LANGUAGES DEFINED BY HIGHER TYPE PROGRAM SCHEMES

Werner Damm
Lehrstuhl für Informatik II
RWTH Aachen

1. Introduction

A recursive program scheme in the sense of Nivat [10] abstracts the control structure of a certain class of recursive procedures in order to investigate their common properties. These procedures take only data elements as parameters and do not allow procedures as parameters. In contrast, higher type procedures as definable in ALGOL 68 have the characteristic feature, that their value as well as their parameters can be of type procedure, procedure procedure, etc. . In this paper, we investigate such higher type procedures. We can prove that the auxiliary use of recursion on higher functional domains increases the computational power of a programming language. To this purpose, we introduce a family $\langle R_n(\Sigma) \mid n \in \mathbb{N} \rangle$ of program scheme classes with recursion on functional level n , Σ denoting the set of given base function symbols.

As with a recursive program scheme [10], we can associate with a higher type scheme a tree grammar. Depending on the order of the derivation steps - innermost outermost or outermost innermost, see [5], [6] - we define two classes of tree languages, $\mathcal{T}^n_{IO}(\Sigma)$ and $\mathcal{T}^n_{OI}(\Sigma)$, generated by level n higher type scheme which generalize the classes of IO- and OI-context free tree languages. The semantics of a level n higher type scheme can be characterized by a level n IO-tree language. This allows to compare the power of higher type schemes by comparing their IO-tree languages. To this end, we introduce classes of string languages $\mathcal{L}^n_{IO}(V)$ over an alphabet V and prove that the set of branches of a tree language in $\mathcal{T}^n_{IO}(\Sigma)$ is a level n IO-string language; in fact, any such string language can be obtained this way. We define level n languages L_n , such that at least $L_1 \notin \mathcal{L}^0_{IO}(V)$ and $L_2 \notin \mathcal{L}^1_{IO}(V)$. This proves all discussed hierarchies strict in the first two steps. In particular, $R_2(\Sigma)$ is more powerful than the class of recursive program schemes over Σ .

The formalization of the topic in the setting of deterministic program schemes was suggested by K. Indermark . In [7], he introduces typed combinator schemes, which include fixpoint combinator schemes at any functional level. Then he shows how to reduce the subclass, which, when interpreted, define functions of type 1, to certain standard forms. By a normal form theorem of Wand [11], these coincide with the classes $R_n(\Sigma)$ introduced in this paper.

Engelfriet and Schmidt define in [5] the corresponding nondeterministic hierarchy of schemes, and prove that the regular, context-free and macro string languages may be obtained as solutions over a particular interpretation at level 0 , 1 and 2 , respectively.

In [12], Wand indicates a proof of a similar result. He uses a categorical generalization of his concept of a μ-clone of an algebra [11], to define a hierarchy of string languages, which starts with the regular, context-free and indexed languages.

I want to thank Klaus Indermark for many stimulating discussions, and Bruno Courcelle for pointing out the necessity to start with reduced grammars in the proof of theorem 3.21 .

2. Algebraic background

An algebraic definition of the semantics of higher type schemes requires many-sorted continuous algebras as interpretations. In this section, we will shortly review the basic definitions. The reader is refered to [1] for details.

Let I be a set of sorts, and I^* be the set of strings over I . For $w \in I^*$, $l(w)$ denotes the length of w . If $l(w) = n > 0$, we write $w = w(1)\dots w(n)$. The empty string will be denoted by e .

An I-sorted alphabet is a family $\Sigma = \langle \Sigma^{\langle w,i\rangle} \rangle_{\langle w,i\rangle \in I^* \times I}$. The elements of $\Sigma^{\langle w,i\rangle}$ are called base function symbols of arity $\langle w,i\rangle$. If $X = \langle X^i \rangle_{i \in I}$ is a family with domain I , then $\Sigma(X)$ is the I-sorted alphabet defined by $\langle e,i\rangle \mapsto \Sigma^{\langle e,i\rangle} \cup X^i$, $w \neq e \Rightarrow \langle w,i\rangle \mapsto \Sigma^{\langle w,i\rangle}$.

A Σ-algebra is a pair $\underset{\sim}{A} = (A, \varphi_A)$, where A is a family of sets $\langle A^i \rangle_{i \in I}$, and φ_A assigns to each $f \in \Sigma^{\langle w,i\rangle}$ a function over A of correct arity, i.e. $\varphi_A(f) : A^w \to A^i$. Here, A^w denotes the generalization of the cartesian product defined by $A^e := \{\bot\}$ - $\bot$ is a new symbol - , and $A^{wi} := A^w \times A^i$.

A family of mappings $h = \langle h^i : A^i \to B^i \rangle_{i \in I}$ between carriers of Σ-algebras $\underset{\sim}{A}$ and $\underset{\sim}{B}$ is a Σ-homomorphism iff

$\forall f \in \Sigma^{<w,i>} h^i(\varphi_A(f)(a_1,..,a_n)) = \varphi_B(f)(h^{w(1)}(a_1),..,h^{w(n)}(a_n))$. If h is a homomorphism, then we write $h : \underset{\sim}{A} \to \underset{\sim}{B}$.

A Σ-algebra $\underset{\sim}{A}$ is <u>continuous</u> iff

(1) each A^i is a partially ordered set with minimal element $\perp_{A^i}$,

(2) any directed set $D \subseteq A^i$ has a least upper bound $\bigsqcup D \in A^i$,

(3) all operations $\varphi_A(f)$ are continuous, i.e. $\varphi_A(f)$ is monotone and for any directed $D_j \subseteq A^{w(j)}$ $\varphi_A(f)(\bigsqcup D_1,..,\bigsqcup D_n) = \bigsqcup \varphi_A(f)(D_1,..,D_n)$.

Let Δ-$\underset{\sim}{\text{alg}}_\Sigma$ denote the class of continuous Σ-algebras. This class contains an initial object, which will be denoted by $\underset{\sim}{CT}_\Sigma$. Intuitively, we can view $t \in CT^i_\Sigma$ as an infinite Σ-tree, where in addition minimum symbols $\perp_s$ may occur as leaves.

