1. Introduction

We consider and compare macro grammars, L systems, stack automata and topdown tree transducers: extensions of context-free grammars which allow certain kinds of copying. Macro grammars are context-free grammars in which the non-terminals have parameters; in particular in 'basic' macro grammars the actual parameters of a non-terminal are terminal strings (i.e. nonterminals are not nested). We also consider 'extended' macro grammars in which the actual parameters may be finite sets of terminal strings. ETOL systems are like context-free grammars, but the rewriting is done in parallel and several independent sets of productions are allowed. Stack automata are pushdown automata that may also read in the stack. Topdown tree transducers transform the set of derivation trees of a context-free grammar. The language of yields of the resulting set of trees is called a tree transformation language.

Several relationships between these devices are known: in both macro grammars and L-systems the operation of iterated substitution plays a role, macro languages can be recognized by 'nested stack automata', nonerasing stack languages are special ETOL languages, ETOL languages are both special tree transformation languages and special macro languages, and finally macro grammars generate the yields of context-free tree languages.

In this paper we continue the comparison of these devices. We show that the additional facilities present in the basic macro grammars, the ETOL systems and the stack automata are independent in the sense that the corresponding classes of languages are incomparable. In particular we present a language which is both in Basic and Stack but is not even a tree transformation language (this also shows that the context-free tree grammars are independent from the topdown tree transducers as string language generating systems). We then prove that Basic, ETOL and Stack are contained in the class EB of extended basic macro languages. Results analogous to those for Stack are given on the operation of substitution in EB. It follows that EB is a full AFL which is not substitution closed. We finally show that the smallest full hyper-AFL (i.e. full AFL closed under iterated substitution) containing EB lies properly between EB and the class of all (OI) macro languages. This shows that OI (= the class of indexed languages) cannot be reached from Basic or Stack by full hyper-AFL operations.

Proofs of these results will only be sketched; full proofs will appear elsewhere.

2. Terminology and facts

We assume the reader to be familiar with macro grammars [16], and more or less
with iterated substitution [26], stack automata [18, 19] and tree transducers [25].
What follows is meant to fix some notation and mention some facts.

The length of a string is denoted by $|w|$; $|\lambda| = 0$.

An (OI or IO) macro grammar $G$ consists of an alphabet $E$ of terminals, an alphabet $N$ of nonterminals each of which has a specified rank (i.e. a nonnegative number of arguments), an initial nonterminal $S$ of rank 0, and a finite set $R$ of rules of the form $A(x_1, \ldots, x_n) \rightarrow t$ where $A$ is a nonterminal of rank $n$, $x_1, \ldots, x_n$ are special symbols called variables and $t$ is a term formed from $\{x_1, \ldots, x_n\} \cup E \cup \{\lambda\}$ by concatenation and the use of the nonterminals as formal operation symbols. The rules are applied in the obvious (outside-in or inside-out respectively) way and $L(G)$ denotes the generated language. For formal definitions see [16]. In a basic macro grammar the terms in right-hand sides of rules do not have nested nonterminals and in a linear basic macro grammar they have at most one nonterminal. The classes of OI, IO, basic and linear basic macro languages are denoted by IO, IO, Basic and LB respectively. An extended macro grammar (cf. [9]) is a macro grammar in which the operation of union, denoted by $+$, and the empty set, denoted by $\emptyset$, may also used in terms. Thus, during derivation, (representations of) finite sets of terms are stored in the arguments and, at the end of the derivation, (the representation of) a finite set of terminal strings is produced; the union of these sets is the language generated. The classes of extended basic and linear basic languages are denoted by EB and ELB respectively. Clearly each extended basic macro grammar can be simulated by an ordinary OI macro grammar which uses additional nonterminals $+$ and $\emptyset$ (with rules $+(x,y) \rightarrow x$ and $+(x,y) \rightarrow y$ for $+$, and no rules for $\emptyset$). Thus $EB \subseteq OI$. As an example, the EB (even EB) grammar $G$ with rules $S \rightarrow \alpha(x)\beta(x)\gamma$, $\alpha(x)\beta(x)\gamma \rightarrow \alpha(x)\beta(y)$ for all $x \in \Sigma$, and $\beta(x,y) \rightarrow \alpha(x)\beta(y)$ generates $L(G) = \{w_1 \# w_2 \# \ldots \# w_n \# w \mid n \geq 1, w \in \{w_1, \ldots, w_n\}\}$.

