Parameter-Reduction of Higher Level Grammars

(Extended Abstract)

Helmut Seidl

Fachbereich Informatik

Johann Wolfgang Goethe-Universität

Postfach 111 932

D-6000 Frankfurt/Main

Abstract:
A higher level (OI-)grammar is called terminating, if for every accessible term t there is at least one terminal term which can be derived from t. A grammar is called parameter-reduced, if it is terminating and has no superfluous parameters.
For every grammar G of level n>0 which generates at least one term we construct grammars R(G) and P(G) such that R(G) and P(G) generate the same language as G but are terminating and paramter-reduced, respectively.
We introduce a hierarchy of restrictions to the deletion capability of the grammars which allow a gradual decrease in the complexity of the algorithms from n-iterated exponential time to polynomial time.

1. Introduction

Computing a fixed-point semantics w.r.t. a suitable small test algebra is an important tool for compile time program optimization. Recently, this method was applied to strictness analysis by several authors [BuHa85, Ab85, ClaJo85, HuYou86]. A function is called strict in one of its arguments if undefinedness of this argument implies undefinedness of the whole function. The motivation of such an analysis is that strict arguments for instance can be evaluated in parallel. However, as far as we know Damm in [Da82] was the first to apply fixed-point considerations to the analysis of higher level recursion in applicative programming. In this paper we extend Damm's work. Especially, we explain techniques on how to use the information about the "dynamic behavior" for some kinds of global optimization.

The theoretical model that our considerations refer to are the grammar families in the OI-hierarchy [Ma74, Wa74, EnSch77/78, Da77-82, Ga84]. The OI-hierarchy can be characterized in several ways: algebraically by solving regular systems of equations over higher and higher substitution algebras [Wa74, EnSch77/78, Ga84], automata-theoretically by using a storage type of iterated pushdowns [DaGoe82, En83] or grammatically by generalizing regular grammars by allowing nonterminal symbols to have parameters of increasing functional level [Da82]. Consider the following functional program for the computation of the factorial function in order to see how these grammars can be used as a model for programs written in a functional programming language.

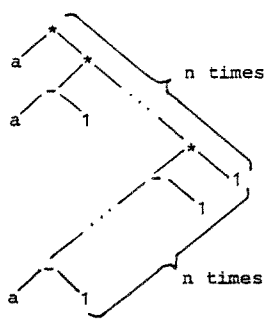
FACT[Y] := IF Y=0 THEN 1
ELSE *[Y,FACT[−[Y,1]]]

The corresponding grammar G consists of the derivation rules:

S → F(a) (serving as the initialization)
F(y)→ 1 | *(y,F(−(y,1)))

In the grammar (the names of) constants and built-in first order functions are modelled by the terminal symbols a,1,-,*, whereas the procedure name corresponds to the nonterminal symbol. Since we have no explicit semantic information about the (interpreted) value of the predicate Y=0 at the different stages of program execution, the best we can do is modelling the IF-THEN-ELSE-construct by a non-deterministic choice between the two possible procedure bodies. Thus the generated language is the set of terms of constants and built-in functions (here algebraic expressions) which possibly are evaluated in the computation, i.e.

t∈L(G) iff t=1 or t =



for some n ≥ 1

moreover, the derivation process of the grammar describes the process of function calls during the evaluation.[1)]

The idea behind this "abstraction" of concrete meanings is to determine those properties of the program which hold under all (reasonable) interpretations. (For an exact treatment of semantic aspects we refer to [Da82]).

Two of the most fundamental decidability questions for programs are the following:

(1) Does a given program terminate under all circumstances?

(2) Given a terminating program, does some function call within this program actually make use of all its formal parameters?

Clearly, for programs both questions are undecidable in general. For grammars, however, they are decidable and, even more, we are able to find effective transformations to the grammar such that the resulting grammar is terminating and has no superfluous parameters.

For contextfree tree grammars, i.e. the level 1 , the positive answers to both questions are known from the literature [Fi68, Cou78, EnSch77/78, Gue79, Le80]. These results are extended to constructions for arbitrary levels. We show that not only termination but also usefulness of level k parameters can be characterized by a fixed-point interpretation over some "small" domain. Besides this unifying aspect our constructions have another two advantages. The neccessary information about the dynamic behavior of the grammar can be computed directly by inspecting the description of the grammar itself, i.e. without using intermediate transformations, for example into grammars generating the language of branches as it was done in Guessarian's paper for the level 1 [Gue79] or in [Sei 86] for arbitrary levels; moreover, this computation is of a very simple structure (just determining a fixed point) and thus gives the

[1)] For the passing of parameters we implicitly use a (simply and homogeneously) typed λ-calculus together with a call-by-name or OI-evaluation strategy. We use applicative terms and consider copying the procedure body followed by a sequence of β-reductions as one derivation step. So, all terms we refer to are already in β-normalform (compare [Da82] that this is no restriction of

possibility for speed-ups corresponding to special features of the class of grammars in question. So, in general, for a grammar G of level n>0 determining simply whether or not G generates the empty language requires n-iterated exponential time (i.e. the grammar's size positioned on top of a tower of 2's of height n). Corresponding to the strong feeling that "in practice" a programmer does not make use of the full complexity of the formalism, we give an example for a hierarchy of restrictions to the structure of a grammar which allow to save more and more exponentiations until one ends up in a (still nontrivial) class of level n grammars with polynomial algorithms. A full version of this paper will appear in TCS.