<u>2.1 theorem (ADJ [1])</u>

For all $\underset{\sim}{A} \in \Delta$-$\underset{\sim}{\text{alg}}_\Sigma$ there exists a unique $\perp$-preserving continuous Σ-homomorphism $h_{\underset{\sim}{A}} : \underset{\sim}{CT}_\Sigma \to \underset{\sim}{A}$.

We denote by $\underset{\sim}{FT}_\Sigma$ the restriction of $\underset{\sim}{CT}_\Sigma$ to finite trees, and by $\underset{\sim}{T}_\Sigma$ the usual Σ-tree algebra.

Let, for $w \in I^*$, $Y_w = \{y_{1,w(1)},..,y_{n,w(n)}\}$ be a set of parameters $(Y_e := \emptyset)$. Y_w can be considered as a family with domain I by setting $Y^i_w := \{y_{j,w(j)} | \, w(j) = i\}$. Because of the above theorem, an infinite tree $t \in CT^i_{\Sigma(Y_w)}$ induces a <u>derived operation</u> over $\underset{\sim}{A} \in \Delta$-$\underset{\sim}{\text{alg}}_\Sigma$:

$$\underline{\text{derop}_{\underset{\sim}{A}}}(t) : A^w \to A^i$$

is given by $a = (a_1,..,a_n) \mapsto h_{\underset{\frown}{A(a)}}(t)$, where $\underset{\frown}{A(a)}$ is the $\Sigma(Y_w)$-algebra obtained from $\underset{\sim}{A}$ by letting $y_{j,w(j)}$ name a_j .

In case $\underset{\sim}{A} = \underset{\sim}{CT}_{\Sigma(Y_v)}$, we write $\underline{\text{sub}}_{Y_w}(t)$ for the derived operation of t over $\underset{\sim}{A}$, because $\underline{\text{sub}}_{Y_w}(t)(t_1,..,t_n)$ is the infinite tree obtained from t by substituting t_j for $y_{j,w(j)}$.

<u>3. Higher type schemes</u>

Higher type schemes will be defined as regular schemes over partly interpreted alphabets. A regular scheme over Σ consists of a finite deterministic system of regular equations with parameters ([1], [9]).

3.1 definition (regular scheme over Σ , $R(\Sigma)$)

The set $R(\Sigma)^{<w,i>}$ of regular schemes over Σ of sort $<w,i> \in I^* \times I$ is the set of all mappings $S : X_v \longrightarrow FT_{\Sigma(Y_w)}(X_v)$ for some $v \in I^*$, such that $v(1) = i$ and $\forall\, 1 \leq j \leq l(v)$ $S(x_{j,v(j)}) \in FT_{\Sigma(Y_w)}(X_v)^{v(j)}$. □

Such a scheme will be written as $l(v)$ equations $x_{j,v(j)} = S(x_{j,v(j)})$. The elements of X_v are called variables, those of Y_w parameters. Note, that the first equation is viewed as the defining equation.

example 1

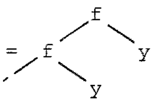
In this and the following examples, Σ will always denote an $\{i\}$-sorted alphabet with $e \in \Sigma^{<e,i>}$, $a , g , h \in \Sigma^{<i,i>}$, and $f \in \Sigma^{<ii,i>}$.

Let $S_0 \in R(\Sigma)^{<i,i>}$ be the scheme $x = f(x,y)$, where we abbreviate x for $x_{1,i}$ and y for $y_{1,i}$.

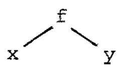
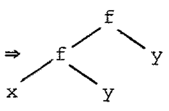
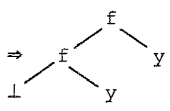
The interpretation of S_0 in a continuous Σ-algebra $\underset{\sim}{A}$ is the monadic function $[\![S_0 , \underset{\sim}{A}]\!] : A \to A$, which maps $a \in A$ onto the least fixpoint of the function $A \to A$ obtained from S_0 by substituting a for the parameter y .

$$[\![S_0 , \underset{\sim}{A}]\!] : A \longrightarrow A$$
$$a \longmapsto Y(\lambda b .\ \varphi_A(f)(b , a))$$

The infinite tree of S_0 is $T(S_0) := [\![S_0 , \underset{\sim}{CT}_\Sigma(Y_i)]\!](y)$.

It is easy to see, that $T(S_0) =$  .

In order to use S_0 as a regular tree grammar, we have to add $x = \bot$ as a production. An example of a derivation is

$x \Rightarrow$  $\Rightarrow$  $\Rightarrow$  $\in FT_\Sigma(Y_i)$.

□

We now define these concepts formally. Let $\underset{\sim}{A} \in \Delta\text{-}\underline{alg}_\Sigma$, and $S : X_v \longrightarrow FT_{\Sigma(Y_w)}(X_v)$, $S \in R(\Sigma)^{<w,i>}$, $l(v) = m$, $l(w) = n$.

3.2 definition (interpretation of S in $\underset{\sim}{A}$, $[\![S , \underset{\sim}{A}]\!]$)

$[\![S , \underset{\sim}{A}]\!] : A^w \longrightarrow A^i$ is defined by

$$a = (a_1,..,a_n) \mapsto \underline{pr}_1^{(m)}(Y(\underline{derop}_{\underset{\sim}{A(a)}}(S(x_{1,v(1)})) ; .. ; \underline{derop}_{\underset{\sim}{A(a)}}(S(x_{m,v(m)}))))$$ □

<u>3.3 definition</u> (infinite tree of S, $T(S)$)

$$T(S) := [\![S , \underset{\sim}{CT}_\Sigma(Y_w)]\!](y_{1,w(1)},\dots,y_{n,w(n)})$$ □

<u>3.4 definition</u> (language generated by S, $L(S)$)

Let $\tau_1, \tau_2 \in FT_{\Sigma(Y_w)}(X_v)$.

τ_2 derives directly from τ_1 in S ($\tau_1 \underset{S}{\Rightarrow} \tau_2$) iff τ_2 is obtained from τ_1 by substituting exactly one occurence of $x_{j,v(j)}$, some $j \in \{1,..,l(v)\}$, by $S(x_{j,v(j)})$ or by $\perp_{v(j)}$.