For a finite set $U$ of substitutions and a language $L$ we define $U^*(L) = \bigcup \{f^{n_L(1)}(L) \mid n_L(1) \geq 0, f \in U\}$. $U^*$ is called an iterated substitution. For a family $K$ of languages we define $H(K) = \{U^*(L) \cap L^* \mid L \in K, U$ is a finite set of $K$-substitutions and $\Sigma$ is an alphabet$\}$, where a $K$-substitution is a substitution that maps symbols into languages of $K$. A construct $(V, E, L, U, L)$ with $L \subseteq V^*$ is called a $K$-iteration grammar: a generalization of ETOL system (see for instance [26]). A family $K$ is called a full hyper-AFL if it is a full AFL closed under iterated substitution (i.e. $H(K) \subseteq K$). The families $H(FIN)$ and $H(ONE)$, where FIN and ONE are the finite and the singleton languages, are denoted by ETOL and EDTOL respectively.

For the definition of (one-way nondeterministic) stack automaton, nonerasing stack automaton and checking stack automaton we refer to [18, 19]. The corresponding classes of languages will be denoted by Stack, NESTack and CStack respectively.

For the definition of a topdown tree transducer we refer to [25, 11, 7]. The family of tree transformation languages (i.e. yields of images of the recognizable tree languages under topdown tree transducers) is denoted by $yD_1$, and the subfamily of deterministic tree transformation languages by $ydetD_1$. We note that $ydetD_1$ equals
the class of ranges of generalized syntax directed translations [3].

We now list a few facts taken from the literature, which establish some connections between the above mentioned concepts.

Known facts
(1) ETOL and OI are full hyper-AFLs [8, 9, 26].
(2) ETOL = ELB and EDTOL = LB [9], hence ETOL ⊆ OI.
(3) Stack ⊆ OI [2, 16] and NESTack ⊆ ETOL [22].
(4) ETOL ⊆ yD₁ and EDTOL ⊆ ydetD₁ [12].
(5) All inclusions in (1)-(4) are proper [10, 19, 15].

We finally refer to [15] for the properties P2 and P3 of a language L. Intuitively P2 says that in a string from L one cannot find two nonoverlapping substrings which may be changed into other substrings independently (without leaving L). P3 says that one cannot find two different nonoverlapping substrings that may be used in place of each other (without leaving L). Thus P2 implies P3.

3. The language of cuts
In this section we present a language L₀ which is both in Basic and Stack but not in yD₁ (and hence not in ETOL, cf. section 2). It follows that OI and yD₁ are incomparable. At the end of the section we put several language families into an inclusion diagram.

L₀ will represent the set of all cuts through the infinite binary tree

A cut is a finite nonempty sequence of words over {0, 1} defined recursively as follows: (i) <λ> is a cut, (ii) if <v₁,...,vₖ> and <w₁,...,wₙ> are cuts, then so is <0v₁,...,0vₖ,1w₁,...,1wₙ>. The strings wᵢ in a cut <w₁,...,wₙ> are called nodes. An example of a cut, corresponding to the above picture, is <00, 010, 011, 10, 11>. A cut is also called a complete binary code.

Definition. Let a and b be symbols different from 0 and 1.
L₀ = {aw₁0bw₁aw₂0bw₂...awₙ0bwₙ 1 | <w₁,...,wₙ> is a cut}.

Note that if <w₁,...,wₙ> is a cut, then so is <w₁0,w₁1,...,wₙ0,wₙ1>.