2. Basic Notation and Concepts

Let I be an arbitrary set of indices. An I-tuple of sets $M=<M^i>_{i\in I}$ is called I-set. For two I-sets $M=<M^i>_{i\in I}$ and $N=<N^i>_{i\in I}$, the I-set $M\cup N$ is defined componentwise by $(M\cup N)^i=M^i\cup N^i$, $i\in I$. Similarily, an I-map $h:M\to N$ is an I-tuple of maps $h^i:N^i\to M^i$, $i\in I$. We omit indices as often as possible. So, if $x\in M^i$ for arbitrary i we say $x\in M$.

Let $\alpha=\alpha(1)...\alpha(m)$ be a word in I^*.

Let M be an I-set and $\alpha=\alpha(1)..\alpha(m)\in I^*$, then M^α denotes the Cartesian product $M^\alpha=M^{\alpha(1)}\times\cdots\times M^{\alpha(m)}$. We use the convention $M^\varepsilon=\{()\}$, i.e. the empty Cartesian product is supposed to be the set containing the empty list only. A typical element in M^α is the list $t_\alpha=(t_{1,\alpha(1)},...,t_{m,\alpha(m)})$ or $t_{1,\alpha(1)}..t_{m,\alpha(m)}$, since for the sake of simplicity we omit brackets if no confusions can arise.

2.1. Types and Terms

If I is a set of base types, then $D(I)=I^*\times I$ is the set of derived types over I.

This operation can be iterated. Let $D^0(I)=I$ and $D^n(I)=D(D^{n-1}(I))$ for n>0, then the set of (simple homogeneous) types over I is defined as $D^*(I)=\bigcup_{n\geq 0} D^n(I)$.

Here we only use a set of base types $I=\{i\}$ for a fixed i; however, similar con-

---

generality).

siderations and constructions also apply to a larger set of base types. If $\tau\in D^k(i)=D^k(\{i\})$, we say τ is of level k and write level(τ)=k. Arbitrary types are denoted by τ, τ', τ_0 etc., words of types of the same level always by α, α' etc. If $\alpha=\alpha(1)..\alpha(m)$, then $|\alpha|=m$ is the length of α. If α has a subscript, it denotes the level of the types in α.

For $k\in\mathbb{N}_0$, the k-rank of a type τ, $rk_k\tau$, is defined by

$$rk_k\tau=\begin{cases}0 & \text{if level}(\tau)\leq k\\ |\alpha| & \text{if level}(\tau)=k+1 \text{ and } \tau=<\alpha,\tau'>\\ \max\{rk_k\tau', rk_k\alpha(j)\,|\,j\in[1,|\alpha|]\} & \text{if level}(\tau)>k+1 \text{ and } \tau=<\alpha,\tau'>\end{cases}$$

The rank $rk(\tau)$ is defined by $rk(\tau)=\max\{rk_k\tau\,|\,k\geq 0\}$.

Finally, we define the size of a type τ, size(τ), as the number of i's occurring in τ.

In the sequel we always consider $D^*(i)$–sets where the special structure of types mirrors the "functional behavior" of the objects in question, that is objects of type i are viewed as constants, objects of type $<i^L,i>$ map a list of L constants onto a constant and so on. So, we call a $D^*(i)$–set M functional, if for all $\tau=<\alpha,\tau'>\neq i$, M^τ is a subset of $(M^{\tau'})^{M^\alpha}$, i.e. a set of functions from M^α into $M^{\tau'}$.

Let N be a $D^*(i)$–set of formal symbols with $N^\tau\cap N^{\tau'}=\phi$ for all $\tau\neq\tau'$. Then the $D^*(i)$-set T_N of applicative terms over N is the smallest $D^*(i)$–set T which contains N and is closed under formal application, i.e. which satisfies the two conditions

(A1) $N^\tau\subseteq T^\tau$;

(A2) if $u'\in T^{<\alpha,\tau>}$ and $u_\alpha\in T^\alpha$, then $u'u_\alpha\in T^\tau$.

Note that by (A2) the $D^*(i)$–set T_N can be viewed (via the canonical injection $u'\to\lambda y_\alpha.u'y_\alpha$) to be functional. Every applicative term t is contained in exactly one set T_N^τ . We call τ the type of t and write type(t)=τ . Accordingly for $t\in T_N^\alpha$ with $\alpha\in(D^k(i))^*$ we write type(t)=α.

Let M be another functional $D^*(i)$–set and $h:N\to M$ a $D^*(i)$–map. Then h can be canonically extended to a map $\bar{h}:T_N\to M$ by:

$$\bar{h}(t) = \begin{cases} h(t) & \text{if } t \in N \\ \bar{h}(u')\bar{h}(u_\alpha) & \text{if } t = u'u_\alpha \end{cases}$$

where $\bar{h}(u_\alpha)$ abbreviates $\bar{h}(u_{1,\alpha(1)}),..,\bar{h}(u_{m,\alpha(m)})$, $m=|\alpha|$. For the sake of simplicity we denote $\bar{h}$ by h as well.

As an example consider the identity map $Id:N\to T_N$. Then the canonical extension of Id to T_N clearly is the identity map on T_N.