Let $\overset{*}{\underset{S}{\Rightarrow}}$ denote the transitive reflexive closure of $\underset{S}{\Rightarrow}$.
The language generated by S is

$$L(S) := \{t \in FT_\Sigma(Y_w)^i \mid x_{1,v(1)} \overset{*}{\underset{S}{\Rightarrow}} t\} .$$ □

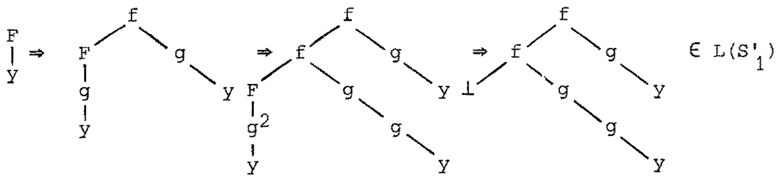
Though regular schemes are a very simple class of schemes, they are powerful enough to model higher type procedures. We will first demonstrate, how recursive schemes can be retrieved in this setting.

<u>example 2</u>

Consider the <u>r</u>ecursive <u>p</u>rogram <u>s</u>cheme $S'_1 \in \underline{rps}(\Sigma)^{<i,i>}$ given by

$$F(y) = f(F(g(y)), g(y)) .$$

Such a scheme can be used as a context-free tree grammar by adding $F(y) = \perp$ as a production. An example of a derivation is



$\in L(S'_1)$.

Assume, that we introduce a new symbol $\$$ which is always interpreted as functional substitution. Then we can rewrite the right hand side of S'_1 to $\$(f, \$(F, g), g)$: substitute into f the composition of F and g , and g . We express this relation by

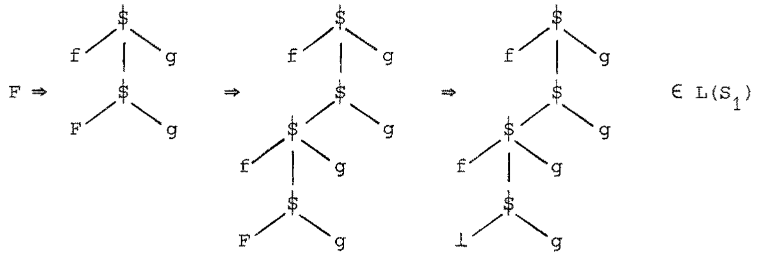
$$\underline{yield}(\$(f, \$(F, g), g) = f(F(g(y)), g(y)) .$$

Now consider the scheme S_1 given by

$$F = \$(f, \$(F, g), g) ,$$

then S_1 is a regular scheme, but over a new alphabet, called derived alphabet of Σ . This alphabet will be denoted $D(\Sigma)$; it contains the base functions in Σ as constants and in addition projection and substitution symbols.

By adding $F = \perp$ as a production, we can use S_1 as a regular tree grammar over $D(\Sigma)$:



Note, that the yield of the last tree is $f(f(\perp , g(g(y))) , g(y))$.

This suggests that we can obtain $L(S'_1)$ by using S_1 as a regular tree grammar and then translating the $D(\Sigma)$-trees in $L(S_1)$ to the Σ-level.

As the right hand side of S_1 is in $FT_{D(\Sigma)}(X)$, we have to interpret S_1 in $D(\Sigma)$-algebras. Let $D(\underset{\sim}{A})$ be the $D(\Sigma)$-algebra, which has continuous functions over A as carrier, the base functions of $\underset{\sim}{A}$ and projections as constants, and functional substitution as operations. Then $[\![S_1 , D(\underset{\sim}{A})]\!] = [\![S'_1 , \underset{\sim}{A}]\!]$ □

The above example has introduced informally all the concepts necessary to treat higher type schemes. In the formal definitions, the projection and substitution symbols will carry type informations, because we want to iterate the constructions indicated in the example.

<u>3.5 definition</u> (derived index set, $D(I)$, derived alphabet, $D(\Sigma)$)

The derived index set of I is $D(I) := I^* \times I$.

The derived alphabet of Σ , $D(\Sigma)$, is the $D(I)$-sorted alphabet, which contains

(i) $f \in \Sigma^{<w,i>}$ as constant of sort $<e , <w,i>>$

(ii) π^w_j with $l(w) > 0$, $1 \leqslant j \leqslant l(w)$, $w \in I^*$,
as projection symbol of sort $<e , <w,w(j)>>$

(iii) $\$^v_{<w,i>}$ with $v , w \in I^*$, $i \in I$, $l(v) = m$,
as substitution symbol of sort $<<e,i> , <w,i>>$ if $v = e$,
$<<v,i><w,v(1)>\dots<w,v(m)> , <w,i>>$ otherwise. □

<u>3.6 definition</u> (derived algebra of $\underset{\sim}{A}$, $D(\underset{\sim}{A})$)

The derived algebra of $\underset{\sim}{A}$ is the $D(\Sigma)$-algebra $(D(A) , \varphi_{D(A)})$, where

(1) the carrier of sort $<w,i>$, $D(A)^{<w,i>}$, is the set of all continuous functions from A^w to A^i and

(2) $\varphi_{D(A)}$ assigns operations to the symbols in $D(\Sigma)$ by

(i) $f \in \Sigma^{<w,i>}$	denotes	$\underline{\perp}$	$\mapsto \varphi_A(f)$
(ii) π_j^w	denotes	$\underline{\perp}$	$\mapsto \lambda(a_1,..,a_n)\ a_j$
(iii) $\$^e_{<w,i>}$	denotes	f	$\mapsto \lambda(a_1,..,a_n)\ f(\underline{\perp})$
$\$^v_{<w,i>}$	denotes	$(f, g_1,..,g_m)$	$\mapsto f \circ (g_1;..;g_m)$

□

It is easy to see, that $D(\underset{\sim}{A}) \in \Delta\text{-}\underline{alg}_{D(\Sigma)}$.

The translation <u>yield</u> is actually a $D(\Sigma)$-homomorphism, obtained by defining a $D(\Sigma)$-structure on $CT_\Sigma(Y)$ (see [8]) .