L₀ is generated by the basic macro grammar with rules S → A(λ), A(x) → A(x0)A(x1),
A(x) + ax0bx1.

$L_o$ can easily be recognized by a stack automaton that stores the consecutive nodes of a cut corresponding to a word of $L_o$ in its stack (one at a time).

$L_o$ has property P3 (to see this one needs, apart from the special form of $L_o$, that, for any cut $<w_1,...,w_n>$, the nodes $w_i$ are all different and $\sum_{i=1}^{n} \frac{-|w_i|}{2} = 1$; moreover one needs that for given integers $k_1,...,k_n$ there is at most one cut $<w_1,...,w_n>$ such that $|w_i| = k_i$ for $1 \leq i \leq n$). Theorem 5 of [15] says that any language in $\text{yD}_1$ with property P3 is in $\text{ydetD}_1$. Thus it now suffices to show that $L_o \notin \text{ydetD}_1$. In [24] an intercalation lemma for tree transducer languages is proved that in a straightforward way gives rise to the following intercalation lemma for $\text{ydetD}_1$: for each $L$ in $\text{ydetD}_1$ there is an integer $p$ such that every $z$ in $L$ longer than $p$ can be written as $z = z_1...z_k$ and (i) $|z_i| \leq p$ for all $1 \leq i \leq k$, and (ii) for every $N$ there are strings $v_1,...,v_k$ such that $v_1...v_k \in L$, $|v_1...v_k| > N$ and $\min(v_i) = \min(z_i)$ for all $1 \leq i \leq k$ (where, for a string $w$, $\min(w)$ denotes the set of symbols occurring in $w$). Thus, assuming that $L_o$ is in $\text{ydetD}_1$, every long string of $L_o$ can be divided into small substrings which can be pumped up keeping the same min alphabet. Take $z = aw_10bw_1...aw_nbw_n$ in $L_o$ with $|w_i| \geq p$ for all $1 \leq i \leq n$. Then pumping up $z_1,...,z_k$ can only influence the 0's and 1's. This would give arbitrary long cuts with the same number (2n) of nodes. This is clearly a contradiction. Hence $L_o \notin \text{ydetD}_1$.

We note that the reader interested only in ETOL can use Theorem 1 of [15] instead, and give the (easy) proof for the above intercalation lemma for ETOL.

The existence of $L_o$ solves the problem left open in [15] whether $\text{OI} \subseteq \text{yD}_1$. Hence $\text{OI}$ and $\text{yD}_1$ are incomparable (cf. [10, 15]); in other words, the classes of context-free tree languages and ranges of topdown finite state tree transducers are incomparable even when yields are taken. We conjecture that $L_o$ is not in any $\text{yD}_n$ (i.e. cannot be obtained by the application of any sequence of tree transducers, cf. [7]).

We can now draw the following inclusion diagram, in which $T_d(\text{REG})$ denotes the class of images of the regular languages under deterministic 2-way gsm's (see [21], where it is shown that $T_d(\text{REG}) \subseteq \text{CStack}$).
The inclusions are clear apart from the inclusion $T_d(\text{REG}) \subseteq \text{ETOL}$ which follows from the fact that $T_d(\text{REG})$ is closed under copying [21] and a copying theorem for ETOL (Theorem 1 of [15]). Incomparabilities and proper inclusions follow from

1. $L_O \in (\text{Basic } \cap \text{Stack}) - \text{ETOL}$ (proved above).
2. $\{a^n b^n c^n \mid n \geq 1\} \in \text{ETOL} - \text{Stack}$ (see [23]).
3. $\{w \in (a, b)^* \mid \text{the number of b's in } w \text{ is not prime}\} \in \text{CStack}$ [19], but not in IO; the latter follows by observing that the proof in [16] of the existence of a language in $\text{OI} - \text{IO}$ proves in fact that if $L \subseteq b^*$ and $h^{-1}(L) \in \text{IO}$ (where $h(a) = \lambda$ and $h(b) = b$), then $L$ is regular.
4. the existence of a language in $\text{IO} - \text{OI}$ [16].
5. $\{a^n \mid n \geq 1\} \in \text{NEStack} - \text{CStack}$ [19].