The depth of t, depth(t), is defined by depth(t)=0 if $t\in N$ and depth(t)=max{depth(u'), 1+depth($u_{j,\alpha(j)}$) | $j\in[1,|\alpha|]$} if $t=u'u_\alpha$.

The size of t, |t| denotes the number of symbols occuring in t.

For any type $\tau=<\alpha_K,..,<\alpha_0,i>..>$ we define the $D^*(i)$-set $Y(\tau)$ of canonical formal parameters by $Y(\tau)^{\tau'}=\{y_{j,\alpha_\kappa(j)} \mid \tau'=\alpha_\kappa(j)\}$. For K=-1 i.e. $\tau=i$, we assume that $Y(\tau)^{\tau'}=\phi$ for all τ'.

Let M be a functional $D^*(i)$–set and $h:N\to M$ a $D^*(i)$–map. Then for $\pi=\rho_{\alpha_K}..\rho_{\alpha_0}$, $\rho_{\alpha_\kappa}\in M^{\alpha_\kappa}$, we denote by h_π the $D^*(i)$–map

$$h_\pi(x) = \begin{cases} h(x) & \text{if } x \in N \\ \rho_{j,\tau'} & \text{if } x = y_{j,\tau'} \end{cases}$$

Analogously to h, the $D^*(i)$–map h_π canonically extends to a $D^*(i)$–map $h_\pi:T_{N\cup Y(\tau)}\to M$. Note that if h_π is always applied to terms of level greater than some k-1, it suffices to specify π down to level k .

One special case of this definition is the usual substitution of the formal parameters $Y(\tau)$. For this assume $\bar{t}=\bar{t}_{\alpha_K}..\bar{t}_{\alpha_0}$ with $\bar{t}_{\alpha_\kappa}\in T_N^{\alpha_\kappa}$ for all $\kappa\in[0,K]$. Then $Id_{\bar{t}}$ defines the substitution of $\bar{t}$ into the formal parameters $Y(\tau)$. For some term $t\in T_{N\cup Y(\tau)}$ we write $t[\bar{t}]$ instead of $Id_{\bar{t}}(t)$.

2.2. Grammars

A grammar G is a 4-tuple $G=(N,\Sigma,S,P)$ with

(G1) N is a finite $D^*(i)$–set of nonterminal symbols;

(G2) Σ is a finite $D^*(i)$-set of terminal symbols with $\Sigma^\tau=\phi$ for all $\tau\in D^*(i)$ with level(τ)>1.

(G3) $S \in N^i$ is the initial symbol;

(G4) P is a finite set of (derivation) rules $f \equiv A! \to t$, where

the left hand side of f, lhs(f)=A, is a nonterminal symbol (A! is an abbreviation denoting "A applied to all its formal parameters", i.e. abbreviating A, if type(A)=i or $Ay_{\alpha_k} \ldots y_{\alpha_0}$ if type(A)=$<\alpha_k, \ldots, <\alpha_0, i> \ldots>$) and where the right hand side of f, rhs(f)=t, is a term in $T^i_{N \cup Y(type(A)) \cup \Sigma}$.

As usual all the occurring alphabet sets are assumed to be mutually disjoint; e.g. no formal parameter can simultaneously serve as a nonterminal symbol and vice versa.

OI-(outside in) derivations are sequences of nonterminal expansions (according to the rules of the grammar) taking place always at outermost occurrences of nonterminals. Leftmost derivations are OI-derivations where always the leftmost nonterminal is expanded.

For example, let t = a(A(B)c) be a term with $a \in \Sigma$, let $f \equiv Ay_{1,<i,i>} y_{1,i} \to y_{1,<i,i>}(y_{1,<i,i>}(y_{1,<i,i>} y_{1,i}))$ be a rule in P. Then A is the leftmost occurrance of a nonterminal in t. Expanding A in t by application of f yields a(B(B(Bc))).

Note: every rule f in P can be viewed (w.r.t. leftmost application) as a partial function $T^i_{N \cup \Sigma} \to T^i_{N \cup \Sigma}$. Accordingly, every $F \in P^*$ defines a partial function $F: T^i_{N \cup \Sigma} \to T^i_{N \cup \Sigma}$ by:

- $t\varepsilon = t$; and
- $tF'f = (tF')f$ for every $F' \in P^*$ and every $f \in P$.

Thus, a leftmost derivation w.r.t. G can be viewed as a pair (t,F) where tF is defined. By the results of Damm in [Da82], every derivation yielding a term t in T_Σ can be rearranged into a leftmost derivation yielding t. Therefore, we can restrict ourselves to the analysis of leftmost derivations. We define SP^* as the set of terms accessible w.r.t. G, and $L(G) = SP^* \cap T^i_\Sigma$ as the language generated by

Since the terms generated by G are (finite ordered Σ-labelled) trees, a grammar G, in general, is called tree grammar. As usual monadic trees are identified with strings (the unique leaf is viewed as an end marker). So, if $\Sigma^{\tau}=\phi$ for any τ of level 1 with $rk(\tau)\neq 1$ (this ensures that all generated trees are monadic), G is also called string grammar.

The level of G is defined as $level(G)=max\{level(A)|A\in N\}$. The k-rank of G is defined as $rk_k(G)=max\{rk_k x \mid x\in N\cup\Sigma\}$; the rank of G is defined as $rk(G)=max\{rk_k(G)\mid k\geq 0\}$.

