, context-free and IO string languages respectively.

<u>Proof</u>. For $n = 0$ the statement should be clear from the isomorphism between Ω_m^* and T_{Ω^m}. For $n = 1$, 2 weknow from Theorem 7.5 that an IO(n) equational subset of T_{Ω^m} is the YIELD of an IO(n−1) equational subset of $T_{D(\Omega^m),0}$. Hence the IO(1) equational subsets of Ω_m^* are the YIELDs of recognizable tree languages in $T_{D(\Omega^m),0}$, and the IO(2) equational subset of Ω_m^* are the YIELDs of IO tree languages in $T_{D(\Omega^m),0}$.

Consider, informally, some tree in $T_{D(\Omega^m),0}$. Then one notices that it contains in general many superfluous subtrees (w.r.t. YIELD), due to the fact that, since all elements of Ω^m have rank 1 or 0, trees in $T_{\Omega^m}(X)$ can contain at most one variable. In fact, one may see that any tree t_1 in T_{Ω^m} is YIELD of a tree t_2 in $T_{D(\Omega^m),0}$ in which only the symbols from $\Omega \cup \{e, c_{1,1}, c_{1,0}\}$ are used. Moreover, for such a tree t_2, when $c_{1,1}$ and $c_{1,0}$ are identified with c, $\mathrm{YIELD}(t_2) = \mathrm{yield}(t_2) = t_1$.

Formally we shall show the existence of two mappings $B_1 : T_{D(\Omega^m),0} \to T_{\Omega^c}$ and $B_2 : T_{\Omega^c} \to T_{D(\Omega^m),0}$ such that $\mathrm{yield}(B_1(t)) = \mathrm{YIELD}(t)$ and $\mathrm{YIELD}(B_2(t)) = \mathrm{yield}(t)$. Thus

$$\begin{tikzcd}
T_{D(\Omega^m),0} \arrow[rr, shift left, "B_1"] \arrow[dr, "\mathrm{YIELD}"'] & & T_{\Omega^c} \arrow[ll, shift left, "B_2"] \arrow[d, "\mathrm{yield}"] \\
& T_{\Omega^m} = & \Omega_c^*
\end{tikzcd}$$

Moreover we shall show that B_1 and B_2 preserve the properties of being a recognizable tree language and being an IO tree language. From this the theorem can be proved as follows. For $n = 1, 2$, if L is an IO(n) equational subset of Ω_m^*, then $L = \mathrm{YIELD}(R)$ for some IO(n-1) equational tree language R in $T_{D(\Omega^m),0}$. Hence $L = \mathrm{yield}(B_1(R))$ and $B_1(R)$ is an IO(n-1) equational tree language in T_{Ω^c}. Therefore, L is an IO(n-1) equational subset of Ω_c^*, i.e. for $n = 1$ a context-free language and for $n = 2$ an IO string language. Conversely, for $n = 1, 2$, if L is an IO(n-1) equational subset of Ω_c^* , the L = yield(R) for some IO(n-1) equational tree language R over Ω^c . Hence L = YIELD(B_2(R)) and B_2(R) is an IO(n-1) equational tree language over D(Ω^m) . Therefore L is an IO(n) equational subset of Ω_m^* .

We now show the existence of B_1 and B_2 . B_1 will be realized as a linear deterministic bottom-up tree transducer mapping from D(Ω^m) to Ω^c (in fact from $\overline{D(\Omega^m)}$ to $\overline{\Omega^c}$) . It then follows from section 6 and Theorem 7.1 that B_1 preserves the recognizable and the IO tree languages. More precisely, B_1 only exists for each finite subalphabet of D(Ω^m) and this clearly suffices for our purpose. We shall however (as we did in section 6) construct an infinite B_1 for D(Ω^m) and leave it to the reader to restrict B_1 to a finite transducer for each finite subalphabet of D(Ω^m) . For notation, see section 6.