4. Extended basic macro languages

In this section we show several properties of EB, in particular the inclusion of Stack in EB and a result on substitution of EB languages. We first note that by the previous section the following diagram is correct (cf. section 2):

When macro grammars are viewed as nondeterministic recursive program schemes (see [14]), the notion of "extendendness" corresponds to allowing choices (tests) in the parameters of a procedure call. Thus the diagram shows that for nonnested recursive program schemes this feature extends their computational power (independent of linearity).

We extend EB grammars still more as follows. Let RB denote the class of languages generated by basic macro grammars in which union, concatenation and moreover Kleene star are used as operations. Thus regular languages are stored in the arguments of a nonterminal rather than finite ones as in EB.

We now list some facts about EB together with sketches of proof.

1. $\text{RB} = \text{EB}$.

   Proof. Any finite approximation of a regular language can be computed in some additional arguments of a nonterminal.

2. EB is a full AFL.

   Proof. $\cup, \ast, \ast$ as for context-free grammars; regular substitution using (1); $\cap R$ by a standard proof (cf [16]).

3. Stack $\subseteq$ EB.

   Proof. By (1) and (2) it suffices to show that a full AFL-generator of Stack is in RB. The full generator of Stack given in [17, Example 5.3.2] is generated by the following RB grammar which remembers the possible sequences of stack-reading instructions in the argument of T:

$$S \rightarrow T(\lambda), \quad T(x) \rightarrow \lambda,$$
T(x) \rightarrow a(L_{1}a^{R}x)^{T}(aL_{1}a^{R}x)^{-1}a^{F}T(x) \text{ and a similar rule for } b.

(4) Let, for languages $L_1$ and $L_2$ over disjoint alphabets, $\tau_{L_2}(L_1)$ denote the result of substituting $aL_2$ for each symbol $a$ in $L_1$. If $\tau_{L_2}(L_1) \subseteq EB$, then $L_1$ is context-free or $L_2 \in ELB$.

Proof (cf. [19]). Consider the ELB grammar $G'$ obtained by replacing each rule $A(...) \rightarrow B_1(...)B_2(...)B_n(...) \text{ of the } EB \text{ grammar } G \text{ generating } r L_2(L_1)$ by the $n$ rules $A(...) \rightarrow B_1(...)$. Either $L_2$ can be obtained from $L(G')$ by a $2$ finite state transducer producing all words of $L_2$ occurring between symbols of $L_1$ (and $ELB = ETOL$ is a full AFL); or $G$ can be changed into a context-free grammar generating $L_1$, since it only has to remember a finite amount of information in its arguments.

(5) $EB$ is not substitution closed.

Proof. Let $L_1 = \{a^{2n} \mid n \geq 1\} \in EB - CF$ and $L_2 = L_{O}$ from section 2 which is in $EB - ELB$. Then (4) implies that $\tau_{L_2}(L_1) \not\subseteq EB$.

(6) The converse of (4) also holds. In fact $EB$ is closed under substitution into context-free languages and under substitution by ELB languages.

Proof. By straightforward grammatical constructions.

We note that results (4, 5, 6) are similar to those in [19] concerning Stack and CStack.

5. A full hyper-AFL between $EB$ and $OI$

Consider the family $H(EB)$. Since $EB$ is a full AFL, $H(EB)$ is the smallest full hyper-AFL containing $EB$ [4]. Since each full hyper-AFL is substitution closed, it follows that $EB \subseteq H(EB)$. We shall show that $H(EB) \subseteq OI$. The main technique is that of copying as used in [27, 10, 15].

(1) If $L$ has property $P_2$ and $L \in H(EB)$, then $L \in LB$.

Proof. Property $P_2$ forces each $EB$ grammar the language of which is used in the substitutions whose iteration gives $L$, to be linear. Hence $L \in H(EB)$. Since $ELB$ is a full hyper-AFL (cf. section 2), $L \in ELB (= ETOL)$. Hence, by Theorem 1 of [15], $L \in ETOL = LB$.