Recall that the class of grammars with level 0 generate exactly the regular languages; whereas the language classes generated by level 1 or level 2 grammars coincide with the contextfree or macro OI-language classes, respectively [EnSch77/78].

2.3. Complexity Measures

The model of computation on which our algorithms and constructions run is a sequential Random Access Machine (RAM) with the uniform cost criterion (compare [Ah74,Paul78] for a precise definition). However, since we never use any multiplications (or divisions) our algorithms can be implemented on a determinstic Turing machine with polynomially related time complexity [Paul78]. Within the random access memory of our RAM a grammar G is represented by:

- a list $((x,type(x)))_{x\in N\cup\Sigma}$, called type convention of G ; and
- a list of the derivation rules.

We assume that every symbol x is stored in a different storage cell; the right hand sides of rules are represented as a labelled ordered tree.

Since we want to measure the complexity of our algorithms relative to the "size" of the input grammar, it makes sense to define the size of G , size(G), by

$$size(G)=\sum_{x\in N\cup\Sigma}(1+size(type(x)))+\sum_{f\in P}(1+|rhs(f)|)\ .$$

3. Terminating Grammars

Let $G=(N,\Sigma,S,P)$ be a grammar of level $n>0$, and let $t\in T^i_{N\cup\Sigma}$ be a term.

We say t is terminating, if there is a leftmost derivation (t,F) with $tF\in T^i_\Sigma$. We say t is faithfully terminating, if for every leftmost derivation (t,F) , the term tF is terminating. We say G is terminating, if every accessible term is terminating.

Damm showed that termination can be characterized by some fixed-point interpretation h^* over the domain B of iterated monotone Boolean functions [Da82]. B is defined by:

$B^i=\{0,1\}$ with $0\sqsubseteq 1$, and

$B^\tau=[B^\alpha\to B^{\tau'}]$ for $\tau=<\alpha,\tau'>$.

In this definition B^α is equipped with the partial order induced by the partial orders of its components, whereas $[..\to..]$ denotes the set of all monotone functions equipped with the standard partial order.

Note that for every $\tau\in D^*(i)$, B^τ is in fact a complete lattice.

The fixed-point is computed by a process of successive approximation:

for every $x\in N\cup\Sigma$, define $h^*x=\bigsqcup_{m\geq 0} h^{(m)}x$ where

$h^{(0)}x=1$ if $x\in\Sigma^i$, $h^{(0)}x=\&_L$ if $x\in\Sigma^{<i^L,i>}$ ($\&_L$ denotes an L-ary logical conjunction) and $h^{(0)}x=0$ for all $x\in N$;

for $m>0$, $h^{(m)}x=h^{(0)}x$ for all $x\in\Sigma$; if $x\in N^\tau$ and $P_x=\{f\in P\mid lhs(f)=x\}$, then $h^{(m)}x=\bigsqcup_{f\in P_x}\rho_f^{(m)}$ where $\rho_f^{(m)}$ is uniquely determined by

$\rho_f^{(m)}\pi=1$ iff $h_\pi^{(m-1)}rhs(f)=1$.

Note that $h^{(m)}x\sqsubseteq h^{(m+1)}x$ for all $x\in N\cup\Sigma$ and $m\geq 0$.

We introduce a hierarchy of stronger and stronger restrictions to the grammar which permit modelling termination with poorer and poorer domains. The result is a gradual decrease in complexity from n-iterated exponential time down to polynomial time.

Let $G=(N,\Sigma,S,P)$ be a grammar of level $n>0$. Let $k\in[0,n]$. We say G is non-deleting

up to level k-1, if for every rule $f\equiv A!\to u\in P$ the following holds:

let $type(A)=<\alpha_K,..<\alpha_0,i>..>$ with $K\geq -1$; then every formal parameter $y_{j,\alpha_\kappa(j)}$ with $\kappa<k$ occurs at least once in the right hand side of f.

Note that every grammar G is non-deleting "up to level -1". If G is non-deleting up to level n-1, then the nonterminal at the active position is the only symbol which can be eliminated by application of a derivation rule but none of its actual parameters. In this case we say G is (totally) non-deleting. Clearly, restrictions to the possibilities of deletion severely restrict the generative capacity of the grammars. However, the "typical" facility of n-iterated exponential growth of generated terms is preserved; what is more, the grammars in [Da82] that are constructed to distinguish between the language classes of different levels are,in fact, non-deleting ones.

Call a $D^*(i)$–set M trivialization of B if M has the following properties:

(1) $M^i=B^i$; and

(2) if $\tau=<\alpha,\tau'>$, then M^τ is a complete sublattice of $[M^\alpha\to M^{\tau'}]$.

By property (2), every trivialization of B is functional. The unique minimal element and the unique maximal element in any M^τ are also denoted by 0 and 1 , respectively. The least upper bound operation join is denoted by $\sqcup$; and the partial order relations always by $\sqsubseteq$.

Let M be a trivialization of B. A $D^*(i)$–map $h:N\cup\Sigma\to M$ is called reducing for G over M if for every $t\in T^i_{N\cup\Sigma}$, ht=1 iff t is terminating.

By Damm's results in [Da82] the $D^*(i)$–map h^* is a reducing map over B. We show how restrictions to the deleting capability of the grammar can be modelled by a corresponding trivialization of B.

