Arbeitsgruppe Informatik
Universität Giessen
Arndtstr. 2, D-35392 Giessen, Germany

5. Theorietag „Automaten und Formale Sprachen“

Schloß Rauischholzhausen, 28./29.9.1995

Martin Kutrib, Thomas Worsch (Hrsg.)

Bericht 9503 Dezember 1995

JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

Vorwort

Nach Magdeburg (September 1991), Kiel (Oktober 1992), Schloß Dagstuhl (Oktober 1993) und Herrsching bei München (September 1994) fand der nunmehr 5. Theorietag „Automaten und Formale Sprachen“ der GI-Fachgruppe 0.1.5 am 28. und 29. September in Schloß Rauischholzhausen bei Gießen statt.

34 Teilnehmer aus Deutschland und Österreich folgten der Einladung und sorgten mit ihren Beiträgen und angeregten wissenschaftlichen Diskussionen für eine lebhafte Atmosphäre. Das wissenschaftliche Programm bestand aus 20 Vorträgen, deren Ausarbeitungen im vorliegenden Bericht enthalten sind.

Wir danken allen Teilnehmern für ihren Beitrag zum Gelingen der Veranstaltung. Wir danken auch der Universität Gießen und insbesondere der AG Informatik und ihrem Chef Prof. Dr. Henner Kröger für ihre Unterstützung.

Dem nächsten Theorietag in Cunnersdorf bei Königsstein in der Sächsischen Schweiz wünschen wir viel Erfolg.

Gießen, im Dezember 1995

Martin Kutrib
Thomas Worsch

Inhaltsverzeichnis

Programm

Donnerstag, 28. September 1995

9:20–9:30 **Begrüßung**

9:30–10:40 Sitzungsleiter: Martin Kutrib

Hauptvortrag: **Rüdiger Reischuk** (Universität Lübeck), *Turing Maschinen mit sublogarithmischen Platzschranken*

10:40–11:10 **Kaffeepause**

11:10–12:30 Sitzungsleiter: Thomas Worsch

Martin Hühne (Universität Dortmund), *Effiziente Speicherstrukturen für Turingmaschinen mit* einem *Kopf*

Klaus P. Jantke (HTWK Leipzig), *Introducing a Two-Level Grammar Concept for Design*

Armin Kühnemann (TU Dresden), *A Pumping Lemma for Output Languages of Attributed Tree Transducers*

Gerhard Buntrock (Universität Würzburg), *Einige Bemerkungen zum Pumpen*

12:30–14:20 **Mittagspause**

14:20–15:25 Sitzungsleiter: Wolfgang Thomas

Ludwig Staiger (Universität Halle-Wittenberg), *Uniformly optimal prediction of infinite words*

Andreas Schrader (Universität-GH Siegen), *Bildkompression mit endlichen Automaten*

Thomas Buchholz (Universität Gießen), *Konstruierbarkeit von Funktionen in zellularen Automaten*

15:25–15:50 **Kaffeepause**

15:50–16:50 Sitzungsleiter: Rüdiger Reischuk

Klaus-Jörn Lange (Universität Tübingen), *Automaten mit der Datenstruktur Menge*

Manfred Droste (Technische Universität Dresden), *Erkennbare, aperiodische und logisch definierbare Sprachen in Nebenläufigkeitsmonoiden*

Rudolf Freund (TU Wien), *Contextual Array Grammars*

16:50–17:30 **Mitgliederversammlung der Fachgruppe**

Freitag, 29. September 1995

9:00–10:30 Sitzungsleiter: Matthias Jantzen

Thomas Worsch (Universität Karlsruhe), *Automaten mit ein bißchen Parallelität*

Henning Fernau (Universität Tübingen), *Programmierte Grammatiken mit unbedingtem Übergang und verwandte Mechanismen*

Henning Bordihn (Universität Magdeburg), *Akzeptierende programmierte Grammatiken ohne Nichtterminale*

Markus Holzer (Universität Tübingen), *On the Complexities of Deterministic Indexed Grammars*

10:30–10:50 **Kaffeepause**

10:50–12:45 Sitzungsleiter: Manfred Droste

Martin Kutrib (Universität Gießen), *On language families of interacting automata*

Norbert Gutleben (TU Cottbus), *Minimierung von Muller-Automaten*

Helmut Seidl (Universität Trier), *A Modal μ–Calculus for Durational Transition Systems*

David Damanik (Universität Frankfurt), *Endliche Automaten mit eingeschränkter 2-Wege-Bewegung*

Daniel Kirsten (HTWK Leipzig), *Properties of Formal languages of Therapy Plans created by Graph Grammars*

12:45– **Mittagspause und Abreise**

Accepting programmed grammars without nonterminals

Henning Bordihn

Fakultät für Informatik
Otto–von–Guericke–Universität Magdeburg
Postfach 4120, D–39016 Magdeburg

bordihn@cs.tu-magdeburg.de

Henning Fernau

Wilhelm–Schickard–Institut für Informatik
Universität Tübingen
Sand 13, D–72076 Tübingen

fernau@informatik.uni-tuebingen.de

1 Introduction

We continue our research about accepting grammars and systems [1, 2, 4] where it is shown that the trivial equivalence between generating and accepting type-i grammars of the Chomsky hierarchy (e.g. see [5]) does not hold for several grammar types. One example for this phenomenon provide programmed grammars (with appearance checking and context-free core rules) which are (strictly) more powerful in accepting mode than in generating mode. More precisely, if we admit erasing productions we arrive at new characterizations of the recursively enumerable languages, and if we do not admit them we get new characterizations of the context-sensitive languages. For the simulation of the type-0 or type-1 grammars, respectively, an extension of the nonterminal alphabet is needed.

In this paper, we consider the pure versions of this grammar type where we give up the distinction between terminal and nonterminal symbols. Such pure grammars are of interest because of the following reasons:

- In some sense, they form a sequential counterpart to Lindenmayer systems.
- The "intermediate" sentential forms remain regarded in the language such that no information about the derivation process is lost. Therefore, we can hope to shed new light on the role of the auxiliary symbols in the simulation of generating grammars by accepting ones: Is it possible to keep the simulation features also in the pure case or do we find new relationships between the families of languages generated and accepted by pure programmed grammars?

A crucial thing is how to define the yield relation in accepting grammars. We follow this general idea: We textually transfer the given definition of a derivation step in generating grammars to the accepting case (for a short discussion, see [1, 2]).

Conventions: $\subseteq$ denotes inclusion, $\subset$ denotes strict inclusion, $\#M$ is the number of elements in the set M. The empty word is denoted by λ. Let $\mathrm{Sub}(w)$, $(\mathrm{Suf}(w)$, $\mathrm{Pref}(w))$ denote the set of subwords (suffixes, prefixes) of w. The length of a word x is denoted by $|x|$. If $x \in V^*$, where V is some alphabet, and if $W \subseteq V$, then $|x|_W$ denotes the number of occurrences of letters from W in x. If W is a singleton set $\{a\}$, we simply write $|x|_a$ instead of $|x|_{\{a\}}$.

Let $V = \{a_1, a_2, \ldots, a_n\}$ be an alphabet, the order of its elements is considered to be fix. To a word $x \in V^*$ we associate its *Parikh vector* $p_V(x) = (|x|_{a_1}, |x|_{a_2}, \ldots, |x|_{a_n})$. Then, for $x \in V^*$, we define the set of all *permutations of* x by $\mathrm{Perm}(x) = \{y \in V^* \mid p_V(y) = p_V(x)\}$.

2 Definitions

A *pure programmed grammar* ([3]) is a triple $G = (V, P, S)$, where V is some alphabet, $S \subseteq V^+$ is a finite set of axioms, and P is a finite set of productions of the form $(r : u \to v, \sigma(r), \phi(r))$, where $r : u \to v$ is a rewriting rule $u \to v$ labelled by r, $u \in V^*$, $v \in V^*$, and $\sigma(r)$ and $\phi(r)$ are two sets of labels of such core rules in P. By $\mathrm{Lab}(P)$ we denote the set of all labels of the productions appearing in P.

A sequence of words over V, $y_0, y_1, \ldots, y_n$, is referred to as a derivation in G iff, for $1 \leq i \leq n$, there are productions $(r_i : u_i \to v_i, \sigma(r_i), \phi(r_i)) \in P$ such that

$$y_{i-1} = z_{i-1} u_i z'_{i-1},\ y_i = z_{i-1} v_i z'_{i-1},\ \text{and, if } 1 \leq i < n,\ r_{i+1} \in \sigma(r_i)$$

or

$$u_i \notin \mathrm{Sub}(y_{i-1}),\ y_{i-1} = y_i,\ \text{and, if } 1 \leq i < n,\ r_{i+1} \in \phi(r_i).$$

In this case, we also write the derivation as

$$y_0 \underset{r_1}{\Longrightarrow} y_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_n}{\Longrightarrow} y_n$$

or simply as $y_0 \Rightarrow y_1 \Rightarrow \cdots \Rightarrow y_n$. Moreover, we use the notation $y_0 \overset{n}{\Rightarrow} y_n$. Sometimes, we simply write $x \Rightarrow y$ iff $x \overset{1}{\Rightarrow} y$ and $x \overset{*}{\Rightarrow} y$ iff $x \overset{n}{\Rightarrow} y$ for some $n \geq 0$.

The set $\sigma(r_1)$ is called success field and the set $\phi(r_1)$ failure field of r_1. For any $r \in \mathrm{Lab}(P)$, we set $\psi^+(r) = \bigcup_{i \geq 1} \psi^i(r)$, where, for $i \geq 1$, $\psi^i(r)$ is defined in the following way:

$$\begin{aligned} \psi^1(r) &= \sigma(r) \cup \phi(r) \\ \psi^{i+1}(r) &= \bigcup_{p \in \psi^i(r)} \psi^1(p)\,. \end{aligned}$$

The language generated by G is defined as
$L_{\mathrm{gen}}(G) = \{w \in V^* \mid \text{there is a derivation } s = y_0, y_1, \ldots, y_n = w$

for some $s \in S\}$, the language accepted by G is defined as
$L_{\text{acc}}(G) = \{w \in V^* \mid \text{there is a derivation } w = y_0, y_1, \ldots, y_n = s$ for some $s \in S\}$.[1]

The family of languages generated (accepted) by pure programmed grammars of the form $G = (V, P, S)$ containing only context-free core rules is denoted by $\mathcal{L}_{\text{gen}}(\text{pP,CF}, \text{ac})$ ($\mathcal{L}_{\text{acc}}(\text{pP,CF}, \text{ac})$). When no appearance checking features are involved, i.e. $\phi(r) = \emptyset$ for each rule in P, we are led to the families $\mathcal{L}_{\text{gen}}(\text{pP,CF})$ ($\mathcal{L}_{\text{acc}}(\text{pP,CF})$). The special variant of a pure programmed grammar where the success field and the failure field coincide for each rule in the set P of productions is said to be a pure programmed grammar with *unconditional transfer.* According to the notation which has been introduced up to here, we shall denote the class of languages generated (accepted) by pure programmed grammars with context-free productions and with unconditional transfer by $\mathcal{L}_{\text{gen}}(\text{pP,CF,ut})$ ($\mathcal{L}_{\text{acc}}(\text{pP,CF,ut})$). If erasing rules are forbidden, we replace the component CF by CF–λ in that notation.

3 Results

Considering pure programmed grammars without appearance checking one can easily carry over the technique for proving a trivial equivalence between accepting and generating mode of nonpure grammars (see [1, 2]). This yields our first statement.

Theorem 3.1 For $\alpha \in \{\text{CF,CF–}\lambda\}$, we have

$$\mathcal{L}_{\text{gen}}(\text{pP}, \alpha) = \mathcal{L}_{\text{acc}}(\text{pP}, \alpha)$$

. □

[1]Note that a pure programmed grammar can only be considered to be in generating mode if there is no production with core rule $\lambda \to v$. Analogously, an accepting pure programmed grammar cannot have core rules of the form $u \to \lambda$.

As already mentioned, the proving ideas for the case of programmed grammars with appearance checking are not transferable from the "classical" case to the pure case, since we need auxiliary symbols.

In order to clarify the above listed questions, we state a simple property of pure programmed grammars with context-free core rules in generating mode, first. Since the fact whether a production of such a grammar $G = (V, P, S)$ is applicable to some string $x \in V^*$ or not does not depend on the order of the symbols in x, we find the following Lemma.

Lemma 3.2 Let $G = (V, P, S)$ be a generating pure programmed grammar with context-free core rules and $x \in V^+$, $y \in V^*$. If there is a derivation $x \overset{*}{\Rightarrow} y$ in G of the form

$$x = y_0 \underset{r_1}{\Longrightarrow} y_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_n}{\Longrightarrow} y_n = y\,,$$

then, for any $x' \in \mathrm{Perm}(x)$, there is a $y' \in \mathrm{Perm}(y)$ and a derivation

$$x' = y'_0 \underset{r_1}{\Longrightarrow} y'_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_n}{\Longrightarrow} y'_n = y'\,,$$

where $y'_i \in \mathrm{Perm}(y_i)$, for $0 \le i \le n$.