B_1 has states $Q = \mathbb{N}$ and final states $F = Q$. The family of mappings $\langle B_n \rangle_{n \in \mathbb{N}}$ is specified as follows:

(i) for $1 \le i \le n$, $B_0(\pi_i^n) = (i, e)$;

(ii) $B_0(e') = (0, e)$;

(iii) for $f \in \Omega$, $B_0(f') = (1, f)$;

(iv) for $n, k \ge 0$, $B_{n+1}(c_{n,k})$ is a mapping $Q^{n+1} \to Q \times T_{\Sigma^c}(X_{n+1})$

such that, for $j, i_1, \ldots, i_n \in N$,

$$B_{n+1}(c_{n,k})(j, i_1, \ldots, i_n) = \begin{cases} (j, x_1) & \text{if } j = 0 \\ (i_j, c(x_1 x_{j+1})) & \text{if } 1 \le j \le n \\ \text{arbitrary otherwise.} & \end{cases}$$

We note that the elements of $T_{\Omega^m}(X)$ can be identified with the strings in $\Omega^* \cup \Omega^* X$. We leave it to the reader to check that (with this identification) for $t \in T_{D(\Omega^m)}$, $i \in \mathbb{N}$ and $t' \in T_{\Omega^c}$, if $\overline{B}_0(t) = (i, t')$ then either $i = 0$ and $\text{YIELD}(t) = \text{yield}(t')$, or $i \ge 1$ and $\text{YIELD}(t) = \text{yield}(t') . x_i$. Hence, for all $t \in T_{D(\Omega^m), 0}$, $\text{yield}(B_1(t)) = \text{YIELD}(t)$.

The mapping $B_2 : T_{\Omega^c} \to T_{D(\Omega^m), 0}$ is easy to describe: for $s \in T_{\Omega^c}$, $B_2(s) = c_{1,0}(te')$, where t is obtained from s by relabeling c as $c_{1,1}$, each $f \in \Omega$ as f', and e as π_1^1. It is easy to see that $t \in T_{D(\Omega^m), 1}$ and $\text{YIELD}(t) = \text{yield}(s). x_1$. Hence $B_2(s) \in T_{D(\Omega^m), 0}$ and $\text{YIELD}(B_2(s)) = \text{yield}(s)$. It should be obvious that B_2 preserves the recognizable and IO tree languages (cf. Theorem 7.1). This concludes the proof of the theorem.

$\square$

Note that, if, as we conjecture, the IO(n) equational tree languages are closed under deterministic bottom-up tree transducer mappings, then the proof of the previous theorem shows that, for all n, the IO(n) equational subsets of Ω^*_m equal the IO(n-1) equational subsets of Ω^*_c.

We conclude this section by considering the OI case. This case can be treated completely analogous to the IO case, using infinite trees (with +) rather than tree languages. Since infinite trees are more or less outside the scope of this paper, no detailed proofs will be given. For notation and some properties of infinite trees we refer to the last part of section 5. The generalization to the many-sorted case is understood (cf. [15]). Recall that, for any S-sorted alphabet Σ, $\Sigma^+ = \Sigma \cup \{+_s \mid s \in S\}$, SET denotes the homomorphism $CT_{\Sigma^+} \to \mathcal{P}(T_\Sigma)$, and YIELD denotes the homomorphism $CT_{D(\Sigma)^+} \to CT_{\Sigma^+}(X_S)$. Note that YIELD maps $CT_{D(\Sigma)^+,\langle\lambda,s\rangle}$ into $CT_{\Sigma^+,s}$. Hence YIELD^n maps $CT_{D^n(\Sigma)^+,t_n(s)}$ into $CT_{\Sigma^+,s}$. Recall also that the solution of a system of regular (context-free) Σ^+-equations in CT_{Σ^+} is called a recognizable (context-free) infinite tree.

The following MW-like result for OI(n) equational elements is the analogue of Theorem 7.5.

(7.10) Theorem. Let Σ be an S-sorted alphabet, B a Δ-continuous Σ-algebra with $\bigsqcup$-complete carriers and h_B the unique Δ-continuous $\perp$-preserving Σ^+-homomorphism $CT_{\Sigma^+} \to B$. For any $s \in S$ and $n \geq 0$, an element of B_s is OI(n) equational if and only if it is the image under $h_B \circ \mathrm{YIELD}^n$ of a recognizable infinite tree in $CT_{D^n(\Sigma)^+, t_n(s)}$.

