On the Power of Subroutines for Finite State Machines

Markus E. Nebel
Johann Wolfgang Goethe-Universität, Frankfurt
Fachbereich Informatik
D-60054 Frankfurt am Main
Germany

Abstract

In this paper we extend the finite state machines by a subroutine concept. Two implementations are considered. The first implementation yields a new class of languages which is a subclass of the context-free languages. The second one leads to an alternative automata-model for the context-free languages. Besides the generative capacity other properties like determinism, reversal languages, etc. are also studied. We prove that determinism for the second implementation is equivalent to the notion of $LL(1)$-languages. The motivation for those observations comes from a description language for plot data called DPF which is used in practice and which possesses simple non-regular constructions only.
Keywords: automata-theory, deterministic parsing, formal languages.

1 Introduction

The generative capacity of finite state machines or alternatively regular expressions, i.e. right-linear grammars, suffices only for a few examples of the application of formal languages. We often need the possibility to express couplings between different parts of a word, e.g. in programming languages where the start of a loop has to be terminated somewhere later in the program. One possibility to obtain such a coupling is to think of a finite state machine with a subroutine concept. In this paper two models are considered on how to extend a normal finite state machine by a subroutine concept.

- **The weak model:** Here, the finite state machine is extended by a stack on which return-addresses (i.e. states) may be stored. There is no real call command for the subroutine-call. After pushing the return address one has to use a normal transition to give the control to the subroutine (see section 3).

- **The strong model:** Here, the weak model is extended by a real call command (see section 4).

In both concepts a final state may by interpreted as a signal to return, i.e. to continue the computation with the state on top of the stack.
Immediately the question arises: What generative capacity do both models imply? In the following sections we will answer this question.

2 Basic Definitions

In this section we give a brief presentation of some basic concepts regarding formal grammars and automata. For details, the reader is referred to [3] and [5]. A *context-free grammar* (CFG) is a four-tuple $G = (I, T, P, S)$ where I and T are finite disjoint sets of *nonterminals* and *terminals*, respectively; $S \in I$ is the *axiom* and P is a finite subset of $I \times (I \cup T)^*$, the set of *productions*. For $(A, \gamma) \in P$ we write $A \to \gamma$. For f being the name of $A \to \gamma$ we write $f : A \to \gamma$. A rule $A \to B$, $A, B \in I$ is called *unit production*. The length of a word w is denoted by $|w|$, the number of symbols a in w is represented by $|w|_a$. The word of length 0 (the empty word) is denoted by ε. If w can be factorized into $w_1w_2w_3$, then w_1 (resp. w_2; w_3) is called a *prefix* (resp. *infix; suffix*) of w; a prefix w_1 (resp. infix w_2; suffix w_3) is *proper* if $w_2w_3 \neq \varepsilon$ (resp. $w_1 \neq \varepsilon$ and $w_3 \neq \varepsilon$; $w_1w_2 \neq \varepsilon$). For any set $\mathcal{A}$ and a natural number k we use $\mathcal{A}_\varepsilon$ as an abbreviation for $\mathcal{A} \cup \{\varepsilon\}$, by $\mathcal{A}^{\leq k}$ we denote $\mathcal{A}^* \backslash \mathcal{A}^* \mathcal{A}^{k+1}$. $\mathcal{A}^0$ denotes the set $\{\varepsilon\}$. For $A \in \mathcal{A}$ we use $A^{\leq k}$ as an abbreviation for $\{A\}^{\leq k}$. A context-free grammar $G = (I, T, P, S)$ is called *right-linear* (resp. *left-linear*) if $P \subseteq I \times T^* I_\varepsilon$ (resp. $P \subseteq I \times I_\varepsilon T^*$) holds. G is called *ε-free* if either $P \subseteq I \times (I \cup T)^+$ or if there is exactly one production $S \to \varepsilon$ with the right-hand side ε and S does not appear on the right-hand side of any production in P. The relation $\Longrightarrow \subseteq (I \cup T)^* \times (I \cup T)^*$ is defined as follows. For any $\alpha, \beta \in (I \cup T)^*$, $\alpha \Longrightarrow \beta$ if and only if $\alpha = \alpha_1 A \alpha_2$, $\beta = \alpha_1 \gamma \alpha_2$ and $A \to \gamma \in P$. If $\alpha_1 \in T^*$ we write $\alpha \overset{lm}{\Longrightarrow} \beta$. In the usual way we denote the transitive and the reflexive-transitive closure of $\Longrightarrow$ by $\Longrightarrow^+$ and $\Longrightarrow^*$, respectively; the same can be done for $\overset{lm}{\Longrightarrow}$. We write $\alpha \Longrightarrow^k \beta$ ($\alpha \overset{lm}{\Longrightarrow}^k \beta$) if there is a sequence $\alpha_0, \alpha_1, \ldots, \alpha_k$ of $k+1$ strings such that $\alpha = \alpha_0$, $\alpha_{i-1} \Longrightarrow \alpha_i$ ($\alpha_{i-1} \overset{lm}{\Longrightarrow} \alpha_i$) for $1 \leq i \leq k$ and $\alpha_k = \beta$. This sequence of strings is called a *derivation of length k* of β from α. A string $x \in (I \cup T)^*$ is said to be a *sentential form* if $S \Longrightarrow^* x$. The set $\mathcal{L}(G) = \{x \in T^* \mid S \Longrightarrow^* x\}$ is said to be the *language generated by G*; the set of all languages that can be generated by a context-free grammar is denoted by CFL. Two grammars are called *equivalent* if they generate the same language. A formal language L is called *ε-free* if $\varepsilon \notin L$. The *transpose-operator* $.^{\mathcal{T}}$ is defined by $(xa)^{\mathcal{T}} := a(x^{\mathcal{T}})$ for $x \in T^*$ and $a \in T$. For any language $\mathcal{L}$ we use $\mathcal{L}^{\mathcal{T}}$ as an abbreviation for $\{w^{\mathcal{T}} \mid w \in \mathcal{L}\}$. A set M of languages is said to be *closed under the transpose operation* if and only if $\mathcal{L} \in M$ implies $\mathcal{L}^{\mathcal{T}} \in M$. A

CFG is called *reduced* if and only if $P = \emptyset$ or $S \Longrightarrow^* \alpha_1 A \alpha_2$ and $A \Longrightarrow^* \alpha$ for all $A \in I \cup T$ and some $\alpha_1, \alpha_2 \in (I \cup T)^*$, $\alpha \in T^*$. It is called $LL(k)$ [1] for some fixed integer k, if whenever there are two leftmost derivations $S \overset{lm}{\Longrightarrow}^* wA\alpha \overset{lm}{\Longrightarrow} w\beta\alpha \overset{lm}{\Longrightarrow}^* wx$ and $S \overset{lm}{\Longrightarrow}^* wA\alpha \overset{lm}{\Longrightarrow} w\gamma\alpha \overset{lm}{\Longrightarrow}^* wy$ such that $\mathsf{FIRST}_k(x) = \mathsf{FIRST}_k(y)$ it follows that $\beta = \gamma$. Here, $\mathsf{FIRST}_k(v) := \{x \in T^* \mid (v \overset{lm}{\Longrightarrow}^* x\gamma \wedge |x| = k) \vee (v \overset{lm}{\Longrightarrow}^* x \wedge |x| < k)\}$ for any $v \in (I \cup T)^*$.
A *finite automaton* is a five-tuple $\mathsf{A} = (Z, \Sigma, \delta, z_0, F)$ where

1. Z is a finite set of *states*;
2. Σ is a finite set of *input-symbols*;
3. $\delta \subseteq (Z \times \Sigma_\varepsilon) \times Z$, is the *state transition relation*;
4. $z_0 \in Z$ is the *initial state*;
5. $F \subseteq Z$ is the set of *final states*.

Note, that δ can be a partial relation so that the finite automaton is *incompletely specified*. As usual, a finite automaton is called *deterministic* if $|\delta(z, a)| \leq 1$ for all $(z, a) \in Z \times \Sigma_\varepsilon$, and if $\delta(z, \varepsilon) \neq \emptyset$ implies that for all $a \in \Sigma$ we have $\delta(z, a) = \emptyset$, $z \in Z$. For the sake of convenience, the natural extension $\hat{\delta}$ of δ to a relation from $Z \times \Sigma^*$ to Z is also denoted by δ. The *language recognized by* A is defined by $L(\mathsf{A}) = \{w \in \Sigma^* \mid \delta(z_0, w) \cap F \neq \emptyset\}$.
A triple (z, a, z') is called *a-transition* if $z' \in \delta(z, a)$ holds.
We will now give the definitions for the weak model of a finite automaton with a subroutine concept.

3 The Weak Model

This section introduces the weak model of a finite automaton with subroutines. We will consider both, the notion of grammars and the notion of automata.

Definition 1 *A* finite state machine of degree k *is a five-tuple* $\mathsf{A} = (Z, T, \delta, z_0, F)$ *with*

- Z *is a finite set of states;*
- T *is the input alphabet;*
- $\delta \subseteq (Z \times T_\varepsilon) \times (Z \times [0 : k-1])$*;*
- $z_0 \in Z$ *is the initial state and*
- $F \subseteq Z$ *is the set of all final states.*

The set of all finite state machines A *of degree* k *is denoted by* $FSM(k)$.

Note, that $\mathsf{A} \in FSM(1)$ is nothing else but a finite automaton as defined in Section 2. Before we comment on this definition in detail we introduce a formal description of the computation performed by an automaton in $FSM(k)$.

Definition 2 *Let* $\mathsf{A} = (Z, T, \delta, z_0, F)$ *be in* $FSM(k)$. *A triple* $(z, w, s) \in Z \times T^* \times Z^*$ *is called* configuration. *The triple* (z_0, w, ε) *is called* start-configuration with input w. *A triple* $(z, \varepsilon, \varepsilon)$ *is called* end-configuration *if* $z \in F$.

The first component of a configuration is the actual state. The second component contains the input which has not been read yet and the third component represents a stack of states (with the top of the stack to the right).

Definition 3 *Let* $\mathsf{A} = (Z, T, \delta, z_0, F)$ *be in* $FSM(k)$. *A* move *of* A *is represented by the relation* $\vdash \subseteq (Z \times T^* \times Z^*)^2$ *with* $(z, av, s) \vdash (z', v, s')$, $a \in T_\varepsilon$ *if and only if*

$$((z, a), (z', j)) \in \delta \text{ and } s' = s(z')^j$$

or

$$a = \varepsilon,\ s = s'z' \text{ and } z \in F.$$

A move with respect to the first alternative is called a-transition, *a move corresponding to the second one will be called* pop-operation.