<u>3.7 definition</u> (<u>yield</u>)

We define a $D(\Sigma)$-structure on $CT_\Sigma(Y)$ by

(1) the carrier of sort $<w,i>$ is $CT_\Sigma(Y_w)^i$

(2) the assignment function is given by

(i) $f \in \Sigma^{<w,i>}$	denotes	$\underline{\perp} \mapsto f(y_{1,w(1)},..,y_{n,w(n)})$
(ii) π_j^w	denotes	$\underline{\perp} \mapsto y_{j,w(j)}$
(iii) $\$^e_{<w,i>}$	denotes	$t \mapsto t$
$\$^v_{<w,i>}$	denotes	$\underline{sub}_{Y_v}$

The by theorem 2.1 unique homomorphism from $\underset{\sim}{CT}_{D(\Sigma)}$ into this algebra will be denoted <u>yield</u> .

□

Until now, we have indicated, how recursive schemes can be redefined as certain regular schemes over $D(\Sigma)$. We will now iterate the ideas of the previous example to demonstrate that higher type schemes allow to define <u>new</u> objects.

<u>example 3</u>

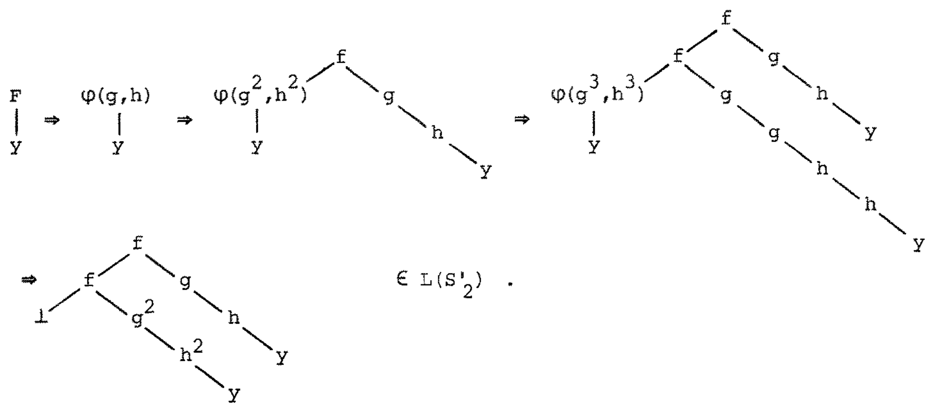
Let S'_2 be the higher type scheme

$$\longrightarrow F(y) = \varphi(g,h)(y)$$

$$\varphi(F_1,F_2)(y) = f(\varphi(F_1 \circ g , F_2 \circ h)(y) , F_1(F_2(y))) .$$

Here, we define recursively a procedure φ , which, when applied to actual parameters of type <u>procedure</u>, delivers a procedure as value. φ is called from the main procedure F with two base functions as actual parameters, thus this scheme computes a monadic function over an interpretation $\underset{\sim}{A}$.

In order to use S'_2 as a tree grammar, we add productions $F(y) = \perp$, $\varphi(F_1 , F_2)(y) = \perp$. An example of a derivation is



$\in L(S'_2)$.

□

Note that the set of branches of $L(S'_2)$ is $\{f^n g^n h^n \mid n \in \mathbb{N}\}$ and thus not context-free. This will be used to prove that S'_2 is not equivalent to any recursive program scheme, by establishing a connection between the semantics of higher type schemes and their tree languages.

The notation employed to define S'_2 may be viewed as an informal way of defining regular schemes over the derived alphabet of $D(\Sigma)$, $D^2(\Sigma)$, corresponding to the fact that a variable of functional level 2 is recursively defined. Let, for $M \in \{I, \Sigma, \underset{\sim}{A}\}$, $D^0(M) := M$, and $D^{n+1}(M) := D(D^n(M))$. In general, recursion on level n can be formalized by using regular schemes over $D^n(\Sigma)$.

Consider the class $R(D^n(\Sigma))$. Among these schemes, there are some, which are of sort $\langle e,\langle\ldots,\langle e,\langle w,i\rangle\rangle\ldots\rangle\rangle$, and thus, after successive applications to $\perp$, define a function of sort $\langle w,i\rangle \in I^* \times I$.

Let $\underline{b}_n : I^* \times I \longrightarrow D^n(I)$ be the mapping given by $\underline{b}_0\langle w,i\rangle := i$, $\underline{b}_1\langle w,i\rangle := \langle w,i\rangle$, $\underline{b}_{n+1}\langle w,i\rangle := \langle e, \underline{b}_n\langle w,i\rangle\rangle$. We will formally define higher type schemes as those regular schemes over $D^n(\Sigma)$, which are of sort $\underline{b}_{n+1}\langle w,i\rangle$ for some $\langle w,i\rangle \in I^* \times I$.

<u>3.8 definition</u> (level n higher type scheme, $R_n(\Sigma)$)

The class $R_n(\Sigma)^{\langle w,i\rangle}$ of level n higher type schemes of sort $\langle w,i\rangle \in I^* \times I$ is defined by

$$R_n(\Sigma)^{\langle w,i\rangle} := R(D^n(\Sigma))^{\underline{b}_{n+1}\langle w,i\rangle} .$$

$R_n(\Sigma)$ is the family $\langle R_n(\Sigma)^{\langle w,i\rangle}\rangle_{\langle w,i\rangle \in I^* \times I}$.

□

When comparing the semantics of higher type schemes, we do not distinguish between an element $f \in D^{n+1}(A)^{\underline{b}_{n+1}\langle w,i\rangle}$ and the function $f\underbrace{(\perp)\ \ldots\ (\perp)}_{n} : A^w \longrightarrow A^i$. Moreover, we will only compare schemes over those interpretations, which respect the intended meaning of all substitution and projection symbols.