<u>Proof</u>. The proof is completely analogous to the proof of Theorem 7.5, using $\mathcal{I}_\Delta$ rather than $\mathcal{I}$, Δ-continuous $D^n(\Sigma)^+$-algebras rather than $D^n(\Sigma)$-algebras, Δ-continuous $\perp$-preserving $D^n(\Sigma)^+$-homomorphisms rather than $D^n(\Sigma)$-homomorphisms, $CT_{D^n(\Sigma)^+}$ rather than $T_{D^n(\Sigma)}$, and extending all $D^n(\Sigma)$-algebras with $\bigsqcup$-complete carriers to $D^n(\Sigma)^+$-algebras by defining + to be $\sqcup$.

$\square$

As a particular case of this theorem we obtain the following result for OI(n) equational subsets of a Σ-algebra.

(7.11) <u>Theorem</u>. Let Σ be an S-sorted alphabet, A a Σ-algebra and h_A the Σ-homomorphism $T_\Sigma \to A$. For $s \in S$ and $n \geq 0$, a subset of A_s is OI(n) equational if and only if it is the image under $h_A \circ \mathrm{SET} \circ \mathrm{YIELD}^n$ of a recognizable infinite tree in $CT_{D^n(\Sigma)^+, t_n(s)}$. In particular, a tree language over Σ is OI(n) equational iff it is $\mathrm{SET}(\mathrm{YIELD}^n(t))$ for some recognizable infinite tree over $D^n(\Sigma)^+$ (of appropriate sort).

Proof. Immediate from Theorem 7.10 by the fact that, for $B = \mathcal{P}(A)$,

$h_B = h_A \circ \mathrm{SET}$.

$\square$

An immediate consequence of this theorem is the following MW-like corollary.

(7.12) Corollary. Let Σ be an S-sorted alphabet, A a Σ-algebra, $n \geq 0$ and $s \in S$. A subset of A_s is OI(n) equational iff it is the Σ-homomorphic image of an OI(n) equational tree language over Σ (of sort s).

$\square$

(7.13) Example. Consider Example 7.3. It is left to the reader to show that the OI(2) equational subset $\{a^{2^{2^n}} \mid n \geq 0\}$ is $h_a(\mathrm{SET}(\mathrm{YIELD}(t)))$, where t is the infinite context-free $D(\Sigma)^+$-tree determined by the nonterminal Q of G . For instance

$$h_A(\mathrm{SET}(\mathrm{YIELD}(c(+(+(\ldots c(dd))d)a'))))) =$$

$$h_A(\mathrm{SET}(+(+(\ldots g(g(aa)g(aa)))g(aa)))) =$$

$$h_A(\{g(aa), g(g(aa)g(aa)), \ldots\}) = \{a^2, a^4, \ldots\} .$$

$\square$

To contrast again the IO and OI cases we state the next IO result, to be compared with Theorem 7.11.

(7.14) Theorem. Let Σ be an S-sorted alphabet, A a Σ-algebra and $h_A : T_\Sigma \to A$. For any $n \geq 0$ and $s \in S$, a subset of A_s is IO(n) equational iff it is the image under $h_A \circ \mathrm{YIELD}^n \circ \mathrm{SET}$ of a recognizable infinite tree over $D^n(\Sigma)^+$ (of sort $t_n(s)$).

Proof. Immediate from Theorem 7.5 and the fact that SET is a Δ-continuous $\perp$-preserving $D^n(\Sigma)^+$-homomorphism from $CT_{D^n(\Sigma)^+}$ into $\mathcal{P}(T_{D^n(\Sigma)})$.

$\square$

Thus IO(n) equational tree languages are obtained from the recognizable infinite trees in $CT_{D^n(\Sigma)^+}$ by application of $\mathrm{YIELD}^n \circ \mathrm{SET}$, and the OI(n) equational tree languages by application of $\mathrm{SET} \circ \mathrm{YIELD}^n$ (this generalizes the diagram at the end of section 5).