Thus an automaton in $FSM(k)$ is an ordinary finite state machine with a stack on which states can be stored. For every transition it is possible to store the state just reached at most $k-1$ times on this stack. If a final state is reached the computation can be continued with the state on top of the stack. We say that *state* z' *can be reached from state* z if $(z, \alpha, \beta) \vdash^* (z', \alpha', \beta')$ for some $\alpha, \alpha', \beta, \beta'$.
Now we are able to define the set of words accepted by an automaton in $FSM(k)$.

Definition 4 *Let* $\mathsf{A} = (Z, T, \delta, z_0, F)$ *be in* $FSM(k)$. *The set* $L(\mathsf{A}) := \{w \in T^* \mid (z_0, w, \varepsilon) \vdash^* (z, \varepsilon, \varepsilon) \wedge z \in F\}$ *is called the* language accepted by A.

Compared with the ordinary finite state machine the notion of determinism is more complicated in our case. We have a nondeterministic alternative if state z is on the stack, all states that were pushed after z and that are still on the stack are final states, an a-transition (or an ε-transition) for z exists and we reach a final state z' for which an a-transition is defined. Then there are two possibilities for reading a (or we do not know if we should take the ε-transition) since the automaton can use a chain of *returns* in order to go to state z. Together with the ordinary conditions for being deterministic we get:

Definition 5 *Let* A *be in* $FSM(k)$. A *is said to be* deterministic *if the following conditions hold:*

- $(\forall (q, a) \in Z \times T_\varepsilon) : (((q, a), (q', i)) \in \delta \wedge ((q, a), (q'', j)) \in \delta \rightsquigarrow q' = q'' \wedge i = j)$;

Figure 1: An automaton in $FSM(2)$ which accepts a simple programming language.

- $((q,\varepsilon),(q',i)) \in \delta \rightsquigarrow (\neg\exists(a,q'',j) \in T \times Z \times [0:k-1]) :$
  $(((q,a),(q'',j)) \in \delta);$
- $(\forall w \in T^*,\ a,b \in T_\varepsilon,\ z,z' \in Z,\ s,s' \in Z^*) : ((z_0,w,\varepsilon) \vdash^* (z',\varepsilon,s'zs) \rightsquigarrow$
  $z' \notin F \vee s \notin F^* \vee \delta(z',a) = \emptyset \vee \delta(z,b) = \emptyset \vee (a \neq b \wedge a \neq \varepsilon \wedge b \neq \varepsilon)).$

Here $\delta(z,a)$, $a \in T_\varepsilon$, is used to denote all possible a-transitions for state z.

Note, that the third condition of this definition is decidable. Let us have a look at an example. The automaton $\mathsf{A} = (Z,T,\delta,z_0,F)$ with $Z = \{1,\ 2,\ 3\}$, $T = \{\mathbf{begin},\ \mathbf{loop},\ \mathbf{end}\}$, $z_0 = 1$, $F = \{3\}$ and $\delta = \{((1,\mathbf{begin}),(2,0)),$ $((2,\mathbf{loop}),(2,1)),\ ((2,\mathbf{end}),(3,0))\}$ is in $FSM(2)$. The automaton A is deterministic, a graphical representation can be found in Figure 1. The language accepted by A is the set of all *programs* which have a body (**begin end**) and arbitrary many (possibly involved) loop-statements which have a **loop**-token at their beginning and which are also terminated by **end**. For $w =$ **begin loop loop end end end** as input, A's computation starts with the configuration $(1,w,\varepsilon)$. We only have one **begin**-transition. Thus

$$(1,\mathbf{begin\ loop\ loop\ end\ end\ end},\varepsilon) \vdash (2,\mathbf{loop\ loop\ end\ end\ end},\varepsilon).$$

The remaining input begins with two **loop**-symbols which have to be processed by the transition $((2,\mathbf{loop}),(2,1))$. We get

$$(2,\mathbf{loop\ loop\ end\ end\ end},\varepsilon) \vdash^2 (2,\mathbf{end\ end\ end},2\cdot 2).$$

To process the following **end** we use the transition $((2,\mathbf{end}),(3,0))$ to switch to state 3. Since state 3 has no outgoing transition, the only possibility to continue the computation is to use the state on top of the stack in order to return to state 2 (which is possible, because state 3 is an accepting state). In this way we get $(2,\mathbf{end\ end\ end},2\cdot 2) \vdash^2 (2,\mathbf{end\ end},2)$. The same has to be done for the next input symbol: $(2,\mathbf{end\ end},2) \vdash^2$ $(2,\mathbf{end},\varepsilon)$. Again, there is no alternative. Since we only have one **end**-transition, the automaton has to proceed the move $(2,\mathbf{end},\varepsilon) \vdash (3,\varepsilon,\varepsilon)$ and the input is accepted.
Note, that a further transition, e.g. $((2,\mathbf{while}),(2,1))$ accompanied by the corresponding change of the alphabet T, might be used in order to enlarge the set of supported control-statements.

Definition 6 *Let $G = (I,T,P,S)$ be a context-free grammar. G is said to be* k-right-linear *(or alternatively* a right-linear grammar of degree k*) if $P \subseteq I \times T^* \bigcup_{A\in I} A^{\leq k}$. The set of all k-right-linear grammars is denoted by $RLIN(k)$.*

The motivation for this definition is the following relation between right-linear grammars and finite automata: Both have the same generative capacity and we know how to translate a right-linear grammar into an equivalent finite automaton, i.e. into an automaton that accepts the language generated by the grammar. A production $A \to vB$ within the grammar is translated into a transition[1] from state A to state B by reading v. With respect to the notion of subroutines the duplication of the nonterminal on the right-hand side of a right-linear production rule corresponds to the storage of the actual state for a later return (assuming a call to be implemented as a normal transition from the state just reached to the target of the jump).
The definition of the class of *k-left-linear* grammars $LLIN(k)$ is straightforward.
By $RREG(k)$ we denote the set of all languages that can be generated by a grammar out of $RLIN(k)$ (analogously $LREG(k)$ for $LLIN(k)$). Recall that $RLIN(1)$ resembles the right-linear grammars as introduced in Section 2. Thus $RREG(1)$ is equal to the set of all regular languages REG. Since REG is closed under the transpose operation we also have $RREG(1) = LREG(1)$. We define $REG(k) := RREG(k) \cup LREG(k)$ which is closed under the transpose operator by definition.

3.1 Fundamental Results

In this section the class $FSM(k)$ is investigated in detail. We will consider the generative capacity as well as other properties like the closure under the transpose operation.

Theorem 1 $RLIN(k) = LLIN(k)^{\mathcal{T}}$, $RLIN(k)^{\mathcal{T}} = LLIN(k)$, $k \in \mathbb{N}$.

Proof: The proof is a trivial consequence of the application of the usual method for reversing the language generated by a grammar. With this method, simply the right-hand side of any production rule is transposed. Thus, in our case, a grammar in $RLIN(k)$ becomes a grammar in $LLIN(k)$ and vice versa.

Theorem 2 *For every grammar $G = (I, T, P, S)$ in $RLIN(k)$ there is an automaton A in $FSM(k)$ with $\mathcal{L}(G) = L(\mathsf{A})$ and vice versa.*

Proof: "$\curvearrowright$:" We assume P to fulfil $P \subseteq I \times T_\varepsilon \bigcup_{B \in I} B^{\leq k}$ which is no restriction as we will see. Consider the rule $C \to a_1 a_2 \cdots a_m B^j$, $a_i \in T$, $1 \leq i \leq m$, $j \leq k$. We just have to replace this rule by the set of rules $C \to a_1 H_1$, $H_m \to a_{i+1} H_{i+1}$, $1 \leq i \leq m-2$, and $H_{m-1} \to a_m B^j$. There, the H_i, $1 \leq i \leq m-1$, are new nonterminals. In order to prove the theorem define $\mathsf{A} = (Z, T, \delta, z_0, F)$ as follows:

[1] Note, that if v consists of more than one nonterminal then the production has to be translated into a chain of transitions.

- $Z := I \dot{\cup} \{z_e\}$,
- $z_0 := S$,
- $F := \{z_e\} \cup \{z \in Z \mid z \to \varepsilon \in P\}$,
- $\delta := \{((z,a),(\tilde{z},i)) \mid z \to a\tilde{z}^{i+1} \in P,\ a \in T_\varepsilon,\ \tilde{z}^{i+1} \neq \varepsilon\} \cup$
  $\{((z,a),(z_e,0)) \mid z \to a \in P,\ a \in T\}$.

It suffices to prove that, if $S = \alpha_0 \overset{lm}{\Longrightarrow}{}^{k} \alpha_k = w\gamma'$, $w \in T^*$, $\gamma' \in I^*$, is a leftmost derivation in G, then for A the relation $(z_0, w, \varepsilon) \vdash^* (z, \varepsilon, \gamma)$, $z\gamma^{\mathcal{T}} = \gamma'$, holds. We prove this by induction on k.

$k = 0$: We have $S \overset{lm}{\Longrightarrow}{}^{0} S$ and $(z_0, \varepsilon, \varepsilon) \vdash^0 (z_0, \varepsilon, \varepsilon)$.

$k \rightsquigarrow k+1$: Consider $S = \alpha_0 \overset{lm}{\Longrightarrow}{}^{k} \alpha_k \overset{lm}{\Longrightarrow} \alpha_{k+1} = w\gamma$, $w \in T^*$, $\gamma \in I^*$. By hypothesis there are $w' \in T^*$ and $\gamma' \in I^*$ such that $\alpha_0 \overset{lm}{\Longrightarrow}{}^{k} \alpha_k = w'\gamma'$ and $(z_0, w', \varepsilon) \vdash^* (z, \varepsilon, \bar{\gamma})$ with $z\bar{\gamma}^{\mathcal{T}} = \gamma'$. But since $w = w'a$ for $a \in T_\varepsilon$ also $(z_0, w, \varepsilon) \vdash^* (z, a, \bar{\gamma})$ holds. We have to distinguish between two cases.

1. case: $\alpha_k \overset{lm}{\Longrightarrow} \alpha_{k+1}$ corresponds to the application of an ε-production. In that case $\alpha_k = wz\gamma$ holds and by construction z is a final state. But then $(z, \varepsilon, \bar{\gamma}) \vdash (z_t, \varepsilon, \tilde{\gamma})$ with $\bar{\gamma} = \tilde{\gamma}z_t$ by popping. Now $z\bar{\gamma}^{\mathcal{T}} = \gamma'$ implies $z_t\tilde{\gamma}^{\mathcal{T}} = \gamma$ as desired.