For $k\geq 0$ define M_k by

(T1) $M^i_k=B^i$;

(T2) for level(τ)≤k and $\tau=<\alpha,\tau'>$, let $M_k^\tau=\{0,1\}$ with $0\rho_\alpha=0$ for all $\rho_\alpha\in M_k^\alpha$; and $1\rho_\alpha=1$ iff $\rho_{j,\alpha(j)}=1$ for all $j\in[1,|\alpha|]$;

(T3) for level(τ)>k and $\tau=<\alpha,\tau'>$, let $M_k^\tau=[M_k^\alpha\to M_k^{\tau'}]$.

Note that M_0 coincides with B.

Actually, Damm's computation of a reducing map h^* over B can be carried over to a construction of a reducing map over the restricted domain M_k provided G is non-deleting up to level k-1. We get:

3.1. Theorem

Let G be non-deleting up to level k-1. Then there exists a reducing map $h^*:N\cup\Sigma\to M_k$.

We show the following theorem:

3.2. Theorem

Let G=(N,Σ,S,P) of level n>0 with $L(G)\neq\phi$. Then there is a terminating grammar $R(G)$ with $L(R(G))=L(G)$; the construction can be done level-, depth- and non-deletion-preserving, i.e.
level(R(G))≤level(G), depth(R(G))≤depth(G), and if G is non-deleting up to level k-1, then R(G) is non-deleting up to level k-1 as well.

The idea of the transformation R is as follows: The value of the actual parameters under h^* is stored in the nonterminal (i.e. as a subscript), whereas the value of future arguments are "guessed" or "produced in parallel" by splitting the corresponding parameter positions from which the correct one later is chosen by the means of projection.

From a practical point of view the grammar R(G) is obtained from G by adding the extra information about the values of occurring argument lists under h^* and suppressing terms of value 0 ; especially, every leftmost derivation w.r.t. G is

transformed into at most one leftmost derivation w.r.t. $R(G)$. Note that a corresponding implementation actually does not need to perform the possibly very expensive splitting of parameters.

The $D^*(i)$–map h^* is nothing but the fixed-point interpretation as it was used in [BuHa85] or [Ab85] for strictness analysis. Since the actual parameters of some function call may be filled in dynamically during a program execution, the extra information about the values under h^* can be used to detect even strictness of certain parameters which only occurs at runtime.

4. Parameter–Reducedness of Grammars

Let $G=(N,\Sigma,S,P)$ be a grammar of level $n>0$. Let $k\in[0,n-1]$.

A term $t\in T^i_{N\cup\Sigma}$ is called parameter-reduced up to level k-1 , if

(1) t is faithfully terminating; and

(2) for every leftmost derivation (t,F) w.r.t. G every subterm u of tF with level(u)<k is used during some leftmost derivation (tF,F').

G is called parameter-reduced up to level k-1, if every accessible term t is parameter-reduced up to level k-1. G is called (totally) parameter-reduced, if G is parameter-reduced up to level n-1.

Note that G is parameter-reduced up to level -1, iff G is terminating. Furthermore, if G is non-deleting up to level k-1 and terminating, then G is parameter-reduced up to level k-1 as well.

The levelwise definition of parameter-reducedness suggests our strategy for eliminating useless parameters. We start at k=0 and proceed level by level. The base of this induction is given by the transformation R of the last chapter. So, we may assume that G is already parameter-reduced up to level k-1 with $k\geq 0$. We will construct a grammar $P_k(G)$ which generates the same language as G and is parameter-reduced up to level k. For this, analogous to the last chapter, we first model the usefulness of level k parameters by an appropriate trivialization of B, then we compute a suitable fixed-point interpretation h^*_k w.r.t. this

trivialization which finally is integrated into the grammar. The parameter-reducing transformation P for a grammar G of level n which is non-deleting up to level k-1 then simply is defined by $P=P_{n-1}..P_k$ if G is already terminating or $P=P_{n-1}..P_kR$ if G is not necessarily terminating. However, the curious fact may be noted that in contrast to the situation for n=1, even for terminating grammars parameter-reduction does not automatically lead to a shrinkage of the grammar, but can even cause a tremendous blow-up of the grammar's size.

For $k\geq 0$ define the $D^*(i)$–set $\overline{M}_k$ by:

(U1) $\overline{M}_k^i=B^i$;

(U2) for $\tau=<\alpha,\tau'>$ and $level(\tau)\leq k$, let $\overline{M}_k^\tau=\{0,1\}$ with

$0\rho_\alpha=1$ iff $\rho_{j,\alpha(j)}=1$ for at least one j, and

$1\rho_\alpha=1$ for all ρ_α;

(U3) for $\tau=<\alpha,\tau'>$ and $level(\tau)=k+1$, let $\overline{M}_k^\tau=\{1_J \mid J\subseteq[1,|\alpha|]\}$ with

$1_J\rho_\alpha=1$ iff $\rho_{j,\alpha(j)}=1$ for at least one $j\in J$;

(U4) for $\tau=<\alpha,\tau'>$ and $level(\tau)>k+1$, let $\overline{M}_k^\tau=[\overline{M}_k^\alpha\to\overline{M}_k^{\tau'}]$.