Next we consider the OI hierarchies of string languages. As in the

IO case, consider a string alphabet Ω and the two possible algebras Ω^*_c and Ω^*_m. Clearly, the hierarchy of OI(n) equational languages in Ω^*_c starts with the context-free and the OI string languages (see Definition 7.8). We now show that the hierarchy of OI(n) equational subsets of Ω^*_m starts with the regular, the context-free and the OI string languages (cf. [42]).

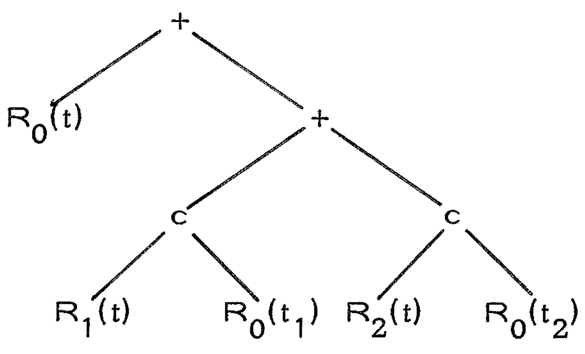
(7.15) Theorem. Let Ω be an alphabet. For $n = 0, 1, 2$ the OI(n) equational subsets of Ω^*_m are the regular, context-free and OI string languages respectively.

Proof. For $n = 0$ the statement is clear. For $n = 1, 2$ we shall only sketch a possible proof. From Theorem 7.10 it can be seen that the OI(1) equational subsets of Ω^*_m are the SET∘YIELDs of recognizable infinite trees over $D(\Omega^m)^+$ (of sort 0), whereas the OI(2) equational subsets of Ω^*_m are the SET∘YIELDs of context-free infinite trees over $D(\Omega^m)^+$ (of sort 0). Analogously to the proof of Theorem 7.9 it would suffice to have mappings T_1 and T_2 such that the following diagram commutes

$$\begin{tikzcd}
CT_{D(\Omega^m)^+,0} \arrow[rrr, shift left, "T_1"] \arrow[d, "\mathrm{YIELD}"'] & & & CT_{(\Omega^c)^+} \arrow[lll, shift left, "T_2"] \arrow[d, "\mathrm{SET}"] \\
CT_{(\Omega^m)^+} \arrow[dr, "\mathrm{SET}"'] & & & \mathcal{P}(T_{\Omega^c}) \arrow[dl, "\mathrm{yield}"] \\
& \mathcal{P}(T_{\Omega^m}) \arrow[r, phantom, "="] & \mathcal{P}(\Omega_c^*) &
\end{tikzcd}$$

and such that T_1 and T_2 preserve recognizability and context-freeness of infinite trees. As in the IO case, the existence of T_2 is obvious. For T_1, think of a deterministic top-down tree transducer (see [29]) working on an infinite tree. Taking the join of all initial pieces of output, one obtains an infinite tree as output of the transducer. Using arguments similar to the language case, one can prove that, in general, the classes of recognizable and context-free infinite trees are closed under deterministic top-down tree transducers (note that copying and deletion are no problem since the infinite tree is "generated by a deterministic grammar"). It should be clear that a deterministic top-down transducer R, realizing T_1, can be constructed (for any finite subalphabet of $D(\Omega^m)$). R has the set of states $\mathbb{N}$ and, denoting the output of R started in state i on input t by $R_i(t)$, it has the property that, for any tree t in $CT_{D(\Omega^m)+}$, $SET(YIELD(t)) = yield(SET(R_0(t))) \cup \bigcup_{i \geq 1} (yield(SET(R_i(t))).x_i)$. For instance, the rule of R for the symbol $c_{2,k}$ and state 0 might look like

$$R_0(c_{2,k}(t\ t_1\ t_2)) =$$



The detailed construction of R is left to the reader.