2. case: $\alpha_k \overset{lm}{\Longrightarrow} \alpha_{k+1}$ corresponds to the application of production $f : z \to a\tilde{z}^i$, $a\tilde{z}^i \neq \varepsilon$. If $i > 0$ we have a transition $((z,a),(\tilde{z},i-1))$ for A and thus $(z, a, \bar{\gamma}) \vdash (\tilde{z}, \varepsilon, \bar{\gamma}\tilde{z}^{i-1})$ while the application of f implies $\alpha_{k+1} = w'a\tilde{z}^i\bar{\gamma}^{\mathcal{T}}$. If $i = 0$ then we have the transition $((z,a),(z_e,0))$ for $z_e \in F$ which implies $(z, a, \bar{\gamma}) \vdash (z_e, \varepsilon, \bar{\gamma}) \vdash (z_t, \varepsilon, \tilde{\gamma})$ with $\bar{\gamma} = \tilde{\gamma}z_t$.

In both cases we have the correspondence of sentential form and configuration as claimed.

"$\curvearrowleft$:"Define $G = (I, T, P, S)$ as follows:

- $I := Z$,
- $S := z_0$,
- $P := \{B \to aC^j \mid ((B,a),(C,j-1)) \in \delta\} \cup \{B \to \varepsilon \mid B \in F\}$.

An analogous induction yields the theorem.

Figure 2 shows an example of a grammar with its corresponding automaton.

Figure 2: Example of a grammar and its corresponding automaton.

Remark 1 Note, that the automaton is not constrained to return to the state on top of its stack if a final state is reached. It also has the possibility to leave a final state by means of a usual transition (if specified)

without popping a state from the stack. In order to obtain the above correspondence between grammars and automata it is not possible to use a more restrictive automata-model in which the control must return to the state on top of the stack every time the automaton reaches a final state. To simulate that restrictive model by a grammar, we would have to restrict the application of some productions in dependence on the number of nonterminals in the actual sentential form as follows: If the stack is not empty, i.e. we have more than one nonterminal in the sentential form, then only the ε-production which simulates the *return* may be applied. If the stack is empty all productions with the corresponding nonterminal on the left-hand side may be applied.

Before we prove a pumping-lemma, we introduce a normal form which sometimes makes it easier to show our statements.

Lemma 1 (normal form) *Let $G = (I, T, P, S) \in RLIN(k)$. Then there is an equivalent grammar $G' = (I', T', P', S') \in RLIN(k)$ with $P' \subseteq I' \times \bigcup_{B \in I'} B^{\leq k} \cup I' \times T^* I_\varepsilon$.*

Proof: The only production rules in P that do not fit into this normal form are those in $\mathcal{P} := I \times T^+ \bigcup_{\substack{B \in I \\ 2 \leq i \leq k}} B^i$. If $f : A \to vB^j \in \mathcal{P}$ is in P we introduce a new nonterminal H and split f into two productions $f_1 : A \to vH$ and $f_2 : H \to B^j$ which obviously simulate f. In this way all productions in P that do belong to $\mathcal{P}$ can be eliminated.

Lemma 2 *For every right-linear grammar of degree k there is an equivalent grammar in $RLIN(k)$ which is ε-free and does not contain any unit production.*

Proof: The well-known construction [1, Algorithm 2.10] which generates an ε-free production system for context-free grammars can be applied. Since this construction only erases nonterminals on the right-hand side of the production rules, it cannot happen that the number of nonterminals becomes larger than k or that we get different nonterminals on the right-hand side of any rule. Thus, the restrictions for $RLIN(k)$ remain fulfilled. Afterwards, it is possible to erase unit-productions by applying [1, Algorithm 2.11]. Since this algorithm only changes the left-hand sides of the productions the grammar remains ε-free and the restrictions for $RLIN(k)$ remain fulfilled.

No ε-rules and no unit productions are generated if we apply our construction for the normal form of Lemma 1. Thus we can always assume a grammar in $RREG(k)$ to be ε-free, to be in normal form and to have no unit productions.

Lemma 3 *For every $\mathsf{A} \in FSM(k)$, $\mathsf{A} = (Z, \Sigma, \delta, z_0, F)$, there is a $m \in \mathbb{N}$ with*

$$(\forall w \in L(\mathsf{A}), |w| > m) : ((z_0, w, \varepsilon) \vdash^* (z, u, s) \vdash^+ (z, v, ss'),\ s' \in Z^*),$$

Figure 3: Maximal path without backward transitions.

i.e. we have a cycle with either constant or growing stacksize and the contents of the stack which is present at the beginning of the cycle is not affected while processing the cycle.

Proof: We assume that every state of A can be reached from state z_0 since otherwise it could be deleted without changing the language accepted by A. Now, consider the longest path $\mathcal{P}$ in the transition graph of A from z_0 to any other state such that $\mathcal{P}$ has no loops. Let $z_0, z_1, \ldots, z_{\ell+1}$ be the sequence of states on $\mathcal{P}$ and let $((z_i, a_i), (z_{i+1}, p_i))$, $0 \leq i \leq \ell$, be the transitions represented by the edges on that path. Our goal is to find the longest possible computation which uses transitions of $\mathcal{P}$ only. Thus we assume that $z_{\ell+1}$ is the only accepting state of $\mathcal{P}$ since every computation that uses state z_j, $j < \ell + 1$, in order to return to the state z_s on top of the stack could be lengthened by using the transitions which connect z_j and $z_{\ell+1}$ on $\mathcal{P}$ first and then returning to z_s by means of $z_{\ell+1}$. If we assume that $p_i = 0$, $0 \leq i \leq \ell$, then $(z_0, a_0 a_1 \cdots a_\ell, \varepsilon) \vdash^\ell (z_{\ell+1}, \varepsilon, \varepsilon)$ is the longest possible computation if we only use the states and transitions of $\mathcal{P}$. We can process a longer input-string by assuming that some of the p_i's are > 0. Further it is obvious that the larger a p_i becomes, the longer the possible computation gets. The same observation holds for the number of p_i's that are greater than 0. Thus a path $\mathcal{P}$ that maximizes the length of a possible computation looks like the one shown in Figure 3. Note, that it makes no sense to assume $p_\ell > 0$ since this would imply $(z_{\ell+1}, v, s z_{\ell+1}^{p_i}) \vdash^{p_\ell} (z_{\ell+1}, v, s)$ i.e. p_ℓ times a return from state $z_{\ell+1}$ to state $z_{\ell+1}$ without processing any symbol. In the case of Figure 3, for suitable choices of w, w_1 and w_2, $(z_0, w, \varepsilon) \vdash^{\ell+1} (z_{\ell+1}, w_1, z_1^{k-1} z_2^{k-1} \cdots z_\ell^{k-1}) \vdash^{2(k-1)} (z_\ell, w_2, z_1^{k-1} z_2^{k-1} \cdots z_{\ell-1}^{k-1}) \vdash^3 (z_{\ell+1}, w_2, z_1^{k-1} z_2^{k-1} \cdots z_{\ell-1}^{k-2} z_\ell^{k-1}) \vdash \cdots \vdash (z_{\ell+1}, \varepsilon, \varepsilon)$. If we count the number of symbols a_i, $0 \leq i \leq \ell$, that could be read during this computation[2], this number follows the well-known Horner scheme in the variable k. Thus the maximal length of a word that could be processed by a computation using transitions of $\mathcal{P}$ only is given by $\sum_{0 \leq i \leq \ell} k^i = \frac{k^{\ell+1}-1}{k-1} =: m$. Since $\mathcal{P}$ is maximal with respect to its length, any word that is longer than m can only be accepted if there is a transition $((z_\alpha, c), (z_\beta, p))$ with $\alpha \geq \beta$. It is not hard to see that such a transition always implies a cycle with the required properties.

Remark 2

- *If we consider grammars, Lemma 3 says that we cannot derive a word w of arbitrary length without the repetition of at least one nonterminal in the sentential forms for the production of w.*

[2]This number is equal to the number of moves applied within the corresponding computation that do not represent a return to a state on top of the stack.

Figure 4: Derivation Tree T.

- *By l'Hopital's rule* $\lim_{k\to 1} \frac{k^{\ell+1}-1}{k-1} = \ell + 1$ *which equals the case of the usual finite state machine.*

We now give a pumping-lemma for the languages of $RREG(k)$.

Theorem 3 (Pumping-Lemma) *Let $L \in RREG(k)$. A constant $m \in \mathbb{N}$ exists such that for any $z \in L$ with $|z| \geq m$ and with at least m positions of z marked, there is a decomposition $z = uvwxy$ with the following properties:*

a) at least one position of w is marked,

b) either both u and v or both x and y have marked positions,

c) vwx has at most m positions marked,

d) $(\exists l \in [1:k] \wedge \exists i, j \in \mathbb{N}_0,\ i + j + 1 = l)$ with

- *if $l = 1$ then $x = \varepsilon$ and $(\forall \rho \in \mathbb{N}_0) : (z' = uv^\rho wy \in L)$.*
- *if $l > 1$ then v, x can be factorized into $v = v_b v_1 v_2 \cdots v_i v_e$, $v_b, v_e \in T^*$, $v_\nu \in T^+$, $1 \leq \nu \leq i$, and $x = x_b x_1 x_2 \cdots x_j x_e$, $x_b, x_e \in T^*$, $x_\mu \in T^+$, $1 \leq \mu \leq j$, such that $(\forall \rho \in \mathbb{N}_0) : (z' = upy \in L$ for an infix $p \in \bar{v}_\rho$ where $\bar{v}_n := v_b \mathcal{C}_n^l x_e$, $0 < n \leq \rho$, $\bar{v}_0 = \{w\}$ and $\mathcal{C}_n := \left(\bigcup_{0\leq\alpha<n} v_e \bar{v}_\alpha x_b\right) \cup \{v_1, v_2, \ldots, v_i, x_1, x_2, \ldots, x_j\}$.*

Proof: We may assume that L is generated by a grammar in $RREG(k)$, which is ε-free. Further, since every grammar in $RREG(k)$ is context-free, Ogden's Lemma (see [1]) implies the existence of a constant $m \in \mathbb{N}$ with:

i) For any $z \in L$ with $|z| \geq m$ and with at least m positions of z marked there is a factorization $z = uvwxy$ with the properties a), b) and c) of our theorem; and

ii) a nonterminal A exists such that $S \Longrightarrow^+ uAy \Longrightarrow^+ uvAxy \Longrightarrow^+ \cdots \Longrightarrow^+ uv^i wx^i y$ for all integers i including 0.