Proof (by induction over the length n of derivation).
For $n = 0$ the statement is obvious. Assume the statement to be proved for $n = k$ and consider an arbitrary derivation of the form

$$x = y_0 \underset{r_1}{\Longrightarrow} y_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_k}{\Longrightarrow} y_k \underset{r_{k+1}}{\Longrightarrow} y_{k+1}\,.$$

Let $(r_{k+1} : a \to v, \sigma(r_{k+1}), \phi(r_{k+1}))$. If $|y_k|_a = 0$, then, for any $y'_k \in \mathrm{Perm}(y_k)$, we have $|y'_k|_a = 0$ and, if $y'_k \underset{r_{k+1}}{\Longrightarrow} y'_{k+1}$ then $y'_{k+1} = y'_k \in \mathrm{Perm}(y_k) = \mathrm{Perm}(y_{k+1})$. Otherwise, there are words z_1 and z_2 such that $y_k = z_1 a z_2$ and $y_{k+1} = z_1 v z_2$. Then, for any $y'_k \in \mathrm{Perm}(y_k)$, there are words z'_1 and z'_2 such that $y'_k = z'_1 a z'_2$. Thus, $y'_k \underset{r_{k+1}}{\Longrightarrow} z'_1 v z'_2 = y'_{k+1}$ and $y'_{k+1} \in \mathrm{Perm}(y_{k+1})$. In both cases, for any $x' \in \mathrm{Perm}(x)$, we have

$$x' = y'_0 \underset{r_1}{\Longrightarrow} y'_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_k}{\Longrightarrow} y'_k \underset{r_{k+1}}{\Longrightarrow} y'_{k+1}\,,$$

with $y'_i \in \mathrm{Perm}(y_i)$, for $0 \leq i \leq k+1$. □

Theorem 3.3 For $\alpha \in \{\mathrm{CF}, \mathrm{CF}{-}\lambda\}$, we have

(i) $\mathcal{L}_{\mathrm{gen}}(\mathrm{pP}, \alpha, \mathrm{ac}) \subset \mathcal{L}_{\mathrm{acc}}(\mathrm{pP}, \alpha, \mathrm{ac})$

(ii) $\mathcal{L}_{\mathrm{gen}}(\mathrm{pP}, \alpha, \mathrm{ut})$ and $\mathcal{L}_{\mathrm{acc}}(\mathrm{pP}, \alpha, \mathrm{ut})$ are incomparable.

Proof. i) First, we show the inclusions.

Let $G = (V, P, S)$ be a pure context-free programmed grammar with appearance checking in generating mode. For $r \in \mathrm{Lab}(P)$, we set $\sigma'(r) = \{p \in \mathrm{Lab}(P) \mid r \in \sigma(p)\}$ and $\phi'(r) = \{p \in \mathrm{Lab}(P) \mid r \in \phi(p)\}$, and with every rule $(r : a \to v, \sigma(r), \phi(r)) \in P$ we associate the productions

(1) $(r' : v \to a, \{p' \mid p \in \sigma'(r)\} \cup \{p'' \mid p \in \phi'(r)\}, \emptyset)$ and

(2) $(r'' : a \to a, \emptyset, \{p' \mid p \in \sigma'(r)\} \cup \{p'' \mid p \in \phi'(r)\})$.

We construct an accepting pure programmed grammar $G' = (V, P', S)$, P' containing all productions of form (1) or (2) as above. Obviously, G' has only (accepting) context-free core rules.

In the following, we prove for any words $s \in S$ and $x \in V^*$, that there is a derivation

$$s = y_0 \underset{r_1}{\Longrightarrow} y_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_{n-1}}{\Longrightarrow} y_{n-1} \underset{r_n}{\Longrightarrow} y_n = x$$

according to G if and only if there is a derivation

$$x = y_n \underset{q_n}{\Longrightarrow} y_{n-1} \underset{q_{n-1}}{\Longrightarrow} \cdots \underset{q_2}{\Longrightarrow} y_1 \underset{q_1}{\Longrightarrow} y_0 = s$$

according to G', where, for $1 \leq i \leq n$, $q_i \in \{r'_i, r''_i\}$. The proof is given by induction over the length n of derivation. For $n = 0$, the assertion trivially holds. Now, we show the assertion for $n = k+1$ assuming that it has been verified for $n = k$.

($\Rightarrow$) Consider, for some $s \in S$, the derivation

$$s = y_0 \underset{r_1}{\Longrightarrow} y_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_{k-1}}{\Longrightarrow} y_{k-1} \underset{r_k}{\Longrightarrow} y_k \underset{r_{k+1}}{\Longrightarrow} y_{k+1} = x$$

in G and let $(r_{k+1} : a \to v,\ \sigma(r_{k+1}),\ \phi(r_{k+1}))$.
Case 1. $r_{k+1} \in \sigma(r_k)$, i.e., the production r_k is successfully applied to y_{k-1} and, by our construction, the dual r'_k of r_k is applicable to y_k yielding y_{k-1}. If there are words z_1 and z_2 such that $y_k = z_1 a z_2$ then $y_{k+1} = z_1 v z_2$. Thus, by G', $y_{k+1} \underset{r'_{k+1}}{\Longrightarrow} y_k$ and, as $r_k \in \sigma'(r_{k+1})$, $r'_k \in \sigma(r'_{k+1})$. Hence, the production r'_k can be applied to y_k, now. If $|y_k|_a = 0$ then $y_{k+1} = y_k$. Therefore, $y_{k+1} \underset{r''_{k+1}}{\Longrightarrow} y_k$ and $r'_k \in \phi(r''_{k+1})$.
Case 2. $r_{k+1} \in \phi(r_k)$, i.e., the production r_k is not applicable to $y_{k-1} = y_k$ and, consequently, also the production $r''_k \in P'$ cannot be applied to y_k. Let $y_k = z_1 a z_2$ for some z_1, z_2, again. Then $y_{k+1} \underset{r'_{k+1}}{\Longrightarrow} y_k$ and, as $r_k \in \phi'(r_{k+1})$, $r''_k \in \sigma(r'_{k+1})$ garanteeing $y_k \underset{r''_k}{\Longrightarrow} y_{k-1}$. In case of $|y_k|_a = 0$ we have $y_{k+1} \underset{r''_{k+1}}{\Longrightarrow} y_k$ as in Case 1 and $r''_k \in \phi(r''_{k+1})$.

($\Leftarrow$) Consider the derivation

$$x = y_{k+1} \underset{q_{k+1}}{\Longrightarrow} y_k \underset{q_k}{\Longrightarrow} y_{k-1} \underset{q_{k-1}}{\Longrightarrow} \cdots \underset{q_2}{\Longrightarrow} y_1 \underset{q_1}{\Longrightarrow} y_0 = s$$

according to G', where $s \in S$ and, for $1 \leq i \leq k+1$, $q_i \in \{r'_i, r''_i\}$. Let $q_{k+1} = r'_{k+1}$ with $(r'_{k+1} : v \to a,\ \{p' \mid p \in \sigma'(r_{k+1})\} \cup \{p'' \mid p \in \phi'(r_{k+1})\},\ \emptyset)$. Then there are words z_1 and z_2 such that $y_{k+1} = z_1 v z_2$ and $y_k = z_1 a z_2$. For $k = 1$, we successfully simulated the above derivation. For $k > 1$, we have to distinguish the following two cases. If $q_k = r'_k \in \{p' \mid p \in \sigma'(r_{k+1})\}$ then the core rule of this production must be applicable to y_k (otherwise, we do not obtain the above derivation). Consequently, we have $y_{k-1} \underset{r_k}{\Longrightarrow} y_k \underset{r_{k+1}}{\Longrightarrow} y_{k+1}$ with $r_{k+1} \in \sigma(r_k)$ according to G. If $q_k = r''_k \in \{p'' \mid p \in \phi'(r_{k+1})\}$ then (in order to get the above derivation) this production is not applicable to y_k. Thus, also the production r_k cannot be applied to $y_{k-1} = y_k$ and we have $y_{k-1} = y_k \underset{r_k}{\Longrightarrow} y_k \underset{r_{k+1}}{\Longrightarrow} y_{k+1}$ with $r_{k+1} \in \phi(r_k)$ according to G. The case $q_{k+1} = r''_{k+1}$ can be proved analogously.

In conclusion, $L(G') = L(G)$. Clearly, if G is a pure programmed grammar without erasing, then G' has this property as well.

ii) Now, we prove that these inclusions are proper. For, let

$$L_1 = \{a^n b^n \,|\, n \geq 1\} \cup \{a^{m+1} b a^n b^k \,|\, m \geq 1,\, n \geq 1,\, m+n = k\}\,.$$

We are going to prove that $L_1 \in \mathcal{L}_{\text{acc}}(\text{pP,CF}{-}\lambda, \text{ut}) \setminus \mathcal{L}_{\text{gen}}(\text{pP, CF, ac})$.

(a) $L_1 = L_{\text{acc}}(G_1)$ for the pure programmed grammar (with unconditional transfer) $G_1 = (\{a, b, F\},\, P_1,\, \{ab\})$, where

$$P_1 = \{(1:\; aab \to a,\, \{2\},\, \{2\}),\, (2:\; ba \to F,\, \{1\},\, \{1\})\}\,.$$

This can be seen as follows. There is no production in P_1 for replacing the symbol F and ab is the only axiom. Hence, $L_{\text{acc}}(G_1) \subseteq \{a, b\}^+$.

Let $w \in L_{\text{acc}}(G_1)$. Clearly, if $ba \notin \text{Sub}(w)$ then $w = a^n b^n$ for some $n \geq 1$. Let $ba \in \text{Sub}(w)$. Then, there is a derivation $w \Rightarrow v \overset{*}{\Rightarrow} ab$ for some $v \in \{a, b\}^+$ with $ba \notin \text{Sub}(v)$ (in case of $ba \in \text{Sub}(v)$, for any v' with $v \Rightarrow v'$, we have $|v'|_F = 1$, but this contradicts $v \overset{*}{\Rightarrow} ab$). Since $v \overset{*}{\Rightarrow} ab$, $v \in L_{\text{acc}}(G_1)$. In conclusion, $v = a^k b^k$ for some $k \geq 1$, and w must be of the form $a^{m+1} b a^n b^k$ with $m \geq 1$, $n \geq 1$, $m + n = k$.

Therefore, $L_{\text{acc}}(G_1) = L_1$ and $L_1 \in \mathcal{L}_{\text{acc}}(\text{pP,CF}{-}\lambda, \text{ut})$.

(b) $L_1 \notin \mathcal{L}_{\text{gen}}(\text{pP,CF,ac})$.

Assume the contrary, and let $G = (V, P, S)$ be a pure programmed grammar with $L_{\text{gen}}(G) = L_1$. First, we prove the following

Claim 1. For any word $w \in \{a^{m+1} b a^n b^k \,|\, m \geq 1,\, n \geq 1,\, m+n = k\}$ there is no $v \neq w$ such that $w \Rightarrow v$ by G.

Let $w = a^{m+1} b a^n b^k \in L_1$ and $(r : x \to y,\, \sigma(r),\, \phi(r))$ be a production applicable to w such that $s \overset{*}{\Rightarrow} w \underset{r}{\Rightarrow} v$ for some $s \in S$. If $x = b$ then the production is applicable to the first b. Therefore, $|y|_b \leq 1$. In case of $|y|_b = 0$ we arrive a word $a^t b^k$ with $t \geq k+1$ which is not contained in L_1. In case of $|y|_a \geq 1$ and $|y|_b = 1$ we derive a word of the form $a^{t_1+1} b a^{t_2} b^k$ with $t_1 + t_2 > k$ what contradicts the structure of the words in L_1, again. Hence, $y = b$.

Now, let $x = a$. Since we can apply this production to the first a in w, we have $|y|_b = 0$. Thus, $y = a$. In conclusion, $v = w$ holds, and the proof of Claim 1 is finished.

Let t be the maximum length of an axiom in S. For any production $(r : x \to y, \sigma(r), \phi(r))$ in P, we set $\lg(r) = |y| - 1$, and

$$\lg(G) = \max\{\lg(r) \mid r \in \mathrm{Lab}(P)\}\,.$$

Obviously, the next statement is a direct consequence of Claim 1.

Claim 2. For $n > t + 2\lg(G)$, there exists a derivation in G which is of the form

$$s \overset{*}{\Rightarrow} w_1 \overset{*}{\Rightarrow} w_1 \underset{p_1}{\Rightarrow} w_2 \overset{*}{\Rightarrow} w_2 \underset{p_2}{\Rightarrow} a^n b^n\,,$$

where $s \in S$, $w_1 = a^{k_1} b^{k_1}$, $w_2 = a^{k_2} b^{k_2}$, and $1 < k_1 < k_2 < n$.