□

Finally we consider the OI hierarchy for an arbitrary nondeterministic monadic algebra, i.e. a domain with a finite number of nondeterministic unary operations. Let Ω be an alphabet, let D be a set and let, for each $f \in \Omega$, f_D be a mapping $D \to \mathcal{P}(D)$, or equivalently a subset of $D \times D$. Consider first the $\bigsqcup$-continuous Ω^c-algebra A with domain $\mathcal{P}(D \times D)$, such that for $f \in \Omega$ $f_A = f_D$, e_A is the identity mapping, i.e. $e_A = \{(d,d) \mid d \in D\}$, and c_A is composition of binary relations. Since there is a (unique) $\bigsqcup$-continuous Ω^c-homomorphism $\mathcal{P}(\Omega_c^*) \to \mathcal{P}(D \times D)$

(see [11]), the equational (OI equational) elements of $\mathcal{P}(D \times D)$ are the Ω^c-homomorphic images of the context-free languages (OI string languages) over Ω. It was shown in [11 , sections 4 and 5] that these are the relations computed by the nondeterministic recursive monadic programschemes in D (the relations computed by the socalled nondeterministic procedure parameter schemes in D respectively). Consider now the nondeterministic Ω^m-algebra D with domain D , nondeterministic monadic operations f_D and some "input element" $e_D \in D$. It follows from Theorem 7.15 and Corollary 7.12 (adapted in the obvious way to the case of a nondeterministic algebra) that, for $n = 0$, 1 and 2 , the OI(n) equational subsets of D are the Ω^m-homomorphic images of the regular, context-free and OI string languages in Ω_m^* , respectively. From a comparison of the Ω^m-homomorphism $\mathcal{P}(\Omega_m^*) \to \mathcal{P}(D)$ with the Ω^c-homomorphism $\mathcal{P}(\Omega_c^*) \to \mathcal{P}(D \times D)$ it easily follows that these equational subsets are the images of e_D under the Ω^c-homomorphic images of the regular, context-free and OI string languages, i.e. they are the sets computed from the input e_D by the nondeterministic Ianov schemes, the nondeterministic recursive schemes and the nondeterministic procedure parameter schemes respectively (see [11, sections 3, 4 and

5]). This shows that these three classes of program schemes correspond in a natural algebraic way to the hierarchy of regular, context-free and OI string languages.

Conclusion.

We have shown that the fixed point approach to formal language theory and the theory of programs can be used to explain the differences between IO (call by value) and OI (call by name) computation. Moreover, in the framework of continuous algebras, we gained a better insight into the various more or less well known Mezei and Wright like results in this area.

The fixed point characterization of context-free tree languages implied that IO tree languages are YIELDs of recognizable second level tree languages, and that OI tree languages are SET∘YIELDs of recognizable second level infinite trees (with +). It might be interesting to prove more results about IO and OI tree languages using either the fixed point characterization itself (see for instance section 4 and [8]) or its implications (see for instance section 6). To treat the OI case it might be advantageous to develop a theory of infinite trees (see [7, 10, 15]).

Not much is known about the hierarchies defined in section 7. We conjecture that all the IO tree language classes in the hierarchy are closed under deterministic bottom-up tree transducers and that all the

OI tree language classes are closed under linear (top-down or bottom-up) tree transducers. For OI, together with the obvious closure under OI substitution, a result in [2 , Theorem 3.2.12] would then imply that all the OI string language classes are (substitution-closed) full AFLs. Other questions concerning the hierarchies are: what is the relationship to the tree transducer hierarchy ([2 , 24])? what "nondeterministic power" is present in the IO hierarchy, and what "copying power" in the OI hierarchy?

As regards programscheme theory, the remarks in section 5, together with the incomparability of the classes of IO and OI tree languages, show that our classes of nondeterministic IO (call by value) and OI (call by name) recursive program schemes (without tests) are incomparable with respect to program scheme equivalence. About deterministic recursive program schemes with tests not very much is known. It is easy to see that every deterministic call by value scheme is equivalent to a deterministic call by name scheme, which is forced to evaluate its arguments by some trivial test (provided these are present). It can also be proved (see [26]) that every deterministic call by name scheme is equivalent to a nondeterministic call by value scheme (which guesses which of its arguments it actually will need). The precise relationship between the classes of deterministic call by value and call by name schemes remains open. It is also an open question whether a useful "universal interpretation" exists for deterministic schemes with tests.