Thus, we only have to prove statement d). The proof of Ogden's Lemma is based on the construction of a path in the derivation tree T of a sufficiently large word z as shown in Figure 4. By constructing the path in dependence on the marked positions one can prove the statements above. Let $\mathbb{P}$ denote the set of all productions applied within $uAy \Longrightarrow^+ uvAxy$ and let $\mathbb{P}_i$ denote the subset of $\mathbb{P}$ which consists of all productions with i nonterminals on the right-hand side. Since we consider a grammar in $RREG(k)$, all productions in $\mathbb{P}$ must be out of $I \times T^* \bigcup_{B\in I} B^{\leq k}$ and there is a maximum $l \in [1:k]$ for which $\mathbb{P}_i = \emptyset$, $i > l$ holds. We isolate one production in $\mathbb{P}_l$, say

$f : C \to v'B^l$, as shown in Figure 4. One of the B's, say the $(i+1)$-th, must be used to derive the second appearance of A. Thus, we have $B \Longrightarrow^+ v_e A x_b$. The first up to the i-th B are together responsible for the derivation of the rest of v, i.e. $v_1 v_2 \cdots v_i$. Thus v has a decomposition $v = v_b v_1 v_1 \cdots v_i v_e$ and since G is assumed to be ε-free none of the v_ν is empty, $1 \leq \nu \leq i$. The rest of x, i.e. $x_1 x_2 \cdots x_j$, is derived from the $(i+2)$-th B and all those to the right of it. Thus, $x = x_b x_1 x_2 \cdots x_j x_e$, where $i + j + 1$ must be equal to the number of B's in the right-hand side of production f. Again, the fact that G is ε-free guarantees the existence of all the x_μ. Note, that if $l = 1$ also $x_b = x_e = \varepsilon$ since we assume l to be maximal and thus, it is not possible to generate x_b and x_e. If we set $l := i + j + 1$, these derivations imply the following set of equations (see [5] for details)

$$A = v_b B^l x_e + w,\ B = v_e A x_b + v_1 + v_2 + \ldots + v_i + x_1 + x_2 + \ldots + x_j. \quad (1)$$

If we now solve the set of equations (1) we obtain the statement of the lemma.

Remark 3

- *In the case $k = 1$ our pumping-lemma leads to the well-known pumping-lemma for regular languages since l must be 1 and therefore only the first case of statement d) applies.*
- *For $k > 1$ we have a refinement of the context-free pumping-lemma where we can conclude that $z' = u(v_b v_1 \cdots v_i v_e)^q w (x_b x_1 \cdots x_j x_e)^q y$ is in the language for any $q \in \mathbb{N}_0$. But this is only one possibility for $z' = upy$ in statement d) of our pumping-lemma. Thus, we have to expect that a grammar in $RLIN(k)$ cannot generate certain dependences within a language that can be generated by a context-free grammar. Therefore, we have to presume that $RLIN(k)$ is a proper subset of CFL.*

The last general result is an interchange lemma which, as the pumping-lemma, can be used to prove that certain languages do not belong to $RREG(k)$.

Lemma 4 (Interchange Lemma) *Let L be a language in $RREG(k)$. Either $L \in REG$ or there is a $w \in L$ which can be factorized into $w = xy_1 y_2 \cdots y_n$, $x \in T^*$, $y_j \in T^+$, $1 \leq j \leq n$, $2 \leq n \leq k$, such that $xy_{i_1} y_{i_2} \cdots y_{i_n}$, $i_j \in \{1, 2, \ldots, n\}$, $1 \leq j \leq n$, is in L, too.*

Proof: We can assume that L is generated by a grammar $G \in RLIN(k)$ which is ε-free, does not have unit productions and is in normal form. If $L \notin REG$ then there is at least one word $w \in L$ which needs the application of at least one production of the form $A \to B^n$, $2 \leq n \leq k$. Let f be the first of these in a derivation of w, i.e. $S \Longrightarrow^* xA \Longrightarrow^f xB^n$ where within

$S \Longrightarrow^* xA$ only right-linear production rules are applied and thus $x \in T^*$. Now each of the B's has to generate at least one terminal symbol (G is assumed to be ε-free), i.e. $B \overset{lm}{\Longrightarrow}^+ y_i$, $y_i \in T^+$, $1 \leq i \leq n$, and thus $S \overset{lm}{\Longrightarrow}^* xB^n \overset{lm}{\Longrightarrow}^+ xy_1y_2 \cdots y_n = w$. Further, each of the B's might be used to derive any of the y_i, $1 \leq i \leq n$, which proves the lemma.

Corollary 1 *Let T be an alphabet, $L \subseteq T^*$ and $\$ \notin T$. $L \cdot \{\$\} \in RREG(k)$, $k \geq 1$, if and only if $L \in REG$.*

Proof: "$\curvearrowright$:"Assume $L \notin REG$ which obviously implies $L \cdot \{\$\} \notin REG$. By the interchange lemma we can conclude that there is a $w \in L \cdot \{\$\}$ which can be factorized into $w = xy_1y_2 \cdots y_n$, $n \geq 2$, such that any permutation of the y_i's yields a word in $L \cdot \{\$\}$ again. But then, e.g. $\bar{w} = xy_ny_1y_2 \cdots y_{n-1}$ is a word in $L \cdot \{\$\}$ which is a contradiction since it is impossible that symbol \$ is in any other position than the last one.
"$\curvearrowleft$:"If $L \in REG$ we know that there is a left-linear grammar $G = (I, T, P, S) \in LLIN(1)$ with $\mathcal{L}(G) = L$. We can construct a left-linear G' for $L\$$ by introducing a new axiom S' together with the production $S' \to S\$$. Thus, $L \cdot \{\$\} \in REG = RREG(1)$.

3.2 Results on the Hierarchy

First, we show that $REG \subset REG(k)$, $k \geq 2$, by giving an example.

Theorem 4 *$REG \subset REG(k)$ for all $k \geq 2$.*

Proof: Consider the language $L := \{w \in \{a,b\}^+ \mid |w|_a = |w|_b - 1 \wedge |v|_a \geq |v|_b \text{ for all proper prefixes } v \text{ of } w\}$. L is generated by the grammar $S \to aSS$, $S \to b$ which is $RLIN(k)$ for all $k \geq 2$. It is obvious how to apply the pumping-lemma for regular languages to prove that this language is not regular, e.g. by considering the word $a^nb^{n+1} \in L$.

Theorem 5 *$RREG(k) \neq RREG(k)^{\mathcal{T}}$ for every $k \geq 2$.*

Proof: Let T be an alphabet, $\$ \notin T$, and let $L \subseteq T^*$ be a non-regular language in $RREG(k)$, $k \geq 2$. Obviously, $L \in RREG(k)$ implies $\{\$\} \cdot L \in RREG(k)$. Now assume that $RREG(k) = RREG(k)^{\mathcal{T}}$. Then $L^{\mathcal{T}} \cdot \{\$\} \in RREG(k)$ and by Corollary 1 we have $L^{\mathcal{T}} \in REG$ which is in contradiction to our assumptions.

Remark 4 One example for a language that fits the assumptions of the previous proof is generated by the set of productions $\{S \to \$A,\ A \to aA^k,\ A \to b\}$. Even if this language is very similar to the language $\mathcal{L}$ generated by $P = \{S \to aSS,\ S \to b\}$, the transposition of $\mathcal{L}$ is in $RREG(2)$. It is generated by $P = \{S \to bA,\ A \to bB,\ A \to \varepsilon,\ B \to aA,\ B \to bBB\}$.

A slight modification of the grammar used in the proof of Theorem 4 shows that $REG(2)$ contains some well-known context-free languages.

Theorem 6 *The grammar* $G = (\{S, X\}, \{(,)\}, \{S \to (X, X \to (XX, X \to)S, S \to \varepsilon\}, S)$ *is in* $RLIN(2)$ *and generates the semi Dyck-language with one type of brackets.* □

This result shows that the possibility to store the actual state for a later return is very powerful. But it is not sufficient to get the whole set of context-free languages as we will learn by the following theorem.

Theorem 7 *The context-free language* $L = \{a^n b^n \mid n \in \mathbb{N}\}$ *is not in* $RREG(k)$ *for all* k.

Proof: This proof is an example for the usage of the above pumping-lemma. For a given constant m we choose the word $z = a^m b^m$ which is in L and we assume all positions to be marked. The pumping-lemma tells us that for any factorization of $z = uvwxy$ according to the items a) to c), also $z' = uwy \in L$; hence $vx = a^\varphi b^\varphi$, $\varphi \geq 1$, must hold because any other choice for vx would force an odd reduction of the number of a's and the number of b's.
1. case: $l = 1$, i.e. $v = v_b v_e$, $x = \varepsilon$.
For $l = 1$ we may choose any $p \in v^i w$, $i \geq 0$. $x = \varepsilon$ implies $v = a^\varphi b^\varphi$, $\varphi \geq 1$, from which $w \in b^+$ follows since we can choose $p = a^\varphi b^\varphi w$. We now come to a contradiction by choosing $z' = uvvwy$.
2. case: $l \geq 2$, i.e. $i > 0$ or $j > 0$.
We set $\rho = 2$ and choose $p = v_b(v_e v w x x_b)^l x_e$ which is a valid choice since $v_b(v_e v w x x_b)^l x_e \in v_b(v_e \bar{v}_1 x_b)^l x_e \subseteq v_b \mathcal{C}_2^l x_e = \bar{v}_\rho$ holds. From $vx = a^\varphi b^\varphi$, $\varphi \geq 1$, we conclude that v must start with at least one a and that x must end with at least one b. We get a contradiction because our pumping-lemma tells us that $upy \in L$ but x (in the first occurrence of vwx) is followed by v (in second occurrence of vwx) and thus at least one b is followed by at least one a.

We now consider the question of how the value of k affects the generative capacity of $RLIN(k)$.

Theorem 8 *If the language* L *is in* $RREG(k)$ *then* L *is in* $RREG(l)$ *with* $l := \max\{j \mid j$ *is a prime factor of* i, $i = 1, 2, \ldots, k\}$.

Proof: The fact $L \in RREG(k)$ implies the existence of a grammar G which generates L and all productions are of the form $A \to wB^{\leq k}$, $w \in T^*$, $A, B \in I$. Consider a production $f : A \to wB^j$ and let j have the prime decomposition $j = \prod_{1 \leq i \leq r} p_i^{a_i}$, where p_i is a prime number and $a_i > 0$, $1 \leq i \leq r$. Without loss of generality let p_r be the largest of all factors. We are now able to *simulate* $A \to wB^j$ by using the following productions: $A \to wX_{1,1}^{p_1}$, $X_{1,1} \to X_{1,2}^{p_1}, \ldots, X_{1,a_1-1} \to X_{1,a_1}^{p_1}$, $X_{1,a_1} \to X_{2,1}^{p_2}$, $X_{2,1} \to X_{2,2}^{p_2}, \ldots, X_{2,a_2-1} \to X_{2,a_2}^{p_2}, X_{2,a_2} \to X_{3,1}^{p_3}, \ldots, X_{r,a_r-1} \to X_{r,a_r}^{p_r}$, $X_{r,a_r} \to B$. The $X_{i,j}$, $1 \leq j \leq a_i$, $1 \leq i \leq r$, are new nonterminals. By this construction the largest factor p_r determines the degree of the production system which is needed to simulate f. But p_r cannot be larger than the

Figure 5: Two different parts of the derivation tree τ.

maximal prime factor of all the numbers $1, 2, \ldots, k$, which completes the proof.