3.9 definition (equivalence , $\sim$, translatability , $\leqslant$)

Let $S \in R_n(\Sigma)$, $S' \in R_m(\Sigma)$, and $\mathcal{S}$, $\mathcal{S}'$ be classes of schemes over Σ .
S is equivalent to S' $(S \sim S')$ iff $\forall \underset{\sim}{A} \in \Delta\text{-}\underline{\text{alg}}_\Sigma$ $[\![S , D^n(\underset{\sim}{A})]\!] = [\![S' , D^m(\underset{\sim}{A})]\!]$
$\mathcal{S}$ is translatable into $\mathcal{S}'(\mathcal{S} \leqslant \mathcal{S}')$ iff $\forall s \in \mathcal{S} \exists s' \in \mathcal{S}'$ $s \sim s'$
$\mathcal{S}$ is equivalent to $\mathcal{S}'(\mathcal{S} \sim \mathcal{S}')$ iff $\mathcal{S} \leqslant \mathcal{S}' \wedge \mathcal{S}' \leqslant \mathcal{S}$ □

Using the informal notation for higher type schemes, it is obvious that any scheme in $R_n(\Sigma)$ is equivalent to a scheme in $R_{n+1}(\Sigma)$.

3.10 corollary

$$R_n(\Sigma) \leqslant R_{n+1}(\Sigma)$$

By definition, this hierarchy starts with the regular schemes over Σ . We will prove, that level 1 higher type schemes are exactly as powerful as recursive program schemes.

3.11 theorem

$$\underline{\text{rps}}(\Sigma) \sim R_1(\Sigma)$$

proof:

Let $S \in R_1(\Sigma)^{<w,i>}$. We define $\underline{\text{yield}}(S) \in \underline{\text{rps}}(\Sigma)^{<w,i>}$ by taking the $\underline{\text{yield}}$ of all right hand sides of S and replacing variables by function variables. To prove $S \sim \underline{\text{yield}}(S)$, is suffices to show, that

$$\begin{tikzcd} CT_{D(\Sigma)} \arrow[rr, "\underline{\text{yield}}"] \arrow[dr, "h_{D(\underset{\sim}{A})}"'] & & CT_\Sigma(Y) \arrow[dl, "\underline{\text{derop}}_{\underset{\sim}{A}}"] \\ & D(A) & \end{tikzcd}$$

commutes.

But this follows from theorem 2.1 , because $\underline{\text{derop}}_{\underset{\sim}{A}}$ is a continuous $D(\Sigma)$-homomorphism. As $\underline{\text{yield}}$ is onto, we have also proved $\underline{\text{rps}}(\Sigma) \leqslant R_1(\Sigma)$. □

In [5], Engelfriet and Schmidt define IO(n) and OI(n) equational elements of a (completely continuous) subset algebra $P(\underset{\sim}{A}) \in \sqcup\text{-}\underline{\text{alg}}_\Sigma$. Let $NR_n(\Sigma)$ denote level n higher type schemes which have finitely many terms in $T_{D^n(\Sigma)}(X)$ as one right hand side, then the OI(n) equational elements are solutions of constant schemes in $NR_n(\Sigma)$. In the OI-case, we can simulate nondeterminism deterministically. Let Σ^+ be obtained from Σ by adding symbols $+_i$ of sort $<i,i>$, which in completely continuous algebras are interpreted as join operation.

3.12 lemma

$$NR_n(\Sigma) \sim_{\sqcup\text{-}\underline{\text{alg}}_\Sigma} R_n(\Sigma^+)$$

In the IO-case, this simulation is not possible.

Now recall definition 3.3 of the infinite tree of a regular scheme: if $S \in R_n(\Sigma)^{<w,i>}$, then $T(S) \in CT^{\underline{b}_n<w,i>}_{D^n(\Sigma)}$. Much as we defined yield, we obtain the existence of a mapping

$$\underline{\text{yield}}^{(n)} : CT^{\underline{b}_n<w,i>}_{D^n(\Sigma)} \longrightarrow CT_\Sigma(Y_w)^i \quad ,$$

which translates $T(S)$ to an infinite $\Sigma(Y_w)$-tree by taking into account the meaning of all the projection and substitution symbols. In fact, the derived operation of this tree over $\underset{\sim}{A}$ is precisely the solution of S in $\underset{\sim}{A}$.

<u>3.13 theorem</u>

$$\underline{\text{derop}}_{\underset{\sim}{A}}(\underline{\text{yield}}^{(n)}(T(S))) = [\![S \, , D^n(\underset{\sim}{A})]\!]$$

proof:

By the Mezei-Wright-like theorem for regular schemes, [1], $h_{D^n(A)}(T(S)) = [\![S \, , D^n(\underset{\sim}{A})]\!]$.
As in the proof of 3.11, we show by induction on $n \in \mathbb{N}$, that

$$\begin{tikzcd}
CT^{\underline{b}_n<w,i>}_{D^n(\Sigma)} \arrow[rrr, "\underline{\text{yield}}^{(n)}"] \arrow[dr, "h_{D^n(\underset{\sim}{A})}"'] & & & CT_\Sigma(Y_w)^i \arrow[dl, "\underline{\text{derop}}_{\underset{\sim}{A}}"] \\
& D^n(A)^{\underline{b}_n<w,i>} \arrow[r, phantom, "\supseteq"] & D(A)^{<w,i>} &
\end{tikzcd}$$

commutes. □

By the above theorem,

$$\underline{\text{yield}}^{(n)}(T(S)) = [\![S \, , D^n(\underset{\sim}{CT}_\Sigma(Y_w))]\!](y_{1,w(1)},\dots,y_{k,w(k)}) \; ,$$

thus this tree characterizes the equivalence class of S .

<u>3.14 corollary</u>

$S \sim S'$ iff $[\![S \, , D^n(\underset{\sim}{CT}_\Sigma(Y_w))]\!] = [\![S' \, , D^m(\underset{\sim}{CT}_\Sigma(Y_w))]\!]$

iff $\underline{\text{yield}}^{(n)}(T(S)) = \underline{\text{yield}}^{(m)}(T(S'))$

Next, we want to characterize equivalence by tree languages generated by schemes. Much as with context-free tree grammars, we can associate an IO- and an OI-tree language with a higher type scheme.