Hopefully this paper will be of some help in the solution of these problems.

References.

1. E. Ashcroft, Z. Manna and A. Pnueli, Decidable properties of monadic functional schemas, JACM 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. of Combinatorial Theory 8 (1970), 115-133.

5. A. Blikle, Equational languages, Inf. and Control 12 (1972), 134-147.

6. P.M. Cohn, "Universal algebra", Harper and Row, New York, 1965.

7. B. Courcelle, Recursive schemes, algebraic trees and deterministic languages, 15th Annual Symposium on Switching & Automata Theory, 1974, 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. Syst. Theory 9 (1975), 198-231.

10. J. Engelfriet, A note on infinite trees, Inf. Proc. Letters 1 (1972), 229-232.

11. J. Engelfriet, Simple program schemes and formal languages, Lecture Notes in Computer Science 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, JCSS 7 (1973), 119-160.

14. S. Ginsburg and H. G. Rice, Two families of languages related to ALGOL, JACM 9 (1962), 350-371.

15. J. A. Goguen and J. W. Thatcher, Initial algebra semantics, 15th Annual Symposium on Switching and Automata Theory, 1974, 63-77.

16. J. A. Goguen, J. W. Thatcher, E. G. Wagner and J. B. Wright, Initial algebra semantics, JACM 24 (1977), 68-95.

17. T. S. E. Maibaum, A generalized approach to formal languages, JCSS 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, CACM 16 (1973), 491-502.

20. J. McCarthy, A basis for a mathematical theory of computation, in: Computer Programming and Formal Systems (eds. P. Braffort and D. Hirschberg), North-Holland Publ. Co., Amsterdam, 1963, pp. 33-70.

21. J. Mezei and J. B. Wright, Algebraic automata and context-free sets, Inf. and Control 11 (1967), 3-29.

22. M. Nivat, Langages algébriques sur le magma libre et sémantique des schémas de programme, in: Automata, Languages and Programming (ed. M. Nivat), North-Holland Publ. Co., Amsterdam, 1973, pp. 293-307.

23. M. Nivat, On the interpretation of recursive program schemes, Symposia Matematica, Vol. 15 (1975), 255-281.

24. W. F. Ogden and W. C. Rounds, Composition of n tree transducers, 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 (ed. J. Loeckx), Lecture Notes in Computer Science 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, CACM 7 (1964), 52-71.

28. B. K. Rosen, Tree-manipulating systems and Church-Rosser theorems, JACM 20 (1973), 160-188.

29. W. C. Rounds, Mappings and grammars on trees, Math. Syst. Theory 4 (1970), 257-287.

30. W. C. Rounds, Tree-oriented proofs of some theorems on context-free and indexed languages, Second Annual Symposium on Theory of Computing, Northampton, Mass., 1970, pp. 109-116.

31. D. Scott, Data types as lattices, SIAM J. Comp. 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, JCSS 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 (ed. A. V. Aho), Prentice-Hall, 1973, pp. 143-172.

36. J. W. Thatcher and J. B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic, Math. Syst. Theory 2 (1968), 57-81.

37. R. Turner, An infinite hierarchy of term languages - an approach to mathematical complexity, in: Automata, Languages and Programming (ed. M. Nivat), North-Holland Publ. Co., Amsterdam, 1973, pp. 593-608.

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, 12th Annual Symposium on Switching and Automata Theory, 1971, 192-201.

40. M. Wand, A concrete approach to abstract recursive definitions, in: Automata, Languages and Programming (ed. M. Nivat), North-Holland Publ. Co., Amsterdam, 1972, pp. 331-344.

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, Lecture Notes in Computer Science 25, Springer-Verlag, Berlin, 1975, pp. 209-213.

43. G. Boudol, Thèse de 3ème cycle, Paris VII.

44. W. Damm, Higher type program schemes and their tree languages, 4th Colloquium on Automata, Languages and Programming, Turku, 1977.