This theorem implies that if we have any hierarchy in k for $RREG(k)$ then it is a hierarchy with respect to the prime numbers.

Lemma 5 *Let $L \subseteq (T^* \dot{\cup} \{\$\})^n T^*$, $n \in \mathbb{N}$, be a language in $RREG(k)$ for some $k \geq 1$. Then $\mathsf{Pf}(L) := \{u \in T^* \mid u\$v \in L \text{ for some } v\} \in REG$.*

Proof: Let us assume that $\mathcal{L}(G) = L$ for $G = (I, T, P, S) \in RREG(k)$ and $\mathsf{Pf}(L) \notin REG$. Let us further assume that for all $u \in \mathsf{Pf}(L)$ there is no production $f : A \to yB^i \in P$, $i \geq 2$, such that the second B on the right-hand side of f or any B to the right of it derive a part of u within a derivation tree τ for a corresponding $u\$v \in \mathcal{L}(G)$. The last assumption implies that it is possible to construct a right-linear grammar that generates $\mathsf{Pf}(L)$ out of G in the following way: For each $A \to yB^i \in P$ we take the production $A \to yB^\Delta$ with $\Delta = 1$ if there is a $u \in \mathsf{Pf}(L)$ and a derivation of $u\$v \in L$ from S such that $S \overset{lm}{\Longrightarrow}^* xA\alpha \overset{lm}{\Longrightarrow} xyB^i\alpha \overset{lm}{\Longrightarrow}^+ xywB^{i-1}\alpha \overset{lm}{\Longrightarrow}^* u\$v$ for xyw a prefix of u or $w = w_1\$w_2$, $w_1 \neq \varepsilon$, and xyw_1 a prefix of u; we choose $\Delta = 0$ otherwise. Obviously, the resulting grammar is in $RREG(1)$ and if we change in a second step each production $A \to \alpha\$\beta$ into $A \to \alpha$ the result is a right-linear grammar for $\mathsf{Pf}(L)$ which is in contradiction to the assumption that $\mathsf{Pf}(L) \notin REG$.
Thus we can conclude that for at least one $u \in \mathsf{Pf}(L)$ there is a production $f : A \to yB^i$ with $i \geq 2$ and the first two B's of B^i both derive a part of u in a derivation tree τ for a corresponding $u\$v \in L$ with respect to G.
Now let g be the production of τ which generates the leftmost symbol \$. We have to distinguish between three cases:
1. case: Within τ the production g is a successor of production f (this case corresponds to the situation in the left derivation tree shown in Figure 5). Then g must be the successor of the second B of the right-hand side of f or any B right to it as otherwise only one of the B's would contribute to u. But then we could substitute the derivation tree rooted at B that generates \$ by that one which derives a part of u. In that way we would get a word in L with a different number of \$'s in contradiction to our assumption.
2. case: Within τ the production f is a successor of production g (this case corresponds to the right derivation tree shown in Figure 5). But as each successor of g can only be responsible for symbols right to \$, this is impossible.
3. case: Both, f and g, have a common predecessor. In that case this predecessor must have at least two identical nonterminals on its right-hand

side, one of which generates f while the other one produces g. By replacing the related subtrees of τ one by the other we could generate a word with a different number of \$'s again.

Theorem 9 *Let $Pr(i)$ denote the i-th prime number and for an alphabet $T \dot\cup \{\$\}$ let $L \subseteq T^*$. Then for all $i \geq 1$*

$$(\$L)^{Pr(i+1)} \in RREG(Pr(i)) \leftrightarrow L \in REG.$$

Proof: "$\curvearrowleft$:" Obviously for $L \in REG$ we have $\$L \in REG$ and since every finite power of a regular language is regular itself we have $(\$L)^{Pr(i+1)} \in REG \subseteq RREG(Pr(i))$ for arbitrary $i \geq 1$.
"$\curvearrowright$:" To prove this implication assume the existence of a grammar $G = (I, T, P, S) \in RLIN(Pr(i))$ that generates $(\$L)^{Pr(i+1)}$ for $L \notin REG$. We may assume that G has no unit productions and is ε-free. We further assume without loss of generality that no production of G possesses more than one \$ on its right-hand side. Now assume that $P \cap (\{S\} \times (I \cup T)^*) = \{S \to \$\alpha_l D_l^{j_l} \mid 1 \leq l \leq n\}$, i.e. all productions for the axiom generate the first \$. Note, that all the j_l's have to be greater than 0 since every word in $\mathcal{L}(G)$ possesses more than one \$ for every possible i. As we can replace all the derivation trees rooted at D_l one by the other, all possible derivation trees rooted at D_l must produce the same fixed, nonzero number of \$'s. Otherwise we could change the number of \$'s generated and thus produce a word not in $(\$L)^{Pr(i+1)}$. Thus, for any value of j_l, the language $L(D_l)$ generated by the grammar (I, T, P, D_l) fulfils the restrictions of Lemma 5 which implies $\mathsf{Pf}(L(D_l)) \in REG$. But now, as $L = \bigcup_{1 \leq l \leq n} \{\alpha_l\} \cdot \mathsf{Pf}(L(D_l))$, we conclude that $L \in REG$ contrary to our assumptions. Thus we know that there must be a production for S with only nonterminals on its right-hand side i.e. $S \to D^j \in P$, for at least one $j \in [2 : Pr(i)]$ holds. Again, all the D's must produce the same fixed, nonzero number of \$'s. But as we have to produce $Pr(i+1)$ \$'s and none of the integers in $[2 : Pr(i)]$ is a divisor of $Pr(i+1)$ this is impossible.

Corollary 2 *Let $Pr(i)$ denote the i-th prime number. Then we have the following hierarchy:*

$$RREG(Pr(i)) \subset RREG(Pr(i+1)),\ i \geq 1.$$

Proof: This is a trivial consequence of Theorem 9 since $L \notin REG \rightsquigarrow (\$L)^{Pr(i+1)} \notin RREG(Pr(i))$. But for any $G = (I, T, P, S) \in RLIN(Pr(i+1))$ the grammar $(I \dot\cup \{S_1, S_2\}, T, P \dot\cup \{S_1 \to S_2^{Pr(i+1)},\ S_2 \to \$S\}, S_1)$ generates $(\$L(G))^{Pr(i+1)}$. Since we have nonregular languages in $RREG(2)$ (and thus in any $RREG(Pr(i+1))$, $i \geq 1$), e.g. the semi-Dyck-language with one type of brackets, this implies a proper inclusion.

3.3 Recognizing Languages in $RREG(k)$

Recognizing a language in $RREG(k)$ is simple if it is accepted by a deterministic $FSM(k)$, because such an automaton can be directly translated into a corresponding algorithm within an arbitrary programming language. But there is no deterministic automaton for every language in $RREG(k)$ which will be shown in the next theorem. In such cases the recognition process is more complicated and needs further investigations.

Theorem 10 *For every k, $k \geq 2$, there are languages in $RREG(k)$ that cannot be accepted by a deterministic automaton in $FSM(k')$ for arbitrary k'.*

Proof: We will construct a language $L \in RREG(k)$, $k \geq 2$, such that the assumption that L can be accepted by a deterministic automaton leads to a contradiction. For this purpose we use the language used in the proof of Theorem 4 which we denote by L_1. L_2 is defined as $\{w \in \{a,b\}^+ \mid 2|w|_a = |w|_b - 2 \wedge 2|v|_a \geq |v|_b$ for all prefixes v of $w\}$. This language is generated by the grammar $S \to aSS$, $S \to bb$. Renaming the axioms of both grammars, say S_1 and S_2, and constructing a new set of productions $P := P_1 \cup P_2 \cup \{S \to S_1,\ S \to S_2\}$ shows that $L := L_1 \cup L_2$ is $RLIN(2)$. Thus, $L \in RLIN(k)$, $k \geq 2$. Now we assume L to be *deterministic*, i.e. we have a deterministic automaton $\mathsf{A} = (Z, \{a,b\}, \delta, z_0, F)$ in $FSM(k')$, k' arbitrary, which accepts L. Since $abb \in L_1$ there must be a $z_m \in F$ with $(z_0, abb, \varepsilon) \vdash^+ (z_m, \varepsilon, \varepsilon)$. The same must hold for $aabbb$, $aaabbbb$ and so on. But since the set F is finite there must exist two words $w = a^i b^{i+1}$ and $w' = a^j b^{j+1}$, $i \neq j$, which are accepted by the same state, say z_f. Now, regard the computation for the input $a^i b^{i+1} b^{i+1}$ which is in L_2. Since our automaton is assumed to be deterministic we have $(z_0, a^i b^{i+1} b^{i+1}, \varepsilon) \vdash^+ (z_f, b^{i+1}, \varepsilon)$ and since the input is in L_2 we must have $(z_f, b^{i+1}, \varepsilon) \vdash^+ (z'_f, \varepsilon, \varepsilon)$, $z'_f \in F$. Thus, we have $(z_0, a^j b^{j+1} b^{i+1}, \varepsilon) \vdash^+ (z_f, b^{i+1}, \varepsilon) \vdash^+ (z'_f, \varepsilon, \varepsilon)$. This implies a contradiction because the input $a^j b^{j+1} b^{i+1}$ is not in $L_1 \cup L_2$ for $i \neq j$.

The last theorem implies that there is no algorithm to make a nondeterministic finite state machine of degree k deterministic. As our contradiction follows independently of k the proof also shows that it is not possible to make a nondeterministic finite state machine deterministic even if the degree of the automaton is increased.

4 The Strong Model

We now consider the case that the call of some state is not implemented as a normal transition. We assume that it is implemented by changing the actual state of the automaton after pushing it onto the stack by means of a modified finite control. Using the same concepts as before and allowing the actual state to be pushed at most k times leads to the following definition:

Definition 7 *An* extended finite state machine of degree k *is a five-tuple* $\mathsf{A} = (Z, T, \delta, z_0, F)$ *with*

- Z *is a finite set of states,*
- T *is the input alphabet,*
- $\delta \subseteq (Z \times T_\epsilon) \times (Z \times Z_\varepsilon \times [0:k])$,
- $z_0 \in Z$ *is the initial state,*
- $F \subseteq Z$ *is the set of final states.*

For $z \in Z$ *and* $a \in T_\varepsilon$ *we define* $\delta(z,a) := \delta \cap (\{z\} \times \{a\}) \times (Z \times Z_\varepsilon \times [0:k])$. *The set of all extended finite state machines of degree* k *is denoted by* $ExFSM(k)$.