Now, let $(p_1 : x \to y, \sigma(p_1), \phi(p_1))$. We assume that $x = a$. Clearly, $p_2 \in \psi^+(p_1)$, and y must be of the form $a^{k_2-k_1+1} b^{k_2-k_1}$. Since the core rule $x \to y$ is also applicable to the first a in w_1, according to Lemma 3.2 we have a derivation

$$s \overset{*}{\Rightarrow} w_1 \underset{p_1}{\Rightarrow} a^{k_2-k_1+1} b^{k_2-k_1} a^{k_1-1} b^{k_1} \overset{*}{\Rightarrow} a^{k_2-k_1+1} b^{k_2-k_1} a^{k_1-1} b^{k_1} \underset{p_2}{\Rightarrow} w'$$

with $w' \neq a^{k_2-k_1+1} b^{k_2-k_1} a^{k_1-1} b^{k_1}$ in G. This contradicts Claim 1.

By analogous arguments one can prove that also the assumption $x = b$ yields a contradiction.

iii) Finally, it is left to show that $\mathcal{L}_{\mathrm{gen}}(\mathrm{pP,CF}\text{–}\lambda, \mathrm{ut}) \setminus \mathcal{L}_{\mathrm{acc}}(\mathrm{pP,CF,ut}) \neq \emptyset$. Let us consider the following generating pure programmed grammar with unconditional transfer $G_2 = (\{a, b, c, d\}, P_2, \{d\})$, where P_2 consists of the productions

$$\begin{array}{ll}
(1 : d \to ac^2b,\ \{2\}), & (2 : b \to b^{11},\ \{3,4\}), \\
(3 : a \to a^{11},\ \{2\}), & (4 : a \to c^4,\ \{5\}), \\
(5 : b \to d^{11},\ \{5\}). &
\end{array}$$

(for the sake of convenience, we write the productions of grammars with unconditional transfer as pairs $(r : u \to v, \sigma(r))$ omitting the failure field $\phi(r) = \sigma(r)$). By G_2, the initial derivation is $d \underset{1}{\Longrightarrow} ac^2b \underset{2}{\Longrightarrow} ac^2b^{11}$. Starting from a string $a^{10(i-1)+1}c^2b^{10i+1}$ after the application of rule 2, we either may increase the number of a's and b's by 10 with $a^{10(i-1)+1}c^2b^{10i+1} \underset{3}{\Longrightarrow} a^{10i+1}c^2b^{10i+1} \underset{2}{\Longrightarrow} a^{10i+1}c^2b^{10(i+1)+1}$, or we can rewrite an a by c^4 and start replacing b's by d^{11} until we arrive a word of the form $yc^2d^{11(10i+1)}$ with $y \in \{a, c^4\}^{10(i-1)+1}$, $|y|_c = 4$. Since no further derivations are possible, the language generated by G_2 is

$$\begin{aligned} L_2 = \{d\} &\cup \{a^{10n+1}c^2b^{10n+1} \mid n \geq 0\} \cup \{a^{10n+1}c^2b^{10n+11} \mid n \geq 0\} \\ &\cup \{yc^2x \mid y \in \{a, c^4\}^{10n+1},\, |y|_c = 4,\, x \in \{b, d^{11}\}^{10n+11},\, n \geq 0\}\,. \end{aligned}$$

Let us assume that there is an accepting pure programmed grammar with unconditional transfer $G = (V, P, S)$ with $L_{\mathrm{acc}}(G) = L_2$. Since $||w| - |w'|| > 1$ for arbitrary words w and w' in L_2 with $w \neq w'$, we may suppose without loss of generality that there is no λ-rule $(r : \lambda \to z, \sigma(r))$ in P. Thus, d must be an axiom.

In the following, let n be a sufficiently large integer. By the structure of the words in L_2, no accepting context-free rule with a, c, or d on its right-hand side can be applied to a string in $\{yc^2x \mid y \in \{a, c^4\}^{10n+1}, |y|_c = 4, x \in \{b, d^{11}\}^{10n+11}, |x|_d > 0\}$ yielding a word in L_2 (if we let productions with core rules of the form $z \to z$ be disregarded). In fact, the only applicable rule is $d^{11} \to b$. Taking into consideration that, for any $n \geq 0$, the word $c^4a^{10n}c^2d^{11(10n+11)}$ is in L_2, there must be a production

$$(r_0 : d^{11} \to b, \sigma(r_0))$$

in P and $\sigma(r_0)$ is determined in a way that enables a derivation $c^4a^{10n}c^2d^{11(10n+11)} \overset{*}{\Rightarrow} c^4a^{10n}c^2b^{10n+11}$ only using r_0 (besides $z \to z$). Furhtermore, the application of a rule with b, c, or d on its right-hand side to the string $c^4a^{10n}c^2b^{10n+11}$ cannot yield an element of L_2, but only $c^4 \to a$. Thus, there is a production

$$(r_1 : c^4 \to a, \sigma(r_1))$$

in P, $r_1 \in \psi^+(r_0)$ such that, for $x \in \{b, d^{11}\}^{10n+11}$, $|x|_d = 11$,

$$\begin{aligned} c^4 a^{10n} c^2 x &\underset{r_0}{\Longrightarrow} c^4 a^{10n} c^2 b^{10n+11} \overset{*}{\Rightarrow} c^4 a^{10n} c^2 b^{10n+11} \\ &\underset{r_1}{\Longrightarrow} a^{10n+1} c^2 b^{10n+11} . \end{aligned}$$

The word $w = a^{10n+1}c^2b^{10n+11}$ derives a string in L_2 only by applying $b^{11} \to b$. Therefore, we need a production

$$(r_2 : \, b^{11} \to b, \, \sigma(r_2))$$

in P in order to accept w. Analogously, a production

$$(r_3 : \, a^{11} \to a, \, \sigma(r_3))$$

is needed because of $a^{10n+1}c^2b^{10n+1} \in L_2$.

Obviously, $r_2 \in \psi^+(r_3) \cap \psi^+(r_1)$ and $r_3 \in \psi^+(r_2)$ have to hold in such a way that the following derivation is possible:

$$\begin{aligned} &c^4 a^{10n} c^2 b^{10(n+1)+1} \underset{r_1}{\Longrightarrow} a^{10n+1} c^2 b^{10(n+1)+1} \overset{*}{\Rightarrow} a^{10n+1} c^2 b^{10(n+1)+1} \\ &\underset{r_2}{\Longrightarrow} a^{10n+1} c^2 b^{10n+1} \overset{*}{\Rightarrow} a^{10n+1} c^2 b^{10n+1} \underset{r_3}{\Longrightarrow} a^{10(n-1)+1} c^2 b^{10n+1} \\ &\overset{*}{\Rightarrow} a^{10(n-1)+1} c^2 b^{10n+1} \underset{r_2}{\Longrightarrow} \cdots \overset{*}{\Rightarrow} ac^2b \end{aligned}$$

only using r_2 and r_3 (except $z \to z$) after the application of r_1.

Consequently, d is the only axiom. This can be seen as follows. If, for some $k \geq 0$, the word $a^{10k+1}c^2b^{10k+1}$ or $a^{10k+1}c^2b^{10k+11}$ is an axiom, respectively, we get $a^{10k+1}c^2b^{10k}d^{11} \in L_{\text{acc}}(G)$ by $a^{10k+1}c^2b^{10k}d^{11} \underset{r_0}{\Longrightarrow} a^{10k+1}c^2b^{10k+1}$, or $a^{10k+1}c^2b^{10(k+1)}d^{11} \in L_{\text{acc}}(G)$ by $a^{10k+1}c^2b^{10(k+1)}d^{11} \underset{r_0}{\Longrightarrow} a^{10k+1}c^2b^{10k+11}$, respectively. Now, let $y \in \{a, c^4\}^{10k+1}$, $|y|_c = 4$, and $x \in \{b, d^{11}\}^{10k+11}$. If $yc^2x \in S$ then $y'c^2x \in L_{\text{acc}}(G)$ by $y'c^2x \underset{r_1}{\Longrightarrow} yc^2x$, where $y' \in \{a, c^4\}^{10k+1}$, c^4 appears exactly twice in y'.

Now, let k be chosen such that P contains a production $(p : \, a^{10k+1}c^2b^{10k'+1} \to d, \, \sigma(p))$, but no production

$(\overline{p} : a^{10(k+m)+1}c^2b^{10(k'+m)+1} \rightarrow d, \sigma(\overline{p}))$, where either $k' = k$ or $k' = k+1$, $m > 0$. Clearly, if $k' = k$ then $p \in \psi^+(r_2)$ has to hold such that

$a^{10k+1}c^2b^{10(k+1)+1} \underset{r_2}{\Longrightarrow} a^{10k+1}c^2b^{10k+1} \overset{*}{\Rightarrow} a^{10k+1}c^2b^{10k+1} \underset{p}{\Longrightarrow} d\,,$

and, if $k' = k+1$ then $p \in \psi^+(r_3)$ has to hold such that

$a^{10(k+1)+1}c^2b^{10(k+1)+1} \underset{r_3}{\Longrightarrow} a^{10k+1}c^2b^{10(k+1)+1} \overset{*}{\Rightarrow} a^{10k+1}c^2b^{10(k+1)+1} \underset{p}{\Longrightarrow} d$, respectively.

Let $n > k+1$. Then, by G

$$\begin{aligned} & a^{10n+1}c^2b^{10(n+1)}d^{11} \underset{r_0}{\Longrightarrow} a^{10n+1}c^2b^{10(n+1)+1} \overset{*}{\Rightarrow} a^{10n+1}c^2b^{10(n+1)+1} \\ & \underset{r_1}{\Longrightarrow} a^{10n+1}c^2b^{10(n+1)+1} \overset{*}{\Rightarrow} a^{10n+1}c^2b^{10(n+1)+1} \underset{r_2}{\Longrightarrow} a^{10n+1}c^2b^{10n+1} \\ & \overset{*}{\Rightarrow} a^{10n+1}c^2b^{10n+1} \underset{r_3}{\Longrightarrow} a^{10(n-1)+1}c^2b^{10n+1} \overset{*}{\Rightarrow} a^{10k+1}c^2b^{10k'+1} \\ & \underset{p}{\Longrightarrow} d. \end{aligned}$$

This contradicts $a^{10n+1}c^2b^{10(n+1)}d^{11} \notin L_2$. □

4 Conclusions

By Theorem 3.3, the inclusions $\mathcal{L}_{\text{gen}}(\text{P,CF}[-\lambda]\text{,ac}) \subseteq \mathcal{L}_{\text{acc}}(\text{P,CF}[-\lambda]$, ac) are proved once more in a "direct" manner, i.e. without simulating type-0 grammars or type-1 grammars, respectively. On the other hand, in our direct simulation "real' appearance checks are needed such that this proof cannot be used for pure programmed grammars with unconditional transfer. Indeed, we have an incomparability result for this case, a relationship between accepting and generating mode which was not found for the grammar classes considered in our former papers. Thus, the only relationship which has not been proved to be correct for a special grammar type is that the generating mode is strictly more powerful than the accepting one.

We summerize the results about pure programmed grammars as follows. For $\alpha \in \{\text{CF,CF}-\lambda\}$, we have:

- $\mathcal{L}_{\text{gen}}(\text{pP},\alpha) = \mathcal{L}_{\text{acc}}(\text{pP},\alpha)$
- $\mathcal{L}_{\text{gen}}(\text{pP},\alpha,\text{ac}) \subset \mathcal{L}_{\text{acc}}(\text{pP},\alpha,\text{ac})$
- $\mathcal{L}_{\text{gen}}(\text{pP},\alpha,\text{ut})$ and $\mathcal{L}_{\text{acc}}(\text{pP},\alpha,\text{ut})$ are incomparable.

References

[1] H. Bordihn and H. Fernau. Accepting grammars and systems. Technical Report 9/94, Universität Karlsruhe, Fakultät für Informatik, 1994.

[2] H. Bordihn and H. Fernau. Accepting grammars with regulation. *International Journal of Computer Mathematics*, 53:1–18, 1994.

[3] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science*. Berlin: Springer, 1989.

[4] H. Fernau and H. Bordihn. Remarks on accepting parallel systems. *International Journal of Computer Mathematics*, 56:51–67, 1995.

[5] A. K. Salomaa. *Formal Languages*. Academic Press, 1973.

On Constructible Functions in Cellular Automata

Thomas Buchholz

AG Informatik
Universität Gießen
Arndtstr. 2, D–35392 Giessen

buchholz@informatik.uni-giessen.de

Abstract

Up to now nobody knows if there exists a language which is acceptable by a cellular automaton but requires more than real-time. Our approach is to investigate time constructibility in cellular automata to obtain some impulses towards a time hierarchy.

Therefore we compare the constructibility capabilities of (two-way) cellular automata with one-way cellular automata, and we give a characterization of OCA-constructible functions by a special family of languages.