(2) If $L \in Basic$, then $\{w \# w^R \mid w \in L\} \subseteq OI \cap IO$, where $w^R$ is the reverse of $w$.

Proof. by a grammatical construction in which the sentential form $A_1(s_1)A_2(s_2)\ldots A_n(s_n)$ of the basic macro grammar, where $s_1$ is the sequence of arguments of $A_1$, is represented as $A_1(s_1,s_1',A_2(s_2,s_2',A_3(\ldots,A_n(s_n,s_n')))\ldots)$ in the new grammar, where $s_1'$ contains the reverses of the elements of $s_1$.

(3) $H(EB) \notin OI$.

Proof. Let $L_1 = \{w \# w^R \mid w \in L_0\}$ where $L_0 \in Basic - ETOL$ is the one of section 3. By (2), $L_1 \subseteq OI$. Since $L_1$ has property $P_2$, $L_1 \in H(EB)$ would imply $L_1 \subseteq ETOL$ by (1).

We thus obtain the following diagram.
This diagram solves the question in [16] whether OI is the smallest full AFL containing Basic, it shows the existence of a full hyper-AFL between the full hyper-AFLs ETOL and OI, and it improves the result in [20] that OI cannot be reached from Stack by nested iterated substitution (in fact the proof in [20] shows that the smallest super-AFL containing Stack is properly contained in H(EB)).

Remark 1. An indexed grammar is restricted if after consumption of a flag no new flags can be created (see [1], last page, for a formal definition). It can be shown that the class of restricted indexed languages is equal to EB. The inclusion of Stack in this class was shown in [2]. The result of this section shows that not all indexed languages can be obtained from the restricted indexed languages by hyper-AFL operations.

Remark 2. A reasoning similar to the one in this section shows that IO cannot be reached from Basic by iterated 'deterministic' substitution (cf. [5]; in the notation of that paper: \( \eta(\text{CF}) \not\subseteq \eta(\text{Basic}) \not\subseteq \eta(\text{IO}) \), where \( L_0 \) and \( L_1 \) are the respective counterexamples.

6. Future work

(1) In [22] the cs-pd machine is defined which recognizes precisely ETOL (= ELB). It is a checking stack automaton with a restricted facility of writing on its stack. A good machine for EB is the s-pd machine, which is a stack automaton with the same writing facility (the machine may write on a second track of its stack in a pushdown fashion: the reading head points at the top of pushdown track whereas its bottom is at the top of the stack). This explains the similarity of the results in [19] and those in section 4. The above s-pd machines are the subject of [13].

(2) For any family \( K \) of languages one can define Basic\( (K) \)-grammars similar to extended basic grammars but with languages from \( K \) rather than finite ones. Thus \( \text{EB} = \text{Basic}(\text{FIN}) = \text{RB} = \text{Basic}(\text{REG}) \). Similarly LB\( (K) \)-grammars can be defined. Generalizing the proof in [9] that ETOL = ELB, it follows that under weak restrictions on \( K \), \( \text{LB}(K) = \text{H}(K) \). Let us call a full AFL \( K \) such that Basic\( (K) \subseteq K \), a full basic-AFL (rather than hyper-AFL). It can be shown that the smallest full basic-AFL is properly contained in OI (\( L_1 \) of section 5 being the counterexample). It is conjectured that it is the union of a proper hierarchy of full hyper-AFLs. These "basic extensions" are the subject of [6].

Acknowledgements. I thank Peter Asveld for many useful discussions, Frits Göbel for his helpful knowledge of cuts, and Erik Meineche Schmidt for directing my attention towards stack automata.
References

[16] M.J. Fischer: Grammars with macro-like productions; Doctoral Diss., Harvard University, 1968 (also: 9th SWAT, 131-142).
[23] W. Ogden: Intercalation theorems for stack languages; 1st ACM Symposium on