It is readily checked, that $\overline{M}_k$ is a trivialization of B. The functions 1_J of level k+1 are supposed to describe the usefulness of formal parameters of level k: if $v\in T_{N\cup\Sigma}^{<\alpha,\tau'>}$ is a term of level k+1 and $J\subseteq[1,|\alpha|]$ is the set of indices of useful parameters to v of level k , then we want to map v to the value 1_J. For this we compute a map h_k^* as the least upper bound of approximations $h_k^{(m)}$ where nonterminals of level $\leq k$ and terminal symbols are interpreted as some sort of "generalized join". Not every formal parameter $y_{j,\alpha_\kappa(j)}$ with level $\kappa<k$ of some nonterminal symbol A necessarily occurs in all right hand sides of rules f with lhs(f)=A. However, since G is parameter-reduced up to level k-1 there is at least one rule f for which $y_{j,\alpha_\kappa(j)}$ occurs in rhs(f). Therefore, we won't compute h_k^* over G itself but over the corresponding typed λ-scheme G_+ [Da82].

Let + be a new terminal symbol of type <ii,i> (denoting formal union) and define

Σ_+ as the $D^*(i)$–set by

$\Sigma_+^{<ii,i>} = \Sigma^{<ii,i>} \cup \{+\}$ and $\Sigma_+^\tau = \Sigma^\tau$ for $\tau \neq <ii,i>$.

The typed λ–scheme G_+ is gained from G by adding + to the set of terminal symbols and formally adding the right hand sides of all rules having the same left hand sides by the means of a binary tree whose inner nodes are labelled by +.

In general this scheme no longer generates a tree language but a (finite or) infinite tree. However, from such a tree the tree language L(G) may be regained by interpreting the undefined tree as the empty set and + as the union operator over the powerset of T_Σ^i [Da82]. Note that Guessarian actually investigated parameter-reduction for typed λ–schemes of level 1.

Set $h_k^* x = \bigsqcup_{m \geq 0} h_k^{(m)} x$ where

$$h_k^{(0)} x = \begin{cases} 1_{[1,L]} & \text{if } x \in \Sigma_+^{<L,i>} \text{ and } k=0 \\ 0 & \text{else} \end{cases}$$

and for $m>0$, $h_k^{(m)}$ is defined as follows:

if $x \in \Sigma_+$ or $level(x) \leq k$, then $h_k^{(m)} x = h_k^{(0)} x$;

if $A \in N$ with $level(A) > k$ and u is the right hand side of the unique rule for A in G_+, let $h_k^{(m)} A$ be the function in $\overline{M}_k^{type(A)}$ with

$(h_k^{(m)} A)\pi = 1$ iff $h_{k,\pi}^{(m-1)} u = 1$.

Assume G is parameter-reduced up to level k-1 for some $k \geq 0$. We integrate the evaluation of h_k^* into the grammar G in a similar way as h^* was embodied into G in chapter 3 .

Similar to the definition of R, the current value under the interpretation of the argument lists is stored in the nonterminal symbol whereas possible future argument values are guessed by splitting of parameters. However, the additional information now is used to eliminate parameters detected as superfluous. We get:

4.1. Theorem

Let G be parameter-reduced up to level k-1 with level(G)=n>0 , then there is a grammar $P_k(G)$ which is parameter-reduced up to level k with $L(P_k(G))=L(G)$. The construction can be done level-, depth- and non-deletion preserving, i.e. $level(P_k(G))\leq level(G)$, $depth(P_k(G))\leq depth(G)$, and if G is non-deleting up to some level less than k, then $P_k(G)$ is non-deleting at least up to the same level.

Similar to *R*, the transformation *P* consists of adding extra information about the value of argument lists under certain fixed-point interpretations, splitting of parameters and removing certain useless parts. Especially, every leftmost derivation is transformed into exactly one new leftmost derivation.

5. Structural and Computational Complexity

In this chapter we give upper bounds for the amount of time for constructing the grammars *R*(G) and *P*(G) in terms of structural constants related to the description of G.

For $K\geq 0$, define the K-iterated exponential function $exp_K(x)$ by:

$exp_0(x)=x$ and $exp_K(x)=2^{exp_{K-1}(x)}$ for $K>0$.

The next theorem summarizes the complexity results for the transformation *R*. In this theorem as well as in the corresponding theorem 5.4. for the transformation *P* we assume for certain subcases that the grammar's depth is bounded by some fixed constant. This restriction does not restrict the generality. Given a grammar G , one always can construct an equivalent grammar of depth 2 just by introducing new nonterminals for subterms of depth greater than 1 occurring in the right hand sides of rules [Da82]. Since this construction can be done in polynomial time without increasing level or rank or introducing deletion, the theorems 5.2. and 5.4. imply that also for grammars without depth-restriction an equivalent terminating grammar or an equivalent parameter-reduced grammar, respectively, can be constructed in a time which is polynomially related to the corresponding time stated in the theorems.

5.1. Theorem

Let G be a grammar of level $n \geq k$ with $L(G) \neq \phi$ and non-deleting up to level k-1. Then the following holds:

(1) if n=k, then $size(R(G)) \leq size(G)$, and $R(G)$ can be constructed in time $O(size(G)^2)$;

(2) if n>k and and the depth of G is bounded by some fixed constant, then $size(R(G)) \leq size(G) \cdot exp_{n-k}(cL)$, and $R(G)$ can be constructed from G in time $O(size(G)^2 exp_{n-k}(cL))$ for a constant c>1 which only depends on n;

(3) for n>k the construction is optimal w.r.t. the number of exponentiations and the degree of the polynomial in the highest exponent.