1 Introduction

The classical grammar-based language theory classifies languages by their structural complexity. Another classification, called computational complexity, is based on the amount of time, space, or other resources needed to recognize a language on some device, such as Turing machines or cellular automata.

As far as Turing machines are concerned it is well known that there exists an infinite time hierarchy. One way to obtain this result is by diagonalization (using the fact that an universal Turing machine exists) with respect to so-called time constructible functions, i.e. an input word of length n will cause a Turing machine to make exactly $f(n)$ steps before accepting the word. For cellular automata a corresponding result could not be proved up to now. Today nobody even knows if there exits a language which is acceptable by a cellular automata but requires more then real-time.

Our approach is to investigate time constructibility in cellular automata to reach similar results as for Turing machines. Since the cellular automata model does not contain a universal automaton, we cannot proceed as in the case of Turing machines at all. But we can hope if we understand the notion of time constructibility well enough to obtain impulses towards a time hierarchy.

2 Definitions

We will give the basic definitions needed in the next section.

Definition.
A *cellular automaton* (CA) is a quintuple $\mathcal{A} = (S, \sigma, q_0, \#, F)$, where

1. S is the finite, nonempty set of *states*.
2. $F \subseteq S$ is the set of *accepting states*.
3. $\sigma : (S \cup \{\#\})^3 \longrightarrow S \cup \{\#\}$ is the *local transition function*.
4. $q_0 \in S$ is the *quiescent state* such that $\sigma(q_0, q_0, q_0) = q_0$.
5. $\# \notin S$ is the *boundary state* such that $\sigma(s, t, p) = \# \iff t = \#$ for all $s, t, p \in S \cup \{\#\}$.

A *configuration* is a mapping $c : \mathbb{Z} \longrightarrow S$. The set of all configurations is denoted by C. So, on C the local transition function σ induces a *global transition function* $\mathcal{T} : C \longrightarrow C$, where for all $c \in C$ it holds

$$(\mathcal{T}(c))(i) = \sigma(c(i-1), c(i), c(i+1)) \text{ for all } i \in \mathbb{Z}.$$

For a word $w = a_1 \cdots a_n \in A^+$ where $A \subseteq S$ the *initial configuration* c_w is defined as follows:

$$c_w(i) = \begin{cases} a_i & 1 \leq i \leq n \\ \# & otherwise \end{cases} .$$

Definition.
A word $w \in A^+$ is *accepted* by a cellular automaton in $n \in \mathbb{N}_0$ time steps if and only if n is the smallest natural number such that $(\mathcal{T}^n(c_w))(|w|) \in F$.

Definition.
A formal language $L \subseteq A^+$ is *accepted* by a cellular automaton with time complexity $t : \mathbb{N} \longrightarrow \mathbb{N}$ if and only if $A \subseteq S$ and

$$L = \{w \in A^+ | (\mathcal{T}^{t(|w|)}(c_w))(|w|) \in F\}.$$

The family of all languages which can be accepted by a cellular automaton in real-time (i.e. $t = id$) is denoted by $\mathscr{L}_{rt}(CA)$.

Definition.
A function $f : \mathbb{N} \longrightarrow \mathbb{N}$ is *CA-constructible* if and only if there exists a cellular automaton which accepts ${q_0}^n$ in $f(n)$ time steps for all $n \in \mathbb{N}$. The set of all CA-constructible functions is denoted by $\mathscr{C}(CA)$.

A restriction of the information flow to the right will lead to one-way cellular automata:

Definition.
An *one-way cellular automaton* (OCA) is a cellular automaton such that it holds

$$\sigma(s, t, p) = \sigma(s, t, q) \text{ for all } s, t, p, q \in S \cup \{\#\}.$$

So, in one-way cellular automata the behaviour of the local transition function σ is independent of the third state. In fact, we will often identify σ with the induced 2-ary local transition function depending only on the first two states.

Notions such as accepting words and languages and constructibility of functions are transferred analogously to one-way cellular automata. Also the meaning of $\mathscr{L}_{rt}(OCA)$ and $\mathscr{C}(OCA)$ should be clear.

3 Results

Now we are able to state some of the results we obtained.

3.1 Unidirectionality versus bidirectionality

Comparing the capabilities of (two-way) cellular automata with one-way cellular automata we are able to show that there exist CA-constructible functions which are not OCA-constructible.

Lemma.
If $f \geq id$ is OCA-constructible it holds that $f \geq id + \lfloor log_b \rfloor$ for some $b \in \mathbb{N}$ or there exists a $c \in \mathbb{N}$ such that $f(n) - n = c$ for infinitely many $n \in \mathbb{N}$.

Proof. Suppose there exists a function f which is constructible by an one-way cellular automaton $\mathcal{A} = (S, \sigma, q_0, \#, F)$, and $f \not\geq id + \lfloor log_b \rfloor$ for all $b \in \mathbb{N}$. Then we can find an $n_0 \in \mathbb{N}$ such that $f(n_0) - n_0 < \lfloor log_{|S|+1}(n_0) \rfloor$.

Since the information flow in an one-way cellular automaton is from left to right we can assume the infinite input word ${q_0}^{\omega}$ (with initial configuration $\#^{\omega}{q_0}^{\omega}$) in order to investigate the behaviour of $\mathcal{A}$. Let $l = \lfloor log_{|S|+1} \rfloor$, and denote for all $n \in \mathbb{N}$ by w_n the word of length l over S which results from the states of cell n from time n up to

$n+l-1$, i.e.

$$w_n = c_n(n)c_{n+1}(n)\cdots c_{n+l-1}(n).$$

In total, there are

$$|S|^l = |S|^{\lfloor log_{|S|+1}(n_0)\rfloor} < (|S|+1)^{\lfloor log_{|S|+1}(n_0)\rfloor} \leq n_0$$

words of length l over S. Hence, the pigeonhole principle ensures that at least two of the words $w_1, \ldots, w_{n_0}$, say w_i and w_j $(i \neq j)$, are equal.

Since $\mathcal{A}$ works deterministically and $c_n(n+1) = q_0$ for all $n \in \mathbb{N}$, the word w_n determines the word w_{n+1} uniquely, and so, each word w_m with $m \geq n+1$. Therefore, $w_i = w_j$ implies $w_{n_0-(j-i)} = w_{n_0}$, and inductively it follows $w_{n_0+k(j-i)} = w_{n_0}$ for all $k \geq -1$.

Let now $c = f(n_0) - n_0$, and fix a $k \geq -1$. Since $w_{n_0+k(j-i)} = w_{n_0}$, cell $n_0+k(j-i)$ enters an accepting state firstly at time $n_0+k(j-i)+c$, i.e. $f(n_0+k(j-i)) = n_0+k(j-i)+c$, and hence $f(n_0+k(j-i)) - n_0 - k(j-i) = c$ which proves the lemma. $\square$

Theorem.

$$\mathscr{C}(OCA) \subset \mathscr{C}(CA).$$

Proof. Obviously, we only have to show that the inclusion is proper. Therefore, we consider the function $f = id + \lfloor log_2 \rfloor \circ \lfloor log_2 \rfloor$. The previous lemma shows that f cannot be OCA-constructible.

That f is CA-constructible can be seen as follows (rather than giving a formal construction we will describe how such a cellular automaton could work): A signal from the right is sent to the left such that it triggers $\lfloor log_2(n) \rfloor$ signals from the right as in the solution of the firing squad synchronization problem by Waksman[7]. Another signal is sent from the left to the right which carries along a binary counter that is increased each time one of the triggered signals meets this signal. The cellular automaton assumes an accepting state after the left side of that counter has reached the right boundary cell. Since the counter consists of $\lfloor log_2(\lfloor log_2(n) \rfloor) \rfloor$ bits at that moment, the cellular automaton accepts the input word after exactly $f(n)$ time-steps. This proves the theorem. $\square$

3.2 Characterization of constructible functions

We introduce a family of languages which allows us to characterize OCA-constructible functions. Further we can show that CA-constructible functions cannot be characterized in that way.

For a function $f : \mathbb{N} \longrightarrow \mathbb{N}$ where $f \geq id$ let

$$L_f = \{a^{f(n)-n}b^n | n \in \mathbb{N}\} \subseteq a^*b^+.$$

Omitting a tedious proof we state:

Theorem.
Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function with $f \geq id$. Then it holds

$$f \in \mathscr{C}(OCA) \iff L_f \in \mathscr{L}_{rt}(OCA).$$

Theorem.
For each $f \in \mathscr{C}(CA)$ there exists a $c \in \mathbb{N}$ such that

$$\lim_{n\to\infty} \frac{f(n)}{c^n} = 0.$$

Proof. It suffices to show that there is a $c \in \mathbb{N}$ such that the sequence $\{\frac{f(n)}{c^n}\}_{n\geq 1}$ is bounded. Namely, if $\frac{f(n)}{c^n} \leq K$ for all $n \in \mathbb{N}$, than the change from c to $2c$ yields, since

$$0 \leq \frac{f(n)}{(2c)^n} = \frac{1}{2^n}\frac{f(n)}{c^n} \leq \frac{1}{2^n}K \longrightarrow 0,$$

a number with the desired property.

We suppose now that there exists a function f which is CA-constructible by a cellular automaton $\mathcal{A} = (S, \sigma, q_0, \#, F)$, and there does not exist a $c \in \mathbb{N}$ such that $\{\frac{f(n)}{c^n}\}_{n \geq 1}$ is bounded. So, we can find especially an $n_0 \in \mathbb{N}$ such that $f(n_0) > |S|^{n_0}$. Since $\mathcal{A}$ passes through at most $|S|^{n_0}$ different configurations after an input of ${q_0}^{n_0}$, the behaviour of the rightmost non boundary cell must by cyclically after at most $|S|^{n_0}$ time steps. Therefore, since $f(n_0) > |S|^{n_0}$, this cell never enters an accepting state in contradiction to the constructibility of f.
□

Corollary.
There exists a function $f : \mathbb{N} \longrightarrow \mathbb{N}$ with $f \geq id$ such that $L_f \in \mathscr{L}_{rt}(CA)$ but $f \notin \mathscr{C}(CA)$.

Proof. One can show that $\{a^{2^{2^n}} b^n | n \in \mathbb{N}\} \in \mathscr{L}_{rt}(CA)$, but $2^{2^{id}} + id$ is not CA-constructible as the previous theorem shows. □

References

[1] Balzer, R. *An 8-State Minimal Time Solution to the Firing Squad Synchronization Problem*. Information and Control 10 (1967), 22–42.

[2] Buchholz, Th. and Kutrib, M. *On time constructibility of functions in one-way cellular automata*. Report 9502, AG Informatik, University of Giessen, Giessen, 1995.

[3] Hopcroft, J. and Ullman, J. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[4] Mazoyer, J. and Terrier, V. *Signals in one dimensional cellular automata*. Research Report No. 94-50, LIP ENS 1994.

[5] Smith III, A. R. *Real-time language recognition by one-dimensional cellular automata*. Journal of Computer and System Sciences 6 (1972), 233–253.

[6] Vollmar, R. *Algorithmen in Zellularautomaten*. Teubner, Stuttgart, 1979.

[7] Waksman, A. *An optimum solution to the firing squad synchronization problem*. Information and Control 9 (1966), 66–78.

Einige Bemerkungen zum Pumpen*

Gerhard Buntrock†

Institut für Informatik
Universität Würzburg‡
D–97072 Würzburg

bunt@informatik.uni-wuerzburg.de

„Pumpinglemmata" oder, wie sie auch manchmal genannt werden, „Schleifenlemmata" sind eine häufig angewandte Methode, mit der man für Sprachen nachweist, daß sie nicht regulär oder nicht kontextfrei sind.

Um zu beweisen, daß es für eine Sprachklasse kein Pumpinglemma gibt, werden wir die folgende Definition verwenden:

Definition 1 *Wir sagen, eine Sprache L erfüllt ein Pumpinglemma, wenn gilt:*
Es gibt ein $n \in \mathbb{N}$, so daß für alle $z \in L$ mit $|z| \geq n$ gilt, es gibt eine Zerlegung $z = uvw$, so daß für unendlich viele $i \in \mathbb{N}$ die Wörter $u'v^iw'$ zu L gehören, wobei u' und w' von u, w und i abhängen.

Nun gibt es eine unendliche Folge F von Zeichen aus $\{0, 1\}$, so daß es für kein $i > 2$ eine endliche Teilfolge z von F gibt, von der Form $z = v^i$. Eine solche Folge ist 1906 von Axel Thue und 1921 von

*Diese Arbeit wurde zum Teil mit Mitteln aus dem DFG-Projekt Wa 847/1-1 finanziert.