<u>3.15 definition</u> ($L_{IO}(S)$, level n IO-tree language , $\mathcal{T}^n_{IO}(\Sigma)$)

Let $S \in NR_n(\Sigma)$. The IO-tree language generated by S is defined by

$$L_{IO}(S) := \underline{\text{yield}}^{(n)}(L(S)) \; .$$

The class of level n IO-tree languages of sort $<w,i>$ is

$\mathcal{T}^n_{IO}(\Sigma)^{<w,i>} := \{ L \subseteq T_\Sigma(Y_w)^i \mid L = L_{IO}(S) \wedge S \in NR_n(\Sigma)^{<w,i>} \}$ □

<u>3.16 definition</u> ($L_{OI}(S)$, level n OI-tree language , $\mathcal{T}^n_{OI}(\Sigma)$)

Let $S \in R_n(\Sigma^+)^{<w,i>}$. The OI-tree language generated by S is defined by

$$L_{OI}(S) := [\![S , D^n(\mathcal{P}(\underset{\sim}{T}_\Sigma(Y_w)))]\!](y_{1,w(1)},\ldots,y_{k,w(k)}) \ .$$

The class of level n OI-tree languages of sort <w,i> is

$$\mathcal{T}^n_{OI}(\Sigma)^{<w,i>} := \{L \subseteq T_\Sigma(Y_w)^i \mid L = L_{OI}(S) \wedge S \in R_n(\Sigma^+)^{<w,i>}\} \qquad \square$$

The tree languages defined in examples 1 , 2 , and 3 are both IO- and OI-tree languages of level 0 , 1 , and 2 , respectively.
It is easy to see, that the IO- and OI-tree languages form a hierarchy with increasing n , which starts with the regular tree languages. The fixpoint characterizations of context-free tree languages proved by Engelfriet and Schmidt [5] imply, that $\mathcal{T}^1_{IO}(\Sigma)$ and $\mathcal{T}^1_{OI}(\Sigma)$ contain precisely the IO- and OI-context-free tree languages, respectively.

In [3,4] we show, how to construct a normalform <u>nf</u> of $S \in R_n(\Sigma)$, such that the IO-tree language of <u>nf</u>(S) contains all finite approximations of T(S) .

<u>3.17 theorem</u>

$S \sim S'$ iff $L_{IO}(\underline{nf}(S)) = L_{IO}(\underline{nf}(S'))$

This result allows to compare the semantics of higher type schemes by comparing their IO-tree languages. To this end, we will characterize the set of branches of level n IO-tree languages as certain string languages. Again, two definitions are possible.

Let V be a string alphabet. To V we associate an {i}-sorted alphabet Σ_V containing V as monadic symbols and a constant symbol e . Then $\mathcal{P}(V^*)$ is a Σ_V-algebra with $v \in V$ denoting left concatenation with v , and e denoting the set consisting of the empty word. As $\mathcal{P}(\underset{\sim}{V}^*)$ is isomorphic to $\mathcal{P}(\underset{\sim}{T}_{\Sigma_V})$, we can define string languages over V by specifying tree languages over Σ_V .

<u>3.18 definition</u> (level n IO-string languages, $\mathcal{L}^n_{IO}(V)$, level n OI-string languages, $\mathcal{L}^n_{OI}(V)$)

$$\mathcal{L}^n_{IO}(V) := \mathcal{T}^n_{IO}(\Sigma_V)^{<e,i>}, \ \mathcal{L}^n_{OI}(V) := \mathcal{T}^n_{OI}(\Sigma_V)^{<e,i>} \qquad \square$$

In the following examples, we will use the informal notation for higher type schemes. Types of variables are indicated by $F : \underline{n}$, where $\underline{0} : i$ and $\underline{n+1} := <\underline{n} , \underline{n}>$. Let $a \in V$.

<u>example 4</u> $x : \underline{0}$, $F : \underline{1}$, $\varphi : \underline{2}$

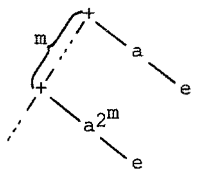
Let $S_2 \in NR_2(\Sigma_V)^{<e,i>}$ be the scheme

$$\longrightarrow x = \varphi(a)(e)$$
$$\varphi(F)(y) = \varphi(F \circ F)(y) \ , \ F(y) \ .$$

Then $L_{IO}(S_2) = \{a^{2m} \mid m \in \mathbb{N}\}$, as the following derivation indicates:

$$x \Rightarrow \begin{array}{c}\varphi(a)\\ | \\ e\end{array} \Rightarrow \begin{array}{c}\varphi(a^2)\\ | \\ e\end{array} \Rightarrow \begin{array}{c}\varphi(a^4)\\ | \\ e\end{array} \Rightarrow \begin{array}{c}\varphi(a^8)\\ | \\ e\end{array} \Rightarrow \begin{array}{c}a^8\\ | \\ e\end{array} \ .$$

Using "+" instead of "," , we obtain
$L_{OI}(S_2) = \{a^{2^m} \mid m \in \mathbb{N}\}$, because <u>yield</u>$^{(2)}(T(S_2))$ =



□

<u>example 5</u> $x : \underline{0}$, $F_1, F_2 : \underline{1}$, $\varphi, \varphi_1 : \underline{2}$, $\psi : \underline{3}$

Let $S_3 \in NR_3(\Sigma_V)^{<e,i>}$ be the scheme

$$\longrightarrow x = \psi(\varphi)(a)(e)$$
$$\varphi(F_1)(y) = F_1 \circ F_1 \ (y)$$
$$\psi(\varphi_1)(F_2)(y) = \psi(\varphi_1 \circ \varphi_1)(F_2)(y) \ , \ \varphi_1(F_2)(y) \ .$$

Then $L_{IO}(S_3) = \{a^{2^{2^m}} \mid m \in \mathbb{N}\} = L_{OI}(S_3)$:

$$x \Rightarrow \begin{array}{c}\psi(\varphi)(a)\\ | \\ e\end{array} \Rightarrow \begin{array}{c}\psi(\varphi^2)(a)\\ | \\ e\end{array} \Rightarrow \begin{array}{c}\psi(\varphi^4)(a)\\ | \\ e\end{array} \Rightarrow \begin{array}{c}\varphi^4(a)\\ | \\ e\end{array}$$

$$\Rightarrow \begin{array}{c}\varphi_1{}^3(a^2)\\ | \\ e\end{array} \Rightarrow \begin{array}{c}\varphi_1{}^2(a^4)\\ | \\ e\end{array} \Rightarrow \begin{array}{c}\varphi_1(a^8)\\ | \\ e\end{array} \Rightarrow \begin{array}{c}a^{16}\\ | \\ e\end{array} \ .$$

□

In general, we can generate languages L_n defined by $L_n := \{a^{f_n(m)} \mid m \in \mathbb{N}\}$, where f_n is given by $f_1(m) := m$, $f_{n+1}(m) := 2^{f_n(m)}$.