Proof:

The upper time bounds are obtained by simply translating the definition of $R(G)$ in the corresponding construction.

To prove the construction to be optimal we use a result of Engelfriet [En83] where he shows the following: given (the adequately coded description of) a n-iterated pushdown automaton A, n>1, then whether or not A accepts the empty language cannot be decided in deterministic Turing machine time $\Omega(exp_{n-1}(c|A|^{2-\varepsilon})$ for every c>0 and every $\varepsilon>0$ (here |A| means the length of the description of A). By the standard constructions of [DaGoe83] and [Da82] one can construct from A a grammar G with polynomially related size, and with rank $|Q|^2$ where Q is the set of states of A such that G generates the same language as A and is non-deleting up to level 0. Since this construction can be performed in polynomial time also, the lower bound of Engelfriet implies that our upper bound for the construction of h is sharp at least for grammars of level n>1 which are non-deleting up to level 0. However, with the help of the tree transformation in [Da82] or [Sei86] it follows that our upper bound is also sharp for arbitrary grammars of level n>0. Now consider for k>0 the following type transformation ϑ_k:

$\vartheta_k i=<\varepsilon,..<\varepsilon,i>..>$ (k empty parameter lists); and

$\vartheta_k<\tau_1..\tau_m,\tau_0>=<\vartheta_k\tau_1..\vartheta_k\tau_m,\vartheta_k\tau_0>$.

By this type transformation the class of grammars of level n-k can be viewed as a subclass of the grammars of level n which are non-deleting up to level k-1. Hence, our result is sharp even for grammars of level n>0 which are non-deleting up to level k-1 for some $k\in[0,n-1]$. This completes the proof.

5.2. Theorem

Let G be a grammar of level n>0 and non-deleting up to level k-1. Let $L=\max\{rk_K(G)\mid K\geq k\}$.

(1) Assume G is terminating. Then the following holds:

(1.1) If n=k, then G already is parameter-reduced.
If n=k+1 , then $size(P(G))\leq size(G)$, and $P(G)$ can be constructed in time $O(size(G)^2L)$.

(1.2) If level(G)>k+1 and the depth of G is bounded by some fixed constant, then $size(P(G))\leq size(G)\cdot exp_{n-k-1}(cL^2)$, and $P(G)$ can be constructed from G in time $O(size(G)^2exp_{n-k-1}(cL^2))$ for some constant c>1 which only depends on n.

(2) Assume $L(G)\neq\phi$, but G is not terminating. Then we have:

(2.1) if n=k, then $size(P(G))\leq size(G)$, and $P(G)$ can be constructed in time $O(size(G)^2)$;

(2.2) if n>k and if the depth of G is bounded by some fixed constant, then $size(P(G))\leq size(G)\cdot exp_{n-k}(cL)$, and $P(G)$ can be constructed from G in time $O(size(G)^2exp_{n-k}(cL))$ for a constant c>1 which only depends on n.

For a grammar G which is not necessarily terminating, the transformation P can be computed in the same order of magnitude of time as the transformation R which proves the transformation to be optimal at least in this situation.

On the whole, the results 5.1. and 5.2. very clearly describe how different

descriptional constants of a grammar influence the computational complexity in different ways: actually it is not the size of the grammar or any constant related to the size (like the number of nonterminals or the number of rules) but the level and the rank i.e. the descriptional constants refering to the type concept of the grammar which have the most tremendous effects on the runtime of our algorithms.

6.Conclusion

We investigated termination and parameter-reducedness for grammars. We found restrictions to the deletion capability of level n grammars such that the time complexity of our constructions could be reduced from n-iterated exponential time to polynomial time. So, the question arises whether there are other syntactical features of grammars which allow the use of simpler models and thus to get improvements of the general complexity result. Since our transformations in general do not leave a grammar unchanged even if the grammar already was parameter-reduced, it seems to be also important to find at least sufficient conditions for the termination or parameter-reducedness of a grammar.

In a second paper we will show for higher level grammars several (hierarchies of) decidability results. In fact, we show that the finiteness problem is decidable as well as regularity for certain simple subclasses of higher level grammars. We also present classes where finiteness or regularity can be decided in polynomial time.

References:

Ab85 S. Abramsky: Strictness analysis and polymorphic invariance. Programs as Data Objects, Proc. of a Workshop Copenhagen, Denmark, 1985, Lecture Notes in Comp. Sci. 217 pp. 1-23;

Ah74 A.V. Aho, J.E. Hopcroft, J.D. Ullman: The Design and Analysis of Computer Algorithms. Addison Wesley, Reading, MA, 1974

ArDau76 A. Arnold, M Dauchet: Un theoreme de duplication pour les forets algebriques. JCSS 13, 1976, pp. 223-244;

ArDau78 A. Arnold, M. Dauchet: Forets algebriques et homomorphismes inverses. Inf. and Control 37, 1978, pp. 182-196;

ArLe80 A. Arnold, B. Leguy: Une propriete des forets algebriques "de Greibach". Inf. and Control 46, 1980, pp. 108-134;

Ba81 H.P. Barendregt: The Lambda-Calculus. North Holland, Amsterdam, 1981;