†`http://haegar.informatik.uni-wuerzburg.de/person/mitarbeiter/bunt/`

‡Oktober '95 bis März '96 Medizinische Universität zu Lübeck

Marston Morse (wieder-)entdeckt worden [Thu06, Mor21] (siehe auch [Jac83]):

$$
\begin{array}{cccccccccccccccc}
0 & & & & & & & & & & & & & & & \\
0 & & & & & & & & 1 & & & & & & & \\
0 & & & & 1 & & & & 1 & & & & 0 & & & \\
0 & & 1 & & 1 & & 0 & & 1 & & 0 & & 0 & & 1 & \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0
\end{array}
$$

$\ldots$

$0110\,1001\,1001\,0110\,1001\,0110\,0110\,1001\,1001\,0110\,0110\,1001 \cdots$

Diese Darstellung soll zwei Bildungsgesetze vermitteln: erstens: Bildung durch wiederholte Anwendung des durch $0 \to 01$ und $1 \to 10$ definierten Homomorphismus' sowie zweitens: die erste Zeile besteht nur aus der 0, die $i + 1$-te Zeile geht aus der i-ten hervor, indem man zunächst die i-te wieder hinschreibt und dahinter deren bitweises Komplement notiert. Die so entstehende unendliche Folge (genauer gesagt den Grenzwert dieser Folge) nennen wir die THUE-Folge (angedeutet in der letzten Zeile der 0-1-Zeilen).

Somit hat jede Sprache L, die aus einer unendlichen Menge von endlichen Teilwörtern der THUE-Folge besteht, die Eigenschaft, daß für kein $i > 2$ und kein Wort $z \in L$ eine Zerlegung $z = uvw$ existiert, so daß v^i ein Teilwort eines Wortes aus L ist.

Recht einfach lassen sich Grammatiken für solche Sprachen angeben, die wachsend kontextsensitiv sind. Daher gibt es für die wachsenden kontextsensitiven Sprachen kein Pumpinglemma.

Andrzej Ehrenfeucht, Rohit Parikh und Grzegorz Rozenberg beweisen, daß es überabzählbar viele Sprachen gibt, die das Pumpinglemma für REG erfüllen [EPR81]. Mit einer einfachen Konstruktion, die von den möglichen Sprachen, die nur aus Teilwörtern der THUE-Folge bestehen, ausgeht, kann man dieses Überabzählbarkeitsresultat recht einfach erhalten. Als ein Maß für die Kompliziertheit der Beweise mag die Anzahl der Alphabetsymbole dienen. In [EPR81] werden 16 verwendet — wir verwenden nur drei. Mit einem Alphabetsymbol charakterisiert das Pumpinglemma für REG ebenso wie das Pumpinglemma für CFL die regulären Sprachen. HENNING FERNAU hat mich auf die Arbeit [BH78] aufmerksam gemacht. Dort wird das Resultat der Überabzählbarkeit auch für Ogdens Lemma erwähnt — ohne expliziten Beweis.

Literatur

[BH78] BOASSON, L. und S. HORVÁTH: *On languages satisfying Ogden's lemma. Theoretical Informatics and Applications (RAIRO)*, 12(3):201–202, 1978.

[EPR81] EHRENFEUCHT, ANDRZEJ, ROHIT J. PARIKH, und GRZEGORZ ROZENBERG: *Pumping lemmas for regular sets.* SIAM Journal on Computing, 10(3):536–541, 1981.

[Jac83] JACOBS, KONRAD: *Einführung in die Kombinatorik.* Walter de Gruyter, Berlin/New York, 1983.

[Mor21] MORSE, MARSTON: *Recurrent Geodesics on a Surface of Negative Curvature.* Transactions of the American Mathematical Society, 22:84–100, 1921.

[Thu06] THUE, AXEL: *Über unendliche Zeichenreihen.* Videnskaps-Selskabets Skrifter (Math.-Natur. Klasse), Skrifter No. 7:22 pp, 1906.

Two-way spectra for regular languages

David Damanik

Fachbereich Mathematik
J.W.Goethe Universität
D–60054 Frankfurt

damanik@math.uni-frankfurt.de

Abstract

It is well known that allowing two-way motion of the input head in a finite automaton can produce exponential savings in the number of states required to recognize a regular language. This paper studies situations intermediate between forbidding two-way motion and allowing it. The two-way complexity of a $2DFA$ is measured by counting the number of left moves in the accepting computations, so that the decrease in the state space that occurs as the two-way complexity is increased in increments can be studied. In analogy to a work of Goldstine, Kintala and Wotschke this intuition is reflected in the two-way spectrum of a regular language. We prove upper and lower bounds for spectra and study their sharpness in order to show that these intermediate situations always lie between two extremes:

1) There are no savings unless the two-way complexity becomes unlimited.

2) Each increment of two-way complexity results in additional savings.

1 Introduction

The basic acceptance model for regular languages is the deterministic finite one-way automaton ($1DFA$). There are two natural ways of extending this model, allowing nondeterminism and two-way motion of the input head, respectively, both of them do not allow the acceptance of non-regular languages. On the other hand, the extentions ($1NFA$ and $2DFA$) can produce exponential savings in the number of states required to recognize a regular language. But in some cases not the whole power of the extention is needed, especially for some languages the minimal $1DFA$ is as small as the minimal $1NFA$ ($2DFA$ resp.). Thus, for a given regular language, one can ask how much nondeterminism (two-way motion resp.) is required to describe it adequately. Of course, one has to introduce well-motivated measures for nondeterminism (two-way motion resp.) first.

In their 1990 paper [3], Goldstine, Kintala and Wotschke handled the case of nondeterminism. The measure they used reflects the maximal (taken over the words in the language) number of branches in the minimal (concerning the number of branches) accepting computation. Starting with a $1DFA$ and allowing more and more nondeterminism according to this measure they studied the decrease of state complexity in the nondeterminism spectrum. Concerning two-way motion, we are going to follow similar lines.

By counting the number of left moves in the accepting computations in a $2DFA$ we measure the two-way complexity of a given automaton. We get the class $2DFA(k)$ by restricting $2DFA$'s according to this measure. The two-way spectrum of a regular language L is now given by a monotonically decreasing infinite sequence where the entries are the sizes of minimal $2DFA(k)$-descriptions of L. The rate of decrease reflects the change of size of a $2DFA$ describing L when the two-way complexity is increased incrementally.

The paper is organized as follows. Section 2 introduces the concepts. Section 3 gives some examples with different behaviour of the spectrum. Upper and lower bounds for spectra together with their sharpness are studied in sections 4 and 5.

Due to the analogy mentioned above, some ideas and techniques are inspired by [3]. Nevertheless, for the sake of self-containedness, full proofs of our theorems are provided.

2 Two-way complexity and restricted deterministic finite two-way automata

Definition 2.1 *A* **finite nondeterministic two-way automaton** *(2NFA) is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is the finite state set, Σ is a finite alphabet, $q_0 \in Q$ is the starting state and $F \subseteq Q$ is the set of accepting states. $\delta : Q \times \Sigma \to 2^{Q \times \{-1,+1\}}$ is the transition function. M is called* **deterministic** *(2DFA) if $|\delta(q,a)| \leq 1 \ \forall q \in Q, a \in \Sigma$. M is* **one-way** *(1NFA, 1DFA) if $\delta(q,a) \subseteq Q \times \{+1\} \ \forall q \in Q, a \in \Sigma$. The* **size** *$|M|$ is defined by $|M| = \#Q$.*

In the definition of the transition function for an one-way automaton we will omit the second component.

Definition 2.2 *A* **configuration** *is a pair $(q,j) \in Q \times \mathbb{N}_0$, a finite sequence of configurations is called* **computation**. *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a 2NFA and $w = a_0 \dots a_{n-1} \in \Sigma^n$ an input string. A computation $(p_0, j_0), \dots, (p_m, j_m)$ is called* **computation on** *w if*

- $p_0 = q_0$
- $j_0 = 0, \ j_m \leq n$
- $\forall i \in \{0, \dots, m-1\} \ : \ 0 \leq j_i < n$ *and* $\exists (p,k) \in \delta(p_i, a_{j_i}) \ : \ p_{i+1} = p, \ j_{i+1} = j_i + k$

A computation on w is **accepting** *if $j_m = n, \ p_m \in F$. Let*

$$T(M) = \{w \in \Sigma^* \mid \exists \text{ accepting computation on } w\}$$

be the **language accepted by** *M.*

Fact 2.3 (Shepherdson, Birget) *For every n-state $2DFA$ there is an equivalent n^n-state $1DFA$.*

The corresponding construction is due to Shepherdson ([5]) and yields the upper bound $n \cdot (n+1)^n$ for the conversion $2DFA \to 1DFA$. It has been improved by Birget ([1]) yielding the bound n^n. Asymptotic sharpness is due to Meyer and Fischer ([4]): introducing the languages

$$L_n = \{0^{i_1}10^{i_2}1 \dots 10^{i_n}2^m0^{i_m} \mid 1 \leq m \leq n,\ 1 \leq i_j \leq n\},$$

one verifies

Fact 2.4 (Meyer, Fischer) *For every n, there is a $5n+5$-state $2DFA$ accepting L_n, but every $1DFA$ accepting L_n has at least n^n states.*

Definition 2.5 *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a $2DFA$ and $w \in \Sigma^n$ an input string with the corresponding computation $(q_0, j_0), \dots, (q_m, j_m)$ on w. Let*

$$\begin{aligned} \zeta(w) &= \#\{i \mid 0 \leq i \leq m,\ j_i = -1\} \\ \zeta(M) &= \sup\{\zeta(w) \mid w \in T(M)\}. \end{aligned}$$

$\zeta(w)$ gives the number of left moves in the computation of M on input w. Hence, $\zeta(M)$ gives the maximal number of left moves occuring in an accepting computation and measures the two-way complexity of a $2DFA$ in an absolute way. It is natural to consider the following complexity classes:

Definition 2.6 *For $k \in \mathbb{N} \cup \{\infty\}$ let*

$$2DFA(k) = \{M \mid M\ 2DFA,\ \zeta(M) \leq k\}.$$

Obviously, the following observations hold:

- $k_1 \leq k_2 \Rightarrow 2DFA(k_1) \subseteq 2DFA(k_2)$
- $2DFA(\infty) = 2DFA$
- $2DFA(0) \approx 1DFA$

where in the last observation one has to ignore left moves. This does not change the accepted language and minimality is preserved.

For a given regular language L one can ask how much two-way motion (according to the measure introduced above) is required to describe it adequately, i.e. with a small number of states. The following definition is a step towards formalizing this problem.

Definition 2.7 *Let L be regular. We define the* **two-way spectrum** *of L by*

$$\sigma(L) = (\sigma_0(L), \sigma_1(L), \ldots; \sigma_\infty(L)),$$

where

$$\sigma_k(L) = \min\{|M| \mid M\ 2DFA,\ T(M) = L\}.$$

The sequence $(\sigma_k(L))$ is bounded and monotonically decreasing. Thus, it takes only finitely many values. $\sigma_0(L)$ $(\sigma_\infty(L))$ gives the size of the minimal $1DFA$ (the minimal size of $2DFA$'s for L). In general, one has $\sigma_0(L) > \sigma_\infty(L)$. The rate of decrease of the sequence describes the change of size of the minimal automaton as the two-way complexity is increased incrementally. Intuitively, a slow decrease for small k means that much two-way motion is required to get a concise representation of the language by means of a $2DFA$. Conversly, rapid decrease indicates that the language can be represented by a $2DFA$ consisely with a small amount of two-way motion.

3 Examples

3.1 The constant spectrum

Our goal in this subsection is to establish a spectrum of the form $(n, n, \ldots; n)$, $n \in \mathbb{N}$. A candidate is the sequence $L_n = \{1^{n-1}\} \subseteq 1^*$ and one easily verifies

- $\sigma_0(L_n) \leq n$
- $\sigma_\infty(L_n) \geq n$

and hence, $\sigma(L_n) = (n, n, \ldots; n)$. Of course, this example is in a way trivial since we deal with unary languages. On the other hand, $(L_n)_{n \in \mathbb{N}}$ also serves as a witness for a constant nondeterminism spectrum - another reason for the importance of this sequence, compare [2].

3.2 The collapse at $n \in \mathbb{N}$

We try to construct a spectrum which is (almost) constant for $\{k \mid k = 0, \ldots, n-1\}$ and for $\{k \mid k = n, n+1, \ldots, \infty\}$ but has a major gap at the transition from $n-1$ to n. To this end, consider the languages $L_n = \{0,1\}^* 1 \{0,1\}^{n-1} \$$, $n \in \mathbb{N}$. The subsequent four lemmas establish the desired form of the spectra.