<u>3.19 lemma</u>

$$L_n \in \mathcal{L}^n_{IO}(V) \cap \mathcal{L}^n_{OI}(V)$$

It follows directly from the corresponding results on tree languages, that the IO- and OI-string language hierarchies start with the regular and context-free languages, thus the given examples proof the first two steps to be strict, if $|V| \geq 2$.

<u>3.20 corollary</u>

$$\mathcal{L}^0_{IO}(V) \subsetneqq \mathcal{L}^1_{IO}(V) \subsetneqq \mathcal{L}^2_{IO}(V) \ \wedge \ \mathcal{L}^0_{OI}(V) \subsetneqq \mathcal{L}^1_{OI}(V) \subsetneqq \mathcal{L}^2_{OI}(V)$$

Moreover, Engelfriet and Schmidt [5] prove, that $\mathcal{L}^2_{IO}(V)$ and $\mathcal{L}^2_{OI}(V)$ equal the class of IO- and OI-macro languages [6], respectively.

We will now characterize the class of branch languages of level n IO-tree languages as level n OI-string languages.

Note first that any level n tree language over a many sorted alphabet Σ is also a level n tree language over a ranked alphabet $\bar{\Sigma}$ obtained from Σ by forgetting the sorts. Thus it is sufficient to prove the characterization for an $\{i\}$-sorted alphabet Σ. For technical reasons, we assume, that Σ contains at least one constant symbol c.

Let V_Σ be the string alphabet, which consists of all none constant symbols in Σ, and let $\underline{\Sigma} := \Sigma_{V_\Sigma}$.

The set $\underline{br}(L)$ of branches of a tree language L is defined by $\underline{br}(y_{j,i}) = \{e\}$, $\underline{br}(a) := \{e\}$, and $\underline{br}(f(t_1,..,t_r)) := \bigcup_{1\leq j\leq r} f(\underline{br}(t_j))$ for any symbol f of rank $r>0$; finally, $\underline{br}(L) := \bigcup_{t\in L} \underline{br}(t)$.

3.21 theorem

$$\underline{br}(\mathcal{T}^n_{IO}(\Sigma)) = \mathcal{L}^n_{IO}(V_\Sigma)$$

proof:

"$\subseteq$" Let $S' \in NR_n(\Sigma)^{<w,i>}$, $w = i^k$, and let $L(S') \neq \emptyset$, thus also $L_{IO}(S') \neq \emptyset$. As the emptiness problem for regular tree languages is solvable, we can eliminate all variables which do not derive a terminal tree, by deleting all productions, which contain such variables. Let $S \in NR_n(\Sigma)^{<w,i>}$ be the reduced grammar constructed from S' in this way, then $L_{IO}(S) = L_{IO}(S')$.

We define mappings $\underline{br}_m : T_{D^m(\Sigma)} \longrightarrow \mathcal{P}(T_{D^m(\underline{\Sigma})})$ $(0<m\leq n)$ by sending all constants in $\Sigma \subseteq D^m(\Sigma)$ to e and $f \in \Sigma$ of rank $r>0$ to the image of $\{f(y_{1,i}),..,f(y_{r,i})\}$ at the $D^m(\Sigma)$-level. Let $\underline{br}_0$ be defined as $\underline{br}$ except that $\underline{br}_0(y_{j,i}) = y_{j,i}$. Using induction, we prove

(1) that substitution at level n commutes with $\underline{br}_n$:

$\underline{br}_n \circ \underline{sub}_{X_\alpha}(t) = \underline{sub}_{X_\alpha}(\underline{br}_n(t)) \circ (\underline{br}_n;\ldots;\underline{br}_n)$

for any $t \in T_{D^n(\Sigma)}(X_\alpha)$, and

(2) that

$$\begin{tikzcd}
\mathcal{P}\_(T^{\underline{b}_n<w,i>}_{D^n(\Sigma)}) \arrow[r, "\underline{br}_n"] \arrow[d, "\underline{yield}^{(n)}"'] & \mathcal{P}\_(T^{\underline{b}_n<w,i>}_{D^n(\underline{\Sigma})}) \arrow[d, "\underline{yield}^{(n)}"] \\
\mathcal{P}\_(T_\Sigma(Y_w)^i) \arrow[r, "\underline{br}_0"'] & \mathcal{P}\_(T_{\underline{\Sigma}}(Y_w)^i)
\end{tikzcd}$$

commutes.

Here, $\underline{\text{sub}}_{x_\alpha}(t)$ and $\underline{\text{br}}_n$ are canonically extended to tree languages, and $P\_(M) := P(M) \setminus \{\emptyset\}$ for any set M.

Define $\underline{\text{br}}_n(S) \in NR_n(\underline{\Sigma})^{<w,i>}$ by taking $\underline{\text{br}}_n$ of each equation in S. Then, using (1) and the fact that S is reduced, we can show by induction on the length of a derivation that $\underline{\text{br}}_n(L(S)) = L(\underline{\text{br}}_n(S))$. But by (2) this implies $\underline{\text{br}}_0(L_{IO}(S)) = L_{IO}(\underline{\text{br}}_n(S))$. By calling $\underline{\text{br}}_n(S)$ with k e's as actual parameters, we obtain $S_1 \in NR_n(\underline{\Sigma})^{<e,i>}$, such that $\underline{\text{br}}(L_{IO}(S)) = L_{IO}(S_1)$.