Note, that $((z,a),(z_1,z_2,j)) \in \delta$ corresponds to the case that we can read symbol a in order to go from z to z_2; z_2 is pushed j times for a later return, then the subroutine z_1 is called.

Definition 8 *Let* $\mathsf{A} = (Z, T, \delta, z_0, F) \in ExFSM(k)$. *A triple* $(z, w, s) \in Z \times T^* \times Z^*$ *is called* configuration. *The triple* (z_0, w, ε) *is called* start-configuration with input w. *A triple* $(z, \varepsilon, \varepsilon)$ *is called* end-configuration *if* $z \in F$ *holds.*

Definition 9 *Let* $\mathsf{A} = (Z, T, \delta, z_0, F) \in ExFSM(k)$. *A* move *of* A *is represented by the relation* $\vdash_\delta \subseteq (Z \times T^* \times Z^*)^2$ *with* $(z_1, w_1, s_1) \vdash_\delta (z_2, w_2, s_2)$ *if and only if*

- $((z_1, a), (z_2, z, j)) \in \delta$, $a \cdot w_2 = w_1$ *and* $s_1 \cdot z^j = s_2$,

or

- $w_1 = w_2$, $s_1 = s_2 \cdot z_2$ *and* $z_1 \in F$.

A move with respect to the first alternative is called a-transition, *a move corresponding to the second one will be called* pop-operation. *If the context is obvious, the index of* $\vdash$ *is omitted. The language* $L(\mathsf{A})$ *accepted by* A *is defined as* $L(\mathsf{A}) := \{w \in T^* \mid (z_0, w, \varepsilon) \vdash^* (z, \varepsilon, \varepsilon) \text{ with } z \in F\}$.

Definition 10 *Let* $\mathsf{A} = (Z, T, \delta, z_0, F) \in ExFSM(k)$. A *is called* deterministic *if the following conditions hold:*

(D1) $(\forall (q,a) \in Z \times T_\varepsilon) : (((q,a),(q_1,q_2,i)) \in \delta \wedge ((q,a),(q_1',q_2',j)) \in \delta \rightsquigarrow q_1 = q_1' \wedge q_2 = q_2' \wedge i = j)$;

(D2) $((q,\varepsilon),(q_1,q_2,i)) \in \delta \rightsquigarrow (\neg\exists (a, q_1', q_2', j) \in T \times Z^2 \times [0:k]) : (((q,a),(q_1',q_2',j)) \in \delta)$;

(D3) $(\forall w \in T^*,\ a,b \in T_\varepsilon,\ z,z' \in Z,\ s,s' \in Z^*) : ((z_0,w,\varepsilon) \vdash^* (z',\varepsilon,s'zs) \rightsquigarrow z' \notin F \vee s \notin F^* \vee \delta(z',a) = \emptyset \vee \delta(z,b) = \emptyset \vee (a \neq b \wedge a \neq \varepsilon \wedge b \neq \varepsilon))$.

Note, that (D3) is decidable.
Let us have a look at an example. The automaton presented in Figure 6 is in $ExFSM(1)$. For z_1 being the initial state it accepts the language $\{\varepsilon, bb\}$. If $x = y$ holds, then the automaton is not deterministic because of a violation of condition (D1); if we have $y = \varepsilon$ then condition (D2) is not fulfilled. A contradiction to condition (D3) would result from $y = \varepsilon$, too.

Figure 6: Example of an automaton in $ExFSM(1)$. A square is used to represent a final state; a label x, z at an edge denotes an x-transition that pushes z.

Definition 11 *Let $G = (I, T, P, S)$ be a grammar. G is said to be* extended k-right-linear *if $P \subseteq I \times T^* I \bigcup_{A \in I} A^{\leq k}$. The set of all extended k-right-linear grammars is denoted by $ExRLIN(k)$.*

It is obvious, that every grammar in $ExRLIN(k)$ is context-free. Furthermore, for $k = 1$, $ExRLIN(k)$ contains the set of all context-free grammars being in 2-standard form. This implies that $ExRLIN(1)$ generates all context-free languages and thus $ExRREG(k) = CFL$ for all $k > 0$; here, $ExRREG(k)$ denotes the set of all languages that can be generated by a grammar in $ExRLIN(k)$.

Theorem 11 *For every grammar $G = (I, T, P, S)$ in $ExRLIN(k)$, $k \geq 0$, there is an automaton $\mathsf{A} = (Z, T, \delta, z_0, F)$ in $ExFSM(k)$ with $L(\mathsf{A}) = \mathcal{L}(G)$ and vice versa.*

Proof: "$\curvearrowright$:" As in the proof of Theorem 2 we can assume the productions of $G = (I, T, P, S)$ to have at most one terminal-symbol on their right-hand sides. Then, an equivalent automaton $\mathsf{A} = (Z, T, \delta, z_0, F)$ is defined as follows:

- $Z := I \dot{\cup} \{z_e\}$,
- $z_0 = S$,
- $F := \{z_e\} \cup \{z \in Z \mid z \to \varepsilon \in P\}$,
- $\delta := \{((z, a), (\bar{z}, \tilde{z}, i)) \mid z \to a\bar{z}\tilde{z}^i \in P,\ a \in T_\varepsilon,\ \bar{z}\tilde{z}^i \neq \varepsilon\} \cup$
  $\{((z, a), (z_e, \varepsilon, 0)) \mid z \to a \in P, a \in T\}$.

Now, an induction similar to the one presented in Theorem 2 shows that A accepts the language generated by G.
"$\curvearrowleft$:" Define $G = (I, T, P, S)$ as follows:

- $I := Z$,
- $S := z_0$,
- $P := \{B \to aCD^j \mid ((B, a), (C, D, j)) \in \delta\} \cup \{B \to \varepsilon \mid B \in F\}$.

It is easy to prove the equivalence of G to A by induction.

Thus, we have a new automata-model for the context-free languages. It differs formally from the classical pushdown automata by the fact that states instead of symbols of a freely choosen alphabet are pushed onto a stack. A major difference between both is the determinism. The language $L = \{a^n b_i^n \mid i \in \{1,2\}\}$ is a deterministic language in the classical context, it cannot be accepted by a deterministic automaton of the new model. Before we can investigate the determinism in detail we need to introduce the notion of a reduced deterministic automaton.

Definition 12 *Let* $\mathsf{A} = (Z, T, \delta, z_0, F) \in ExFSM(k)$. *A state* $z \in F$ *is called* nullable *if there exists* $((z, \varepsilon), (z_1, z_2, j)) \in \delta$ *such that* $(z_1, \varepsilon, z_2^j) \vdash^* (z_f, \varepsilon, \varepsilon)$ *with* $z_f \in F$. *We say that* A *is* nonnullable *if* A *has no nullable state. For* $a \in T$ *and* $z \in Z$ *we define*

$$\Delta^z(a) := \{\bar{z} \in Z \mid (z, \varepsilon, \varepsilon) \vdash^* (\bar{z}, \varepsilon, \alpha) \wedge \delta(\bar{z}, a) \neq \emptyset\}$$

and

$$\Delta^z := \bigcup_{a \in T} \Delta^z(a).$$

Note, that it is decidable whether or not a state is nullable.

Lemma 6 *Let* $\mathsf{A} = (Z, T, \delta, z_0, F) \in ExFSM(k)$. *Then there exists an* $\mathsf{A}' = (Z, T, \delta', z_0, F) \in ExFSM(k)$ *with* $L(\mathsf{A}) = L(\mathsf{A}')$ *such that for all* $((z, a), (z_1, z_2, j)) \in \delta'$, $j \geq 1$, *holds:* $\bigcup_{b \in T_\varepsilon} \delta'(z_2, b) \neq \emptyset$. *If* A *is deterministic then* A' *is deterministic, too.*

Proof: Assume that there is a transition $t = ((z, a), (z_1, z_2, j)) \in \delta$ such that $\bigcup_{b \in T_\varepsilon} \delta(z_2, b) = \emptyset$. We have to distinguish between two cases:
1. case: $z_2 \in F$. In this case we construct δ' from δ by deleting t while adding the new transition $((z, a), (z_1, \varepsilon, 0))$. It is obvious that this modification does not change the language accepted by the automaton. It is also obvious that it does not introduce new nondeterminism.
2. case: $z_2 \in Z \backslash F$. Since there is no move starting from (z_2, v, γ), $v \in T^*$, $\gamma \in Z^*$, the input is rejected by A, whenever z_2 is reached or pushed (as each state on the stack must eventually become the actual state if the automaton should accept the input). Thus we set $\delta' := \delta \backslash \{(Z \times T_\varepsilon) \times (Z \times \{z_2\} \times [1:k]) \cup (Z \times T_\varepsilon) \times (\{z_2\} \times Z_\varepsilon \times [0:k])\}$ without changing the accepted language. It is obvious that this construction does not introduce nondeterminism since $\delta' \subseteq \delta$ and therefore each nondeterministic choice for A' would exist for A also.

Lemma 7 *Let* $\mathsf{A} = (Z, T, \delta, z_0, F) \in ExFSM(k)$ *be deterministic and let* $z \in Z$ *be a nullable state. Then*

$$(\forall a \in T) : (|\Delta^z(a)| \leq 1)$$

and for $\bar{z} \in \Delta^z$ *holds*

$$(z, \varepsilon, \varepsilon) \vdash^+ (\bar{z}, \varepsilon, \alpha) \wedge (z, \varepsilon, \varepsilon) \vdash^+ (\bar{z}, \varepsilon, \beta) \rightsquigarrow \alpha = \beta.$$

Proof: Assume the existence of $a \in T$ and $\bar{z}_1, \bar{z}_2 \in Z$ with $\delta(\bar{z}_i, a) \neq \emptyset$, $(z, \varepsilon, \varepsilon) \vdash^* (\bar{z}_i, \varepsilon, \alpha_i)$, $i \in \{1, 2\}$, $(\bar{z}_1 \neq \bar{z}_2 \vee \bar{z}_1 = \bar{z}_2 \wedge \alpha_1 \neq \alpha_2)$. Then there exists a maximal $v \in \mathbb{N}_0$ with

$$(z, \varepsilon, \varepsilon) \vdash^v (\hat{z}, \varepsilon, \gamma) \begin{cases} \vdash^i (\bar{z}_1, \varepsilon, \alpha_1) \\ \vdash^j (\bar{z}_2, \varepsilon, \alpha_2) \end{cases}$$