Lemma 3.1 $\sigma_0(L_n) \leq 2^n + 1$

Proof Define the $1DFA$ $M = (Q, \Sigma, \delta, q_0, F)$ by

$$\begin{aligned} Q &= \{0,1\}^n \cup \{f\} \\ \Sigma &= \{0,1,\$\} \\ q_0 &= (0, \ldots, 0) \\ F &= \{f\} \end{aligned}$$

and

$$\delta((a_1,\ldots,a_n),a) = \begin{cases} (a_2,\ldots,a_n,a) & a \in \{0,1\} \\ f & if\ a = \$\ and\ a_1 = 1 \\ undefined & otherwise \end{cases}$$

Obviously, $T(M) = L_n$ and $|M| = 2^n + 1$. □

Lemma 3.2 $\sigma_{n-1}(L_n) \geq 2^n + 1$

Proof Let $M = (Q, \{0,1,\$\}, \delta, q_0, F) \in 2DFA(n-1)$ with $T(M) = L_n$. Intuitively, after reading $'\$'$ M cannot move back n symbols. Therefore, at any stadium M has to remember the last n symbols in the finite state control which requires 2^n states. We are going to follow this intuition. Since $\zeta(M) < \infty$,

$$\exists\, \hat{w} \mid \zeta(\hat{w}) = \max\{\zeta(v)\ :\ v \in \{0,1\}^*\}$$

Furthermore, every $w \in \{0,1\}^*$ is a prefix of some word in L_n. Thus, the computation of M on w reaches the right end of w in q_w. We are going to prove that $\{\hat{w}v \mid v \in \{0,1\}^n\}$ is a distinguishing set of words. Assume to the contrary

$$q_{\hat{w}v_1} = q_{\hat{w}v_2}\ for\ v_1, v_2 \in \{0,1\}^n,\ v_1 \neq v_2$$

Without loss of generality, we have $v_1 = x_1 1 y$, $v_2 = x_2 0 y$ with $|x_1| = |x_2|$. $\hat{w}v_1 0^{n-|y|-1}\$$ and $\hat{w}v_2 0^{n-|y|-1}\$$ are either both accepted or both rejected, a contradiction. Hence, $|Q| \geq 2^n$. Since $L_n \neq \emptyset$ and all the $q_{\hat{w}v}$ are non-accepting we have $|Q| \geq 2^n + 1$. □

Lemma 3.3 $\sigma_n(L_n) \leq n + 3$

Proof Define the $2DFA$ $M = (Q, \Sigma, \delta, q_0, F)$ by

$$\begin{aligned} Q &= \{q_0, \ldots, q_{n+1}, f\} \\ \Sigma &= \{0, 1, \$\} \\ F &= \{f\} \end{aligned}$$

and

$$\delta(q_i, a) = \begin{cases} (q_0, +1) & i = 0 \; and \; a \in \{0,1\} \\ (q_1, -1) & i = 0 \; and \; a = \$ \\ (q_{i+1}, -1) & 1 \leq i \leq n-1 \; and \; a \in \{0,1\} \\ (q_{n+1}, +1) & i = n \; and \; a = 1 \\ (q_{n+1}, +1) & i = n+1 \; and \; a \in \{0,1\} \\ f & i = n+1 \; and \; a = \$ \\ undefined & otherwise \end{cases}$$

Obviously, $M \in 2DFA(n)$, $T(M) = L_n$ and $|M| = n+3$. □

Lemma 3.4 $\sigma_\infty(L_n) \geq n+2$

Proof Arriving at $'\$'$, a $2DFA\ M$ has to move back n symbols, for otherwise $|M| \geq 2^n + 1$ as we saw in the proof of Lemma 3.2. Counting n arbitrary input symbols requires $n+1$ states in a $2DFA$ ([2]). Furthermore, M has an accepting state : $|M| \geq n+2$. □

3.3 The collapse at infinity

Consider the Meyer-Fischer sequence

$$L_n = \{0^{i_1} 1 0^{i_2} 1 \dots 1 0^{i_n} 2^m 0^{i_m} \,|1 \leq m \leq n, \; 1 \leq i_j \leq n\}$$

which serves as a witness for the (asymptotic) optimality of the Shepherdson-Construction. Intuitively, it is clear that for k fixed this sequence can be modified in order to prove asymptotic sharpness of the $2DEA \rightarrow 2DFA(k)$ conversion, simply by adding 1^k between 0^{i_n} and 2^m. On the other hand, when variing k in the spectrum we will get a collapse at finite n. However, in the next section we will prove that another modification works. Introducing $L'_n = (\$L_n\$)^*$, we get the desired collapse at infinity:

- $\sigma_k(L'_n) = \Theta(n^n) \; for \; k \in I\!N$
- $\sigma_\infty(L'_n) = \Theta(n)$

3.4 The stricly decreasing spectrum

Consider the sequence

$$L_n = \{ww^R \mid w \in \{0,1\}^n\}$$

We want to show that increasingly allowing two-way motion results in additional savings concerning state complexity starting with $\sigma_0(L_n) = \Omega(2^n)$ and reaching $\sigma_\infty(L_n) = \sigma_{\hat{k}}(L_n) = \mathcal{O}(n^2)$. First, we try to explain why two-way motion is better than one-way motion when acceptance of L_n is considered. Intuitively, a one-way automaton has to remember the word w which requires 2^n states even in a $1NFA$ as it is well known. A two-way automaton, however, can move back and forth and compare symbol for symbol. Thus, state space is required to count distances between symbols so that $\mathcal{O}(n^2)$ states are sufficient. Since L_n is finite, every $2DFA$ for L_n will have finitely bounded two-way complexity. Thus, $\sigma_\infty(L_n) = \sigma_{\hat{k}}(L_n)$ for some suitably chosen $\hat{k} \in I\!N$. Now, why is the spectrum decreasing? If $M \in 2DFA(k)$, the automaton can use its k left-moves in order to scan the middle part of the input checking if it is of the form vv^R. The remaining prefix and the remaining suffix can be compared in one-way fashion requiring state space exponential in their length. Increasing k then results in a decrease of this lenght. Thus, the size of M is a function of two terms, one representing the one-way part (prefix-suffix comparison) and being exponetial in the lenght of the prefix/suffix (which is of order $\mathcal{O}(n - \sqrt{k})$, and another one representing the two-way part (checking the middle of the input) which can of course be bounded by $\mathcal{O}(n^2)$. Further terms will all be bounded $\mathcal{O}(n^2)$, too. Hence, increasing two-way complexity yields a decrease of the prefix lenght which results in a decrease of the exponential part of the state space.

4 Upper bounds for spectra

The Shepherdson-bound for the $2DFA \rightarrow 1DFA$ conversion immediately yields the following upper bound for the spectrum of a regular language L.

$$\sigma(L) \leq (n^n, n^n, \ldots; n)$$

where L requires n states to be recognized by a $2DFA$. In this section, we show that this upper bound is nearly sharp. Of course, languages L_n with a spectrum of the form

$$\sigma(L_n) = (k_n, k_n, \ldots; n)$$

will be interesting, i.e. finitely bounded two-way motion is useless for L_n concerning state-complexity. The following lemma exhibits a whole class of such languages.

Lemma 4.1 *Let $L \subseteq \Sigma^*$ be regular and $\$ \notin \Sigma$. Then $k, n \in I\!N$ exist such that*

$$\sigma((\$L\$)^*) = (k, k, \ldots; n).$$

Proof Let $L' = (\$L\$)^*$. In the following, we will use without further comment

- $v, w \in L' \quad \Rightarrow \quad vw \in L'$
- $vw \in L',\ v \in L' \quad \Rightarrow \quad w \in L'$
- $vw \in L',\ w \in L' \quad \Rightarrow \quad v \in L'$

It suffices to show

$$\sigma_k(L') \geq \sigma_0(L') \quad \forall k \in I\!N$$

For an arbitrarily given $M = (Q, \Sigma \cup \{\$\}, \delta, q_0, F) \in 2DFA(k)$ with $T(M) = L'$ we will construct an equivalent $N = (Q, \Sigma \cup \{\$\}, \delta', q_0', F) \in 2DFA(0)$. The assertion then follows. Define

$$S = \delta(q_0, L')$$

and

$$T = \{q \in Q | \delta(q, L') \cap F \neq \emptyset\}.$$

Let

$$\delta'(p, a) = \begin{cases} (q, r) & if\ \delta(p, a) = (q, r)\ and\ r = +1 \\ undefined & otherwise \end{cases}$$

be the one-way part of δ. We will prove

Claim 1 $L' = \{w | \delta'(S, w) \cap T \neq \emptyset\}$

Claim 2 Let S_0 be a minimal subset of S such that Claim 1 remains true with S_0 in place of S. Then $L' = \{w | \delta'(s_0, w) \in T\}$ for all $s_0 \in S_0$.

$N = (Q, \Sigma \cup \{\$\}, \delta', q_0', F)$ with $q_0' = s_0 \in S_0$ arbitrarily chosen now accepts L' and uses only right moves, even in the non-accepting computations. Thus, $N \in 2DFA(0)$ as required.

Proof of Claim 1 $' \subseteq '$ Let $v \in L'$. Then $v^{k+1} \in L'$. Because of $\zeta(M) \leq k$ some factor v is input to a computation of M from S to T having only right moves. Hence, $v \in \{w | \delta'(S, w) \cap T \neq \emptyset\}$.
$' \supseteq '$ Let $v \in \{w | \delta'(S, w) \cap T \neq \emptyset\}$. Then $x, y \in L'$ exist with $xvy \in L'$. Hence, $v \in L'$.

Proof of Claim 2 It suffices to show

$$w \in L' \;\Rightarrow\; \delta'(s_0, w) \in T.$$

Suppose to the contrary that $\delta'(s_0, w) \notin T$ for some $w \in L'$. Then

$$\delta'(s_0, wL') \cap T = \emptyset. \tag{1}$$

Furthermore

$$\delta'(S_0, wv) \cap T \neq \emptyset \quad \forall v \in L'. \tag{2}$$

(1) and (2) imply

$$\delta'(S_0 \backslash \{s_0\}, wv) \cap T \neq \emptyset \quad \forall v \in L'$$

We have

$$L' = \{v | \delta'(S_1, v) \in T\}$$

where $S_1 = \delta'(S_0 \backslash \{s_0\}, w)$. This contradicts the minimality of S_0 and proves Claim 2. □

Lemma 4.2 *Let $L \in \Sigma^*$ be regular, $L' = (\$L\$)^*$ and $\sigma_0(L) \geq f(\sigma_\infty(L))$ with a monotonically increasing function f. Then*

$$\sigma_0(L') \geq f(\sigma_\infty(L') - 1) + 1.$$

Proof By adding one state, the corresponding minimal automata for L can be modified in order to accept L' without increasing the two-way complexity. Thus, $\sigma_0(L') \leq \sigma_0(L) + 1$ and $\sigma_\infty(L') \leq \sigma_\infty(L) + 1$. Below, we show

$$\sigma_0(L') \geq \sigma_0(L) + 1 \tag{3}$$

yielding

$$\sigma_0(L') = \sigma_0(L) + 1 \geq f(\sigma_\infty(L)) + 1 \geq f(\sigma_\infty(L') - 1) + 1.$$

Proof of (3): It is well known that the minimal number of states in a $1DFA$ for a regular language $R \in \Sigma^*$ is equal to the number of nonempty left quotients $x \backslash R$ of R, $x \in \Sigma^*$. For any w not containing \$ we have

$$(\$w)\backslash(\$L\$)^* = w\backslash L\$(\$L\$)^*.$$

Thus, L' has at least as many nonempty left quotients as L. Since $L' = \varepsilon\backslash L'$ is another one, it has at least one more. □

Lemmas 4.1 and 4.2 now imply

Theorem 4.3 *Let L_n be regular and f monotonically increasing with $\sigma_\infty(L_n) = n$ and $\sigma_0(L_n) \geq f(n)$. Then, there exists a sequence R_n of regular languages with $\sigma_\infty(R_n) = n$ and*

$$\sigma(R_n) \geq (f(n-1) + 1, f(n-1) + 1, \ldots; n).$$

Thus, the optimality of the upper bound for spectra has essentially been reduced to the optimality of the Shepherson-bound.

5 Lower bounds for spectra

For a regular language L with $\sigma_\infty(L) = n$

$$\sigma(L) \geq (n, n, \ldots; n)$$

is a trivial lower bound for the spectrum. However, it is sharp as the example in 3.1 shows. A lower bound

$$\sigma(L) \geq (m, \ldots; \ldots)$$

is more interesting, where $m = \sigma_0(L)$ and the entries are functions of m. To this end, we consider the conversion $2DFA(k) \to 1DFA$ because the corresponding blow-up is related to the decrease under study. For $M \in 2DFA(k)$ we can construct an equivalent $1DFA$ in two steps:

Step 1 Store the last k tape symbols into the finite state control and use ε-moves in order to simulate backtracking.

Step 2 Eliminate ε-moves.

Proposition 5.1 *For every n-state $2DFA(k)$ over the alphabet Σ there is an equivalent $1DFA$ with $n \cdot (k+1) \cdot (|\Sigma| + 1)^k$ states.*

Remarks a) The blow-up is linear in n for fixed k. This is not true for fixed finite nondeterminism, compare [3].
b) We have an exponential blow-up in k. Sharpness of this bound would imply a rapid decrease of the spectrum. However, this bound is not expected to be exact, as it is to be seen from the proof.
c) For $k \geq \mathcal{O}(n \cdot \log n)$, our construction has no advantage compared to the Shepherdson-Construction.