"$\supseteq$" Let $S \in NR_n(\underline{\Sigma})^{<e,i>}$ be reduced. We define $D^m(\underline{\Sigma})$ - homomorphisms $\underline{\text{ext}}_m : T_{D^m(\underline{\Sigma})} \longrightarrow T_{D^m(\Sigma)}$ by imposing the following $D^m(\underline{\Sigma})$-structure on $T_{D^m(\Sigma)}$:

(i) $\underline{\Sigma} \ni e$ denotes $\underline{\bot} \mapsto c$

(ii) $\underline{\Sigma} \ni f$ denotes $\begin{cases} t \mapsto f(t,..,t) & \text{if } m=0 \\ \underline{\bot} \mapsto \text{the image of } f(y_{1,i},..,y_{1,i}) \\ \quad\text{at the } D^m(\Sigma)\text{-level} \end{cases}$

(iii) all other symbols in $D^m(\Sigma)$ retain their meaning.

Using induction, we show, that

(1) $\underline{\text{br}} \circ \underline{\text{ext}} = \underline{\text{id}}_{P(T_{\underline{\Sigma}})}$,

(2) substitution at level n commutes with $\underline{\text{ext}}_n$, and

(3)
$$\begin{tikzcd}
P\_(T^{b_n <e,i>}_{D^n(\underline{\Sigma})}) \arrow[r, "\underline{\text{ext}}_n"] \arrow[d, "\underline{\text{yield}}^{(n)}"'] & P\_(T^{b_n <e,i>}_{D^n(\Sigma)}) \arrow[d, "\underline{\text{yield}}^{(n)}"] \\
P\_(T_{\underline{\Sigma}}^{i}) \arrow[r, "\underline{\text{ext}}_0"'] & P\_(T_{\Sigma}^{i})
\end{tikzcd}$$

commutes.

Here, $\underline{\text{br}} = \underline{\text{br}}_0$, and $\underline{\text{ext}} := \underline{\text{ext}}_0$.

Define $\underline{\text{ext}}_n(S) \in NR_n(\Sigma)^{<e,i>}$ by taking $\underline{\text{ext}}_n$ of each equation in S. Then $\underline{\text{ext}}_n(S)$ is reduced, and using (2) we can show $\underline{\text{ext}}_n(L(S)) = L(\underline{\text{ext}}_n(S))$. But then $L_{IO}(\underline{\text{br}}_n(\underline{\text{ext}}_n(S))) = \underline{\text{br}}(\underline{\text{ext}}(\underline{\text{yield}}^{(n)}(L(S)))) = L_{IO}(S)$. □

In [2], Courcelle proves that OI-context-free tree languages have context-free branch languages. We conjecture that in general the branch language of a level n OI-tree language is a level n OI-string language, and that any such string language can be obtained this way.

By these characterizations and the examples, the tree language and scheme hierarchies are strict in the first two steps, if Σ contains at least a binary and a constant symbol.

<u>3.22 theorem</u>

$$\mathcal{T}^{0}_{IO}(\Sigma) \subsetneqq \mathcal{T}^{1}_{IO}(\Sigma) \subsetneqq \mathcal{T}^{2}_{IO}(\Sigma)$$
$$\mathcal{T}^{0}_{OI}(\Sigma) \subsetneqq \mathcal{T}^{1}_{OI}(\Sigma) \subsetneqq \mathcal{T}^{2}_{OI}(\Sigma)$$
$$R_2\ (\Sigma) \nleq R_1\ (\Sigma) \nleq R_0\ (\Sigma)$$

Thus recursion on higher types allows to define new objects at the base level. In particular, $R_2(\Sigma)$ is more powerful than the class of recursive program schemes over Σ.

We conjecture that for all $n \in \mathbb{N}+1$, $L_{n+1} \notin \mathcal{L}^{n}_{IO}(V) \cap \mathcal{L}^{n}_{OI}(V)$. To prove that L_{n+1} is not a level n language, we have to find characteristic closure operations for each level. A result in this direction is the increase in copying power for the IO-hierarchy:

let $\quad \delta_n : \mathcal{P}(V^*) \longrightarrow \mathcal{P}(V^*)$ be given by

$$\delta_n(L) := \{w^{f_n(m)} \mid m \in \mathbb{N} \wedge w \in L\}\ ,$$

then

$$\delta_n(\mathcal{L}^{n}_{IO}(V)) \subseteq \mathcal{L}^{n}_{IO}(V)\ .$$

4. References

[1] ADJ: Goguen, J.A./Thatcher, J.W./Wagner, E.G./Wright, J.B. Initial algebra semantics, IBM-report RC 5701, 1975

[2] Courcelle, B. Ensembles algêbriques d'arbres et langages dêterministes; quelques applications aux schêmas de programme, IRIA report, 1975

[3] Damm, W. Higher type program schemes and their tree languages, Proc. 3rd GI conference on Theoretical Computer Science, Lecture Notes in Computer Science, 48, Springer Verlag, 1977

[4] Damm, W. Higher type program schemes, to appear as a technical report of the RWTH Aachen

[5] Engelfriet, J./Schmidt, E.M. IO and OI, Datalogisk Afdelning report, DAIMI PB-47, Aarhus University, Denmark, 1975

[6] Fisher, M.J. Grammars with macro-like productions, Proc. 9th IEEE conference on Switching and Automata Theory, 1968

[7] Indermark, K. Schemes with recursion on higher types, Proc. 5th conference on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, 45, Springer Verlag, 1976

[8] Maibaum, T.S.E. A generalized approach to formal languages, JCSS 8 (1974), 409-439

[9] Nivat, M. Langages algêbriques sur le magma libre et sêmantique des schêmas de programme, in: Automata, Languages and Programming, ed. M. Nivat, North-Holland Publishing Company, Amsterdam, 1973

[10] Nivat, M. On the interpretation of recursive program schemes, Symposia Matematica, Vol. 15, Academic Press, 1975

[11] Wand, M. A concrete approach to abstract recursive definitions, in: Automata, Languages and Programming, ed. M. Nivat, North-Holland Publishing Company, Amsterdam, 1973

[12] Wand, M. An algebraic formulation of the Chomsky hierarchy, in: Category Theory applied to Computation and Control, Lecture Notes in Computer Science, 25, 1975