1. case: $i = j = 0$. Here $\bar{z}_1 = \bar{z}_2$ and $\alpha_1 = \alpha_2$, this is in contradiction to our assumption.
2. case: $i = 0, j \geq 1$ ($i \geq 1, j = 0$ analogously). Regard $(\bar{z}_1, \varepsilon, \alpha_1) \vdash^+ (\bar{z}_2, \varepsilon, \alpha_2)$. As there is no input-symbol to be read, the whole computation $\vdash^+$ must consist of ε-transitions and pop-operations only. But assume that there is a first ε-transition within $\vdash^+$. Then we have a contradiction to (D3) since $\delta(\bar{z}_1, a) \neq \emptyset$. Thus $\vdash^+$ must entirely consist of pop-operations which again implies a contradiction to (D3) as $\delta(\bar{z}_1, a) \neq \emptyset \wedge \delta(\bar{z}_2, a) \neq \emptyset$. Thus $(\bar{z}_1, \varepsilon, \alpha_1) \vdash^+ (\bar{z}_2, \varepsilon, \alpha_2)$ is impossible.
3. case: $i \geq 1, j \geq 1$: This case corresponds to the situation that $(\hat{z}, \varepsilon, \gamma) \vdash (\hat{z}_1, \varepsilon, \gamma_1) \vdash^* (\bar{z}_1, \varepsilon, \alpha_1)$ and $(\hat{z}, \varepsilon, \gamma) \vdash (\hat{z}_2, \varepsilon, \gamma_2) \vdash^* (\bar{z}_2, \varepsilon, \alpha_2)$ with $(\hat{z}_1, \varepsilon, \gamma_1) \neq (\hat{z}_2, \varepsilon, \gamma_2)$ by the maximal choice of v. The different successors of $(\hat{z}, \varepsilon, \gamma)$ cannot result from two different pop-operations (as there is a unique right-most symbol of γ) and they cannot result from two different ε-transitions as A is deterministic. But the case of one ε-transition together with a pop-operation implies a contradiction to (D3) since the pop-operation (or any pop-operation that is performed afterwards) must eventually reach a state (at least $\bar{z}_i$, $i \in \{1, 2\}$) with a transition.

Lemma 8 *Let* $\mathsf{A} = (Z, T, \delta, z_0, F) \in ExFSM(k)$ *be deterministic. Then there is a deterministic, nonnullable automaton* $\mathsf{A}' = (Z', T, \delta', z_0, F) \in ExFSM(k)$ *with* $L(\mathsf{A}) = L(\mathsf{A}')$.

Proof: Let N denote the set of all nullable states of A. By Lemma 6 we can assume that A pushes no state without a transition. Thus, since $N \subseteq F$, we can conclude that $(z_0, w, \varepsilon) \vdash^* (z, \varepsilon, \alpha)$, for $z \in N$, implies $\alpha = \varepsilon$ as any state on the top of the stack would contradict the determinism of A with respect to (D3). Therefore it suffices to prove that we can make z nonnullable without changing the set $L(z) := \{w \in T^* \mid (z, w, \varepsilon) \vdash^* (z_f, \varepsilon, \varepsilon) \wedge z_f \in F\}$. We do this by deleting the ε-transition starting at z while introducing the following new transitions and states: For all $a \in T$ and all states $\bar{z} \in \Delta^z(a)$ we determine the corresponding tuples $(z_1, \beta) \in Z \times Z^*$ such that $(z, \varepsilon, \varepsilon) \vdash^+ (\bar{z}, \varepsilon, \beta') \wedge ((\bar{z}, a), (z_1, z_2, i)) \in \delta \wedge \beta = \beta' z_2^i$. We define $m(\beta)$ to be the smallest integer such that there are $\underline{z}_i \in Z$ and $k_i \in \mathbb{N}$, $1 \leq i \leq m(\beta)$, with $\beta = \underline{z}_1^{k_1} \cdots \underline{z}_{m(\beta)}^{k_{m(\beta)}}$. If $m(\beta) = 0$ it suffices to add the transition $((z, a), (z_1, \varepsilon, 0))$. If $m(\beta) = 1$ we add the transition $((z, a), (z_1, \underline{z}_1, k_1))$ and we are done. In all other cases we introduce the new states $\tilde{z}_i$, $1 \leq i < m(\beta)$, and the transitions $((\tilde{z}_{j-1}, \varepsilon), (\tilde{z}_j, \underline{z}_j, k_j))$, $1 < j < m(\beta)$. We further add the transitions $((z, a), (\tilde{z}_1, \underline{z}_1, k_1))$ and

$((\tilde{z}_{m(\beta)-1}, \varepsilon), (z_1, \underline{z}_{m(\beta)}, k_{m(\beta)}))$. Note, that if k_i is larger than k, $1 \leq i \leq m(\beta)$, then the corresponding transition must be split into an equivalent chain of transitions by introducing additional new states. Since none of the new states is accepting, none of them can be nullable. By Lemma 7 we know that for each $a \in T$ there is at most one state $\bar{z} \in \Delta^z(a)$ and if $\bar{z}$ exists, it uniquely determines the corresponding tuple (z_1, β). Thus our construction does not introduce nondeterminism with respect to (D1) or (D2). New nondeterminism with respect to (D3) cannot be introduced as none of the new states is pushed or is accepting. Thus it remains to prove that $L(z)$ is left unchanged. By definition, $z \in N$ is accepting and thus $\varepsilon \in L(z)$. But the set of accepting states is not changed by our construction and thus ε is accepted by z for A' as well. Now consider any word $w \in L(z) \cap T^+$. By Lemma 7, starting at (z, w, ε), the first symbol a of w determines uniquely the configuration that A possesses directly after reading a. Let this configuration be (z_1, w', β). But then, by construction, there is a chain of moves for A' such that $(z, w, \varepsilon) \vdash_{\delta'}^+ (z_1, w', \beta)$. Therefore A' will accept w as well.

Let us return to the examplary automaton of Figure 6. We assume that $x = a$ and $y = c$ such that the automaton is deterministic. We regard all final states in order to determine the set of nullable states. State z_1 is nullable because of the transition $((z_1, \varepsilon), (z_2, z_4, 1))$ and the moves $(z_2, \varepsilon, z_4) \vdash (z_3, \varepsilon, z_4) \vdash (z_4, \varepsilon, \varepsilon)$ with z_4 accepting. States z_3 and z_6 are not nullable since they do not possess an ε-transition. State z_5 is not nullable because its ε-transition pushes state z_9 but z_9 cannot accept the empty word. To apply the construction of the previous proof in order to make z_1 nonnullable we need to determine the sets $\Delta^{z_1}(x)$, $x \in \{a, b, c\}$. We find $\Delta^{z_1}(a) = \{z_6\}$, $\Delta^{z_1}(b) = \{z_9\}$ and $\Delta^{z_1}(c) = \{z_6\}$. The related tuples (z, β) are given by (z_7, z_9^3), (z_{10}, z_9) and (z_8, z_9^3), respectively. Thus, $m(\beta) = 1$ in all three cases; the first and the last case have $k_1 = 3$, the second case has $k_1 = 1$. If we now apply the construction given in the previous proof we find the automaton presented in Figure 7. Note, that the resulting automaton is not in $ExFSM(1)$ because (for the symbols a and c) $k_1 = 3$ holds. Therefore we have to split the transition $((z_1, a), (z_7, z_9, 3))$ into the equivalent chain of transitions $((z_1, a), (z_{n1}, z_9, 1))$, $((z_{n1}, \varepsilon), (z_{n2}, z_9, 1))$, $((z_{n2}, \varepsilon), (z_7, z_9, 1))$ by introducing the additional new states z_{n1} and z_{n2}. The transition $((z_1, c), (z_8, z_9, 3))$ must be treated in the same way.

Figure 7: A nonnullable automaton equivalent to the one shown in Figure 6.

Now, we are prepared to prove the following theorem which gives a detailed characterization of the different notions of determinism.

Theorem 12 *The set of languages accepted by a deterministic automaton* $\mathsf{A} \in ExFSM(k)$, $k \geq 1$, *is equal to those languages that can be generated by a context-free grammar* G *which is* $LL(1)$.

Proof: "$\curvearrowright$:" Without loss of generality we assume that $\mathsf{A} = (Z, T, \delta, z_0, F)$ is nonnullable. Now regard the grammar $G = (I, T, P, S)$ constructed from A as presented in the proof of Theorem 11 and assume that G is not $LL(1)$. Then there exist two derivations d_1 and d_2 with $d_1 = S \overset{lm}{\Longrightarrow}^* wB\alpha \overset{lm}{\Longrightarrow} w\beta\alpha \overset{lm}{\Longrightarrow}^* wx$ and $d_2 = S \overset{lm}{\Longrightarrow}^* wB\alpha \overset{lm}{\Longrightarrow} w\gamma\alpha \overset{lm}{\Longrightarrow}^* wy$ with $\mathsf{FIRST}_1(x) = \mathsf{FIRST}_1(y)$ but $\beta \neq \gamma$, i.e. there are two different productions $f_1 : B \to \beta$ and $f_2 : B \to \gamma$. We have to distinguish between the following cases:
1. case: Both, β and γ, start with a terminal symbol. This has to be the same symbol a for both since $\mathsf{FIRST}_1(x) = \mathsf{FIRST}_1(y)$. Now, by construction, f_1 and f_2 result from two different a-transitions starting at the same state B which contradicts the determinism of A with respect to (D1).
2. case: At most one of the right-hand sides of f_1 and f_2 starts with a terminal symbol, none of the right-hand sides is ε. By construction, f_1 and f_2 result from two different transitions for the same state B but since at least one of them must be an ε-transition this contradicts the determinism of A with respect to (D1) or (D2).
3. case: At least one of the right-hand sides of f_1 and f_2 is ε. Without loss of generality we assume $f_1 : B \to \varepsilon$ and since $\beta \neq \gamma$ either $\gamma \in T^+Z^*$ or $\gamma \in Z^+$ must hold. $\gamma \in T^+Z^*$ implies $\mathsf{FIRST}_1(y) \neq \{\varepsilon\}$ and therefore $\mathsf{FIRST}_1(x) \neq \{\varepsilon\}$ also. $\gamma \in Z^+$ implies the existence of a transition $((B, \varepsilon), (C, D^j)) \in \delta$ with $CD^j = \gamma$ and as A is nonnullable we know that $\varepsilon \notin \mathcal{W} := \{w \in T^* \mid (C, w, D^j) \vdash^* (z_f, \varepsilon, \varepsilon) \wedge z_f \in F\}$. But since $w\gamma\alpha \overset{lm}{\Longrightarrow}^* wy \in T^*$ we know that $\mathcal{W} \neq \emptyset$ and we conclude that $\mathsf{FIRST}_1(y) \neq \{\varepsilon\}$ in this case as well. Thus for any choice of γ there must exist an $X \in I$ and $\alpha_1, \alpha_2 \in I^*$ with $\alpha = \alpha_1 X \alpha_2$ such that X does not produce ε within the derivation d_1. We choose $|\alpha_1|$ minimal which implies that $Y \to \varepsilon \in P$ for all Y in α_1. But then for the corresponding computation of A we have the actual state B which is accepting and all states that are above X on the stack are accepting, too. This fact implies a contradiction to (D3) because f_2 implies that $\delta(B, a) \neq \emptyset$ and $\delta(X, b) \neq \emptyset$ for $a = \varepsilon \vee b = \varepsilon \vee a = b$ as $\mathsf{FIRST}_1(x) = \mathsf{FIRST}_1(y)$.