BuHa85 G.L. Burn, C.L. Hankin, S. Abramsky: The theory of strictness analysis for higher order functions. Programs as Data Objects, Proc. of a Workshop, Copenhagen, Denmark 1985, Lecture Notes in Comp. Sci. 217, pp. 46-66;

ClaJo85 C. Clack, S.L.P. Jones: Strictness analysis - a practical approach. Functional Programming Languages and Computer Architecture, Lecture Notes in Comp. Sci. 201, 1985, pp. 35-49;

Cou78 B. Courcelle: A representation of trees by languages. TCS 6, 1978, pp. 255-279 and TCS 7,1978, pp. 25-55;

Cou86 B. Courcelle: Equivalences and transformations of regular systems - applications to recursive program schemes and grammars. TCS 42, 1986, pp. 1-122;

Da77a W. Damm: Higer type program schemes and their tree languages. Proc. 3. GI-Conf. on Th. Comp. Sci. Lecture Notes in Comp. Sci. 48, 1977, S. 51-72

Da77b W. Damm: Languages defined by higher type program schemes. Proc. 4th Int. Coll. on Automata, Languages and Programming Lecture Notes in Comp. Sci. 52, 1977, pp. 164-179;

Da79 W. Damm: An algebraic extension of the Chomsky-hierarchy. Proc. Conf. on Math. Foundations of Comp. Sci. Lecture Notes in Comp. Sci. 74, 1979, pp. 266-276;

Da82 W. Damm: The IO- and OI-hierarchies. TCS 20, 1982, pp. 95-205;

DaFe80 W. Damm, E. Fehr: A schematological approach to the analysis of the procedure concept in ALGOL-languages. Proc. 5ieme Colloque sur les Arbres en Algebre et en Programmation, 1980, pp. 130-134;

DaFeIn78 W. Damm, E. Fehr, K. Indermark: Higher type recursion and self-application as control structures. In: E. Neuhold (ed.): Formal Description of Programming Concepts, North Holland, Amsterdam, 1978, pp. 461-487;

DaGoe82 W. Damm, A. Goerdt: An automata-theoretical characterization of the OI-hierarchy. Inf. and Comp. 71, 1986, pp. 1-32;

DaGue81 W. Damm, I. Guessarian: Combining T and level N. Tech. Report Laboratoire Informatique Theorique et Programmation 81-11, 1981;

DaGue83 W. Damm, I. Guessarian: Implementation techniques for recursive tree transducers on higher order data types. Tech. Report Laboratoire Informatique Theorique et Programmation 83-16, 1983;

EnSch77/78 J. Engelfriet, E.M. Schmidt: IO and OI. JCSS 15, 1977, pp. 328-353, and JCSS 16, 1978, pp. 67-99;

En83 J. Engelfriet: Iterated pushdown automata and complexity classes. Proc. 15th STOC, 1983, pp. 365-373;

Fi68 M.J. Fischer: Grammars with macro-like productions. Proc. 9th IEEE Conf. on Switching and Automata Theory, 1968, pp. 131-141;

FoLei83 S. Fortune, D. Leivant, M. o'Donnell: The expressiveness of simple and second order type structures. JACM 30, 1983, pp. 151-185;

Ga84 J.H. Gallier: n-Rational algebras. SIAM J. of Computing 13, 1984, pp. 750-794;

Gue79 I. Guessarian: Program transformations and algebraic semantics. TCS 9, 1979, pp. 39-65;

HuYou86 P. Hudak, J. Young: Higher-order strictness analysis in untyped lambda calculus. Proc. of the 13th Ann. Symp. on Principles of Programming Languages, 1986, pp. 97-109;

Le80 B. Leguy: Reductions, transformations et classification des grammairs algebriques d'arbres. These de 3ieme cycle Lille, 1980;

Le81 B. Leguy: Grammars without erasing rules, the OI-case. Proc. 6ieme Colloque sur les Arbres en Algebre et on Progammation, 1981, Lecture Notes in Comp. Sci. 112, pp. 268-273;

Mai74 T.S.E. Maibaum: A generalized approach to formal languages. JCSS 8, 1974, pp. 409-439;

Ma74 A.N. Maslov: The hierarchy of indexed languages of an arbitrary level. Soviet. Math. Dokt. 15(14), 1974, pp. 1170-1174;

Mi77 R. Milner: Fully abstrct models of typed lambda–calculus. TCS 4, 1977, pp. 1-22;

My80 A. Mycroft: The theory and practice of transforming call-by-need into call-by-value. Proc.4th Colloque International sur la Programmation Paris, 1980, pp. 269-281

Pa78 W.J. Paul: Kompexitätstheorie. Teubner Stuttgart, 1978;

Sch78 E.M. Schmidt: Succinctness of description of contextfree, regular and finite languages. Datalogisk Afdelning Report DAIMI PB-84, Aarhus Univ., 1978;

Sei86 H. Seidl: Regularität bei Grammatiken höherer Stufe. Diss. Thesis, Frankfurt/Main, 1986;

Sei87 H. Seidl: Parameter-reduction of Higher Level Grammars. To appear in TCS

Wa75 M. Wand: An algebraic formulation of the Chomsky-hierarchy. Category Theory Applications to Computation and Control, Lecture Notes in Comp. Sci. 25, 1975, pp. 209-219.