Proof We follow the above idea. Let $M = (Q, \Sigma, \delta, q_0, F) \in 2DFA(k)$.

Step 1 : Construction of an equivalent $1DFA$ $M' = (Q', \Sigma, \delta', q_0', F')$ with $|Q'| = |Q| \cdot (k+1) \cdot (|\Sigma| + 1)^k$

Let

$$
\begin{aligned}
Q' &= Q \times \{0, \ldots, k\} \times (\Sigma \cup \{\$\})^k \\
q_0' &= (q_0, 0, \$, \ldots, \$) \\
F' &= \{(f, 0, a_k, \ldots, a_1) \mid f \in F,\ a_i \in \Sigma \cup \{\$\}, \\
&\quad\ 1 \le i \le k\} \\
\delta'((p, 0, a_k, \ldots, a_1), a) &= \begin{cases} (q, 0, a_{k-1}, \ldots, a_1, a) \\ \quad if\ \delta(p, a) = (q, +1) \\ (q, 1, a_k, \ldots, a_1) \\ \quad if\ \delta(p, a) = (q, -1) \end{cases} \\
\delta'((p, j, a_k, \ldots, a_1), \varepsilon) &= \begin{cases} (q, j-1, a_k, \ldots, a_1) \\ \quad if\ \delta(p, a_j) = (q, +1),\ j > 0 \\ (q, j+1, a_k, \ldots, a_1) \\ \quad if\ \delta(p, a_j) = (q, -1),\ j > 0 \end{cases}
\end{aligned}
$$

where $\$ \notin \Sigma$ and transitions not listed are undefined. Obviously, $T(M') = T(M)$ and M' is deterministic.

Step 2 : Elimination of ε-moves

We construct the $1DFA$ $M'' = (Q', \Sigma, \delta'', q_0', F'')$ where

$$
\begin{aligned}
\varepsilon(q) &= \{q' \mid \delta'(q, \varepsilon^*) = q'\} \\
\delta''(q, a) &= \begin{cases} \delta'(q, a) & if\ \delta'(q, a) \neq \emptyset \\ \delta'(\varepsilon(q), a) & if\ \varepsilon(q) \neq \emptyset \end{cases} \\
F'' &= F' \cup \{q_0' \mid \varepsilon(q_0') \cap F' \neq \emptyset\}
\end{aligned}
$$

for $q \in Q'$ in order to eliminate ε-moves. δ'' is well-defined (determinism of M') and M'' is a $1DFA$ with $|Q| \cdot (k+1) \cdot (|\Sigma| + 1)^k$ states and $T(M'') = T(M)$. $\square$

Corollary 5.2 *For any n-state $1DFA$ there is no equivalent $2DFA(k)$ with less than $n \cdot (k+1)^{-1} \cdot (|\Sigma| + 1)^{-k}$ states.*

Now, we have the following lower bound for $\sigma(L)$:

Theorem 5.3 *Let L be regular with $\sigma_0(L) \geq n^n$. Then*

$$\sigma(L) \geq (n^n, \frac{1}{2}c \cdot n^n, \frac{1}{3}c^2 \cdot n^n, \frac{1}{4}c^3 \cdot n^n, \ldots, n, n, \ldots; n)$$

where $c = (|\Sigma| + 1)^{-1}$.

Now, we return to the construction of Proposition 5.1. $2DFA$'s with a weaker restriction can be handled, namely those only having access to the last k input symbols. Therefore, one cannot expect the construction to be optimal. Nevertheless, we give the following example which shows that asymptotically one has the following behaviour in the worst case: an exponential dependence on k and a linear dependence on n. Consider the languages

$$L_{n,k} = \{0,1\}^* 1 \{0,1\}^{k-1} \$_1 \cup \{w \in \{0,1\}^* \mid \#_1(w) = n-1\} \$_2.$$

It is easy to construct an $n+k+2$-state $2DFA(k)$ for $L_{n,k}$. Furthermore, every $1DFA$ for $L_{n,k}$ has at least $\Omega(n \cdot 2^k)$ states: it can be shown that the $1DFA$ obtained by cartesian product construction is minimal, where one has to take care of some 'impossible' states yielding a multiplicative constant smaller than 1. Unfortunately, this example does not give a good optimality result concerning Theorem 5.3 because $L_{n,k}$ is interesting when two-way complexity is constant and equal to k. For a fixed $L_{n,k}$, however, we will get an analog of example 3.2 when two-way complexity is varied, so for a sharpness result we have to refer to example 3.4.

References

[1] J.C.Birget : *State-Complexity of Finite-State Devices, State-Compressibility and Incompressibility*, Math. Systems Theory 26, 1993, 237-269

[2] J.C.Birget : *Two-Way Automata and Length-Preserving Homomorphisms*, to appear in Math. Systems Theory

[3] J.Goldstine, C.Kintala, D.Wotschke : *On Measuring Nondeterminism in Regular Languages*, Information and Computation 86, 1990, 179-194

[4] A.R.Meyer, M.J.Fischer : *Economy of description by automata, grammars and formal systems*, Proc. 12th IEEE Symp. on Switching and Automata Theory, 1971, 188-191

[5] J.C.Shepherdson : *The reduction of two-way automata to one-way automata*, IBM J. Res. Develop. 3, 1959, 198-200

Erkennbare, aperiodische und logisch definierbare Sprachen in Nebenläufigkeitsmonoiden

Manfred Droste

Institut für Algebra
Technische Universität Dresden
Mommsenstr. 13, 01062 Dresden

droste@math.tu-dresden.de

Ein klassisches Resultat von Büchi (1960) besagt, daß im freien Monoid aller Wörter eine Sprache genau dann erkennbar ist, wenn sie durch einen Satz der monadischen Logik zweiter Stufe definierbar ist. Nach McNaughton und Papert (1971) fallen die sternfreien Sprachen gerade mit den in der Logik erster Stufe definierbaren Sprachen zusammen. Diese Resultate wurden kürzlich von Thomas (1990) und Ebinger und Muscholl (1993) auf Sprachen in Spur-Monoiden verallgemeinert. Wir geben eine weitere Verallgemeinerung dieser Resultate auf Sprachen in sogenannten Nebenläufigkeitsmonoiden an.

Spuren bilden ein mathematisches Modell für ein paralleles System, in dem von der Reihenfolge zweier unabhängiger Aktionen abgesehen wird: die Unabhängigkeit wird hierbei durch eine binäre Relation auf dem zugrunde liegenden Alphabet der Aktionen beschrieben. Wir betrachten hier ein allgemeineres Modell, in dem die Unabhängigkeit zweier Aktionen auch von dem jeweiligen Zustand des zugrunde liegenden Systems abhängt. Ein *Automat mit Nebenläufigkeitsrelationen* ist ein Quadrupel $\mathcal{A} = (S, E, T, \|)$, in dem S die Menge der

Zustände oder Situationen, E die Menge der Aktionen (Ereignisse), $T \subseteq S \times E \times S$ die deterministische Transitionsrelation und $\| = (\|_s)_{s \in S}$ ein System binärer Unabhängigkeitsrelationen $\|_s \quad (s \in S)$ auf E ist. Zwei Berechnungsfolgen $(s,a,p)(p,b,r)$ und $(s,b,q)(q,a,r)$ werden für äquivalent erklärt, falls $a \|_s b$. Dies induziert auf der Menge aller Berechnungsfolgen $\mathrm{BF}(\mathcal{A})$ eine Kongruenz $\sim$, und der Quotient $\mathrm{M}(\mathcal{A}) = \mathrm{BF}(\mathcal{A})/\sim \cup \{0\}$, mit Konkatenation als Operation (und 0 als „Fehlerelement", um eine überall definierte Monoid-Operation zu erhalten) heißt *Nebenläufigkeitsmonoid* über $\mathcal{A}$. Falls $\mathcal{A}$ nur einen Zustand hat, erhalten wir so gerade die Spur-Monoide.

In vorhergehenden Arbeiten haben wir Zusammenhänge dieser Automaten mit der Theorie der Bereiche und mit Petri-Netzen mit Kapazitäten (place/transition nets) untersucht. Ferner erhielten wir eine Kleene-Charakterisierung der erkennbaren Sprachen in Monoiden $\mathrm{M}(\mathcal{A})$, falls $\mathcal{A}$ ein endlicher stabil-nebenläufiger Automat ist. In einer Arbeit mit F. Bracho und D. Kuske wurden Darstellungen der Berechnungen von $\mathcal{A}$ durch geeignete azyklische Graphen angegeben. Hier zeigen wir, wie sich durch Aussagen der Prädikatenlogik über derartigen Graphen Sprachen von Berechnungen spezifizieren lassen. Wie im Satz von Büchi stellen sich die erkennbaren Sprachen in $\mathrm{M}(\mathcal{A})$ genau als die in der monadischen Logik zweiter Stufe definierbaren Sprachen heraus. Unter geeigneten Voraussetzungen (jedoch nicht im allgemeinen) fallen auch die aperiodischen und die in der Logik erster Stufe definierbaren Sprachen zusammen. Dies zeigt Grenzen auf, inwieweit sich die zu Anfang erwähnten klassischen Resultate auf allgemeinere Monoide übertragen lassen. (Gemeinsame Arbeit mit D. Kuske, Dresden)

Literatur

[1] BRACHO, F., M. DROSTE and D. KUSKE: *Representation of computations in concurrent automata by dependence orders.* Internal report of the TU Dresden, MATH-AL-3-1994.

[2] BRACHO, F., M. DROSTE and D. KUSKE: *Dependence orders for computations of concurrent automata.* In: *STACS'95*, Lect. Notes in Comp. Science. vol. 900, Springer, 1995, 467-478.

[3] DROSTE, M.: *Concurrent automata and domains.* Intern. J. of Found. of Comp. Science **3** (1992), 389-418.

[4] DROSTE, M.: *A Kleene theorem for recognizable languages over concurrency monoids.* In: *21st ICALP*, Lect. Notes in Comp. Science vol. 820, Springer, 1994, 388-398.

[5] DROSTE, M. and D. KUSKE: *On logical definability of languages in concurrency monoids.* Presented at CSL'95.

[6] DROSTE, M. and D. KUSKE: *Automata with concurrency relations - a survey.* In: ASMICS- and MEP-Workshop on Algebraic and Syntactic Aspects of Concurrency, Chantilly, 1995.

[7] EBINGER, W. and A. MUSCHOLL: *Logical definability on infinite traces.* In: *20th ICALP*, Lect. Notes in Comp. Science vol. 700, Springer, 1993, 335-346.

[8] KUSKE, D.: *Nondeterministic automata with concurrency relations.* In: *Colloquium on Trees in Algebra and Programming*, Lect. Notes in Comp. Science vol. 787, Springer, 1994, 202-217.

[9] THOMAS, W.: *On logical definability of trace languages.* In: *Proc. of the workshop Algebraic Methods in Computer Science*, Kochel am See, FRG (1989) (V. Diekert, eds.), Report TUM-I9002, TU Munich, 1990, 172-182.

Programmierte Grammatiken mit unbedingtem Übergang und verwandte Mechanismen

Henning Fernau
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Sand 13, D–72076 Tübingen
fernau@informatik.uni-tuebingen.de

Da seit der Anfertigung der ersten Zusammenfassung doch einige neue Ergebnisse erzielt worden sind, möchte ich die Gelegenheit nutzen, diese im Rahmen dieser Vortragszusammenfassungen vorzustellen, zumal ich schon einiges davon mündlich in Rauischolzhausen vorgetragen hatte. Aus Zeitgründen ist das folgende Material in englischer Sprache. Beweise werden grundsätzlich weggelassen. Eine ausführliche Version dieser Arbeit ist von mir erhältlich.

1 Introduction

One of the main problems investigated in the theory of formal languages is the question of the relation between language classes. We will concentrate on this question in the present paper in connection with context-free regulated and limited forms of context-free parallel rewriting. Between these fields, there is a close connection, first observed by Dassow [1], cf. also [3] and [7, Theorem 3.1].

Surprisingly enough, although grammars with unconditional transfer were already introduced in the very first papers on regulated rewriting, lots of problems remained unsolved. In this paper, we (at least partially) solve open problems stated in [1, 3, 4, 8, 13, 16, 18].

Conventions: $\subseteq$ denotes inclusion, $\subset$ denotes strict inclusion, $|M|$ is the number of elements in the set M. The empty word is denoted by λ. The length of a word x is denoted by $|x|$. If $x \in V^*$, where V is some alphabet, and if $W \subseteq V$, then $|x|_W$ denotes the number of occurrences of letters from W in x. $\partial_a L$ denotes the left derivative of $L \subseteq V^*$, i.e., $\partial_a L = \{w \in V^* \mid aw \in L\}$.