"$\curvearrowleft$:" Let $G = (I, T, P, S)$ be $LL(1)$. We use the construction given in proof [4, Theorem 3] which generates a $LL(1)$ grammar $G' = (I_1, T, P_1, S_1)$ with $\mathcal{L}(G') = \mathcal{L}(G) \backslash \{\varepsilon\}$, where G' satisfies the properties:

- For all $f : A \to \alpha \in P_1$ holds: either $\alpha = \varepsilon$ or $\alpha = X\alpha'$, $\neg X \Longrightarrow^* \varepsilon$;
- $I_1 = I \,\dot\cup\, \{\bar{A} \mid A \in I \wedge A \Longrightarrow^+ \varepsilon$ with respect to $P\}$, $(\bar{A} \to \alpha \in P_1) \rightleftharpoons (A \to \alpha \in P_1 \wedge \alpha \neq \varepsilon)$;
- $S_1 = S$ if $\varepsilon \notin \mathcal{L}(G)$ otherwise $S_1 = \bar{S}$.

Therefore, $\mathcal{L}(G_1) = \mathcal{L}(G)$ for $G_1 := (I_1, T, P_1, S)$ and we claim that G_1 is $LL(1)$, too. To prove this claim assume the contrary and ask for two leftmost derivations starting at S that contradict the $LL(1)$ property for G_1 but that do not exist when starting at S_1. Now, as $S \to \varepsilon$ is the only production that might exist for S but not for S_1, $(S \Longrightarrow^+ \alpha) \wedge \neg(S_1 \Longrightarrow^+ \alpha)$ is only possible if $S \to \varepsilon$ is the first production applied. But then, in order to contradict the $LL(1)$ property, there must exist a production $S \to \alpha$, $\alpha \neq \varepsilon$, $\alpha \overset{lm}{\Longrightarrow}^+ \varepsilon$, but for P_1 this is impossible.
For $\mathcal{E} := \{f : A \to \varepsilon \mid f \in P_1\}$ we apply the construction given in proof [4, Theorem 4] to the grammar $(I_1, T, P_1 \backslash \mathcal{E}, S)$. The result is a grammar (I_1, T, P_2, S) such that $G_2 := (I_1, T, P_2 \cup \mathcal{E}, S)$ is $LL(1)$ and $\mathcal{L}(G_2) = \mathcal{L}(G)$ holds. Additionally each production $f : A \to \alpha$ of P_2 fulfils $\alpha \in TI_1^*$. It is possible that P_2 possesses productions $f : A \to aBC^j\alpha$ that do not fulfil the restrictions of a grammar in $RLIN(k)$ for arbitrary k. For all such f we factorize $C^j\alpha$ into $A_1A_2 \cdots A_m$, $A_i \in I_1$, $1 \leq i \leq m$, and introduce the new nonterminal H_i, $1 \leq i < m$, together with the rules $H_i \to A_iH_{i+1}$, $1 \leq i < m-1$, $H_{m-1} \to A_{m-1}A_m$. Finally, we replace f by $A \to aBH_1$ such that $A \Longrightarrow^m aBC^j\alpha$. It is obvious that the resulting grammar generates the same language and remains $LL(1)$. Only for technical reasons we introduce the new nonterminal X_e together with the production $X_e \to \varepsilon$ and transform each terminal rule $A \to a$, $a \in T$, into $A \to aX_e$. Let the resulting grammar be named $G_3 = (I_3, T, P_3, S)$ and consider the automaton A which is constructed from G_3 as given in the proof of Theorem 11. Let us assume that A is not deterministic.
A contradiction to (D1) implies the existence of two productions $f_1 : A \to a\alpha$ and $f_2 : A \to a\beta$, $a \in T_\varepsilon$, $\alpha \neq \beta$, and as we can assume (without loss of generality) that A has no useless state; this contradicts the $LL(1)$ property of G_3 for $a \in T$. Now, since P_2 has Greibach normal form, $a = \varepsilon$ is only possible if A is one of the new nonterminals H_i introduced while constructing G_3. But then $\alpha = \beta$ because there is only one production for every H_i.
A contradiction to (D2) would imply the existence of a production $A \to \alpha$ with $\alpha \in I_3^+$ together with a second production g for the nonterminal A. Again, because of the Greibach normal form of P_2, $A \to \alpha$ is only possible if A is one of the new nonterminals H_i introduced while constructing G_3. But since there is only one production for every H_i, the existence of g is impossible.
A violation of (D3) could only occur if there exist $w \in T^*$, $a, b \in T_\varepsilon$ and $A \in I_3$, $A \to \varepsilon \in P_3$, such that $S \overset{lm}{\Longrightarrow}^* wA\alpha$ with $\alpha = \alpha_1 B \alpha_2$, $C \to \varepsilon \in P_3$ for all C in α_1, together with two productions $f_1 : A \to a\beta$, $f_2 : B \to b\gamma$ in P_3 fulfilling $\beta \in I_3^+$, $\gamma \in I_3^+$ and $(a = b \vee a = \varepsilon \vee b = \varepsilon)$. The cases $a = \varepsilon$ and $b = \varepsilon$ are impossible because they would imply either $A = H_i$ but $H_i \to \varepsilon \notin P_2$ or $B = H_j$ but H_j is never pushed; here H_i and H_j are new states introduced while constructing G_3 from G_2. The case $a = b$ implies

the following derivations for G_3:

$$S \overset{lm}{\Longrightarrow} wA\alpha \begin{cases} \overset{lm}{\Longrightarrow} w\alpha_1 B\alpha_2 \overset{lm}{\Longrightarrow}^* wB\alpha_2 \overset{lm}{\Longrightarrow} wb\gamma\alpha_2 \overset{lm}{\Longrightarrow}^* wby' = wy \\ \overset{lm}{\Longrightarrow} wa\beta\alpha \overset{lm}{\Longrightarrow}^* wax' = wx \end{cases}$$

with $\mathsf{FIRST}_1(x) = \mathsf{FIRST}_1(y)$ but $a\beta\alpha \neq \alpha_1 B\alpha_2$ which is impossible as G_3 is $LL(1)$.

Note, that the assumption for A being nonnullable is essential. A nullable automaton would imply productions like $A \to aB$, $B \to \varepsilon$, $B \to C$, $C \to \varepsilon$ where we have two different B-productions with identical FIRST_1-sets.

Corollary 3 *The set of languages that can be accepted by a deterministic automaton in $ExFSM(1)$ is equal to those languages that can be generated by a context-free grammar which is $LL(1)$.*

Proof: It is sufficient to prove that every deterministic automaton out of the class $ExFSM(k)$ can be transformed into an equivalent deterministic automaton in $ExFSM(1)$. This can be done in the following way. Assume that $((z,a),(z_1,z_2,j))$ is a transition with $j > 1$. Then we delete this transition and introduce the new states $z(i)$, $1 \leq i < j$, together with the new transitions $((z,a),(z(1),z_2,1))$, $((z(i),\varepsilon),(z(i+1),z_2,1))$, $1 \leq i < j-1$ and $((z(j-1),\varepsilon),(z_1,z_2,1))$. Obviously, the behavior of the automaton is left unchanged. As none of the new states is accepting or pushed by any transition, and each new state has exactly one outgoing transition the automaton stays deterministic.

5 Conclusion

Within this paper we have considered the generative capacity of subroutines introduced for finite state machines. Two possibilities of implementing the procedure-call have been regarded. First, a call was implemented as a normal transition. This led to a new class of languages called $RREG(k)$ which includes some context-free languages like the Dyck-language with one type of brackets, but is a proper subset of the set CFL. However, it seems to be possible to generalize this result; there is a conjecture that each bounded language in $RREG(k)$ is regular (for a definition of *bounded* see [2]). An automata-model corresponding to $RREG(k)$ was introduced and fundamental results like a pumping-lemma were presented. Second, we considered a call to be implemented as a jump which made a finite state machine as powerful as a pushdown automaton. As a consequence an alternative automata-model for the class CFL was found. One interesting detail of this new automaton is the determinism. We have proved that the set of languages which can be accepted deterministically is the same as the set of languages which can be generated by a $LL(1)$ grammar. It is well known that the set of languages which can be accepted by a deterministic

pushdown automaton is equal to those for which we have a grammar that is $LR(1)$. The class of languages that are $LL(1)$ is a proper subclass of those that are $LR(1)$. Thus the new automata-model in its deterministic variation is less powerful than the classical deterministic pushdown automata. From a practical point of view, the new automata-model is well motivated. For example, there is a language called DPF (Disc Plot Format) which is used by the company Barco Graphics in Gent, Belgium, to represent graphical information. Most parts of that language are generated by right-linear rules only. Thus a syntax-check might be implemented as a finite state machine or its programming-language equivalent. However, one rule within the production system of the DPF is not right-linear. This rule is of the pattern $A \to a(S)$ (A, S nonterminals). As shown in this paper a simple extension of the finite automaton by a stack on which states might be stored suffices to get an equivalent automaton. This becomes interesting when thinking of the reusability of software. If the syntax of any language is changed such that a formerly regular language becomes non-regular, a program (the finite automaton) needs not to be rewritten from scratch as most of its parts can be kept the way they are. This was the starting point of the present paper where the idea of finite state machines with subroutines came up.

Acknowlegements: I would like to thank Uwe Trier who brought the DPF to my attention and thus gave me the motivation for this work. I also wish to thank an anonymous referee whose comments helped to improve the quality of this paper.

References

[1] Alfred V: Aho and Jeffrey D. Ullman, *The Theory of Parsing, Translation and Compiling, Volume I: Parsing*, Prentice-Hall Series in Automatic Computation, 1972

[2] S. Ginsburg, *The Mathematical Theory of Context-Free Languages* , McGraw-Hill, 1966

[3] Michael A. Harrison *Introduction to Formal Language Theory*, Addison-Wesley Publishing Company, 1978

[4] D. J. Rosenkrantz and R. E. Stearns, Properties of deterministic top-down grammars, *Information and Control* **17**, 226-256, 1970

[5] A. Salomaa and M. Soittola, *Automata-Theoretic Aspects of Formal Power Series*, Springer, 1978