Sometimes, we use notations like $\mathcal{L}$(1lE[P]T0L) = $\mathcal{L}$(P,CF[$-\lambda$],ut) in order to say that the equation holds both in the case of excluding erasing productions (as indicated by the P and $-\lambda$ enclosed in brackets) and in the case of admitting erasing productions (neglecting the bracket contents). We consider two languages L_1, L_2 to be equal iff $L_1 \setminus \{\lambda\} = L_2 \setminus \{\lambda\}$, and we simply write $L_1 = L_2$ in this case. We term two devices describing languages equivalent if the two described languages are equal. We have a similar interpretation of the inclusion relation of languages.

We presuppose some knowledge of formal language theory on side of the reader. Especially, the Chomsky hierarchy $\mathcal{L}(\mathrm{CF}) \subset \mathcal{L}(\mathrm{CS}) \subset \mathcal{L}(\mathrm{REC}) \subset \mathcal{L}(\mathrm{RE})$ should be known.

2 Regulated rewriting

A *grammar controlled by a bicoloured digraph* [2, 11, 13, 19] or *G-grammar* is an 8-tuple $G = (V_N, V_T, P, S, \Gamma, \Sigma, \Phi, h)$ where

- V_N, V_T, P, S define, as in a phrase structure grammar, the set of nonterminals, terminals, context-free core rules, and the start symbol, respectively;
- Γ is a bicoloured digraph, i.e., $\Gamma = (U, E)$, where U is a finite set of nodes and $E \subseteq U \times \{g, r\} \times U$ is a finite set of directed edges coloured by g or r ("green" or "red");

- $\Sigma \subseteq U$ are the initial nodes;
- $\Phi \subseteq U$ are the final nodes;
- $h : U \to (2^P \setminus \{\emptyset\})$ relates nodes with rule sets.

There are two different definitions of the appearance checking mode in the literature: We say that $(x, u) \Rightarrow (y, v)$ [$(x, u) \Rightarrow_c (y, v)$, respectively] holds in G with $(x, u), (y, v) \in (V_N \cup V_T)^* \times U$, if either $x = x_1 w x_2$, $y = x_1 z x_2$, $w \to z \in h(u)$, $(u, g, v) \in E$ or every [one, respectively] rule of $h(u)$ is not applicable to x, $y = x$, $(u, r, v) \in E$. The reflexive transitive closure of $\Rightarrow$ [$\Rightarrow_c$, respectively] is denoted by $\stackrel{*}{\Rightarrow}$ [$\stackrel{*}{\Rightarrow}_c$, respectively]. The corresponding languages generated by G are defined by $L(G_{[c]}) = \{x \in V_T^* \mid \exists u \in \Sigma \exists v \in \Phi (S, u) \stackrel{*}{\Rightarrow}_{[c]} (x, v)\}$.

G is said to be with *unconditional transfer* [13] (there, the notion of "full checking" is used instead) iff $(u, g, v) \in E \iff (u, r, v) \in E$. If $E \cap U \times \{r\} \times U = \emptyset$, G is *without appearance check*.

The corresponding language families are denoted (this time deviating from [2]) by $\mathcal{L}(\mathrm{G}_{[c]}, \mathrm{CF,ac})$, $\mathcal{L}(\mathrm{G}_{[c]}, \mathrm{CF,ut})$, and $\mathcal{L}(\mathrm{G}_{[c]}, \mathrm{CF})$. Here and in the following, we write CF$-\lambda$ instead of CF if we do not allow erasing core rules of the form $A \to \lambda$.

By definition, we have the following relations:

Lemma 1 *Let $X \in \{CF, CF-\lambda\}$. Then, we have:*

- $\mathcal{L}(G_{[c]}, X), \mathcal{L}(G_{[c]}, X, ut) \subseteq \mathcal{L}(G_{[c]}, X, ac)$ *and*
- $\mathcal{L}(G_c, X) = \mathcal{L}(G, X)$.

We consider three special cases of G-grammars in the following.

- A *programmed grammar* has the following features:
  - ◦ Every node contains exactly one rule. Therefore, both modes of appearance checks coincide in this case.
  - ◦ There are no designated initial or final nodes. It is possible to start a derivation in each node containing a rule whose left-hand side equals to the start symbol S, and it is possible to stop anywhere when a terminal string has been derived.

Usual notation of rules: $(r : A \to w, \sigma, \phi)$, where r — called label — is the name of the particular node, and σ (success field) [or ϕ (failure field), respectively] is the set of nodes green arcs [or red arcs, respectively], starting from r, are pointing to.
As language families, we obtain, e.g., $\mathcal{L}(\mathrm{P}, \mathrm{CF,ac})$, $\mathcal{L}(\mathrm{P}, \mathrm{CF,ut})$, and $\mathcal{L}(\mathrm{P}, \mathrm{CF})$.

- A *matrix grammar* has the following features:
  - Every node contains exactly one rule. Therefore, both modes of appearance checks coincide in this case.
  - If there is a red arc from node n to node n', then there is also a green arc from node n to node n'. Only the initial nodes (not necessarily containing rules with left-hand side S) are allowed to have more than one ingoing green arc, while only the final nodes are allowed to have more than one outgoing green arc. More precisely, between every final node and every initial node, there is a green arc.

  As language families, we obtain, e.g., $\mathcal{L}(\mathrm{M}, \mathrm{CF,ac})$, $\mathcal{L}(\mathrm{M}, \mathrm{CF,ut})$, and $\mathcal{L}(\mathrm{M}, \mathrm{CF})$.
- A *(periodically) time-variant grammar*
  - If there is a red arc from node n to node n', then there is also a green arc from node n to node n'. Every node has exactly one ingoing green arc and one outgoing green arc. In other words, the graph of green arcs has a simple ring structure.
  - There is one designated initial node, and every node can be a final node.

  Usual notation: there is a periodic function $\phi : \mathbf{N} \to 2^P \setminus \{\emptyset\}$ prescribing the rule set $\phi(n)$ of the n^{th} derivation step.
  Now, it makes sense to consider both the $\Rightarrow$ and the $\Rightarrow_c$ derivations, leading to the families of languages $\mathcal{L}(\mathrm{TV}_{[c]}, \mathrm{CF,ac})$, $\mathcal{L}(\mathrm{TV}_{[c]}, \mathrm{CF,ut})$, and $\mathcal{L}(\mathrm{TV}_{[c]}, \mathrm{CF})$. For reasons unknown to us, only the c-mode has been considered in the literature.

Another way of viewing at such regulations of the derivation is the consideration of control lanuages. Without introducing them formally, we mention the class of regularly controlled grammars leading to the families of languages $\mathcal{L}(\mathrm{rC}, \mathrm{CF,ac})$, $\mathcal{L}(\mathrm{rC}, \mathrm{CF,ut})$, and $\mathcal{L}(\mathrm{rC}, \mathrm{CF})$.

From the monograph [2], the following relations are known:

Theorem 2 *Let* $X \in \{G_c, P, M, rC, TV_c\}$, $Y \in \{CF, CF - \lambda\}$, $Z \in \{ac, \lambda\}$. *Then* $\mathcal{L}(X, Y, Z) = \mathcal{L}(G, Y, Z)$.

From [2, 5, 6, 9, 10], we know the following:

Theorem 3 • $\mathcal{L}(G, CF) \subset \mathcal{L}(G, CF, ac) = \mathcal{L}(RE)$.
• $\mathcal{L}(G, CF - \lambda) \subset \mathcal{L}(G, CF - \lambda, ac) \subset \mathcal{L}(CS)$.

What about unconditional transfer? Although already Rosenkrantz treated this case in his famous starting paper [12], very little was known in the literature.

Theorem 4 *Let* L *be some recursively enumerable language over the alphabet* V_T. *Then, there is a (P,CF,ut) grammar* $\tilde{G}$ *such that* $w \in L$ *iff* $w\$\# \in L(\tilde{G})$, *where* $\{\$, \#\} \cap V_T = \emptyset$.

Corollary 5 $\mathcal{L}(P,CF,ut)$ *contains non-recursive languages.*

Corollary 6 $\mathcal{L}(P,CF{-}\lambda,ut) \subset \mathcal{L}(P,CF,ut)$

We say that a language class $\mathcal{L}$ is closed under *endmarking* iff $L \in \mathcal{L}$, $L \subseteq V^*$ and $\# \notin V$ implies $L\{\#\} \in \mathcal{L}$. Using standard constructions, one obtains the following closure properties.

Lemma 7 $\mathcal{L}(P,CF,ut)$ *is closed under homomorphism, union and endmarking;*
$\mathcal{L}(P,CF{-}\lambda,ut)$ *is closed under non-erasing homomorphism (but not closed under arbitrary erasing homomorphisms), union and endmarking.*

Theorem 8 *The following statements are equivalent:*

• $\mathcal{L}(P,CF,ut) = \mathcal{L}(RE)$;

- *$\mathcal{L}$(P,CF,ut) is closed under intersection with regular sets;*
- *$\mathcal{L}$(P,CF,ut) is closed under intersection;*
- *$\mathcal{L}$(P,CF,ut) is closed under left or right derivatives;*
- *$\mathcal{L}$(P,CF,ut) is closed under general sequential machine (gsm) mappings;*
- *$\mathcal{L}$(P,CF,ut) is closed under inverse homomorphisms.*

Note that none of the closure properties required in the above theorem can be effective, since for (P,CF,ut) grammars, the emptiness problem is decidable, while it is undecidable for (P,CF,ac) grammars [12].

We encounter a similar situation in the λ-free case. We omit proofs here.

Theorem 9 *Let L be some (P,CF$-\lambda$,ac) language over the alphabet V_T. Then, there is a constant $m > 0$ and a (P,CF$-\lambda$,ut) grammar $\tilde{G}$ such that $w \in L$ iff $w\$\#^m \in L(\tilde{G})$, where $\{\$, \#\} \cap V_T = \emptyset$.*

Theorem 10 *The following statements are equivalent:*

- *$\mathcal{L}(P,CF-\lambda,ut) = \mathcal{L}(P,CF-\lambda,ac)$;*
- *$\mathcal{L}$(P,CF$-\lambda$,ut) is closed under intersection with regular sets and restricted homomorphisms;*
- *$\mathcal{L}$(P,CF$-\lambda$,ut) is closed under left or right derivatives;*
- *$\mathcal{L}$(P,CF$-\lambda$,ut) is closed under λ-free gsm mappings;*
- *$\mathcal{L}$(P,CF$-\lambda$,ut) is closed under inverse homomorphisms and restricted homomorphisms.*

Corollary 11 *If for some language family $\mathcal{L}$ we know that*

$$\mathcal{L}(P,CF[-\lambda],ut) \subseteq \mathcal{L} \subseteq \mathcal{L}(P,CF[-\lambda],ac),$$

and $\mathcal{L}$ is closed under intersection with regular sets [and restricted homomorphisms], then $\mathcal{L} = \mathcal{L}(P,CF[-\lambda],ac)$.

What happens in case of unconditional transfer in connection with other regulation mechanisms? We can show the following equalities, some of them were already known in the literature. Note that the case of TV$_c$-grammars is somewhat tricky, but we had to omit the proof for reasons of space.

Theorem 12 $\mathcal{L}(P,CF[-\lambda],ut) = \mathcal{L}(M,CF[-\lambda],ut) =$
$\mathcal{L}(rC,CF[-\lambda],ut) = \mathcal{L}(TV_c,CF[-\lambda],ut) = \mathcal{L}(G_c,CF[-\lambda],ut)$

Can there be something more interesting within the case of unconditional transfer? This is the case, as it is shown in the next theorem which solves an open problem raised in [13], where the corresponding statement in the case of left derivations is shown.

Theorem 13 *For* $X \in \{CF, CF{-}\lambda\}$, $\mathcal{L}(G,X,ut) = \mathcal{L}(G,X,ac)$ *holds.*

Theorem 14 • $\mathcal{L}(TV,CF,ut) = \mathcal{L}(G,CF,ut) = \mathcal{L}(RE)$
• $\mathcal{L}(TV,CF{-}\lambda,ut) = \mathcal{L}(G,CF{-}\lambda,ut) \subset \mathcal{L}(CS)$

3 Parallel rewriting

A *partition-k-limited ET0L system* is a sixtuple

$$G = (V, V', \{P_1, \ldots, P_r\}, \omega, \pi, k)$$

where V' is a non-empty subset (terminal alphabet) of the alphabet V, $\omega \in V^+$, and each so-called table P_i is a finite subset of $V \times V^*$ which satisfies the condition that, for each $a \in V$, there is a word $w_a \in V^*$ such that $a \to w_a \in P_i$, such that each P_i defines a finite substitution $\sigma_i : V^* \to 2^{V^*}$. π is a partition of V, i.e., $\pi = \{\pi_1, \ldots, \pi_s\}$ with $\bigcup_{i=1}^{s} \pi_i = V$ and $1 \le i < j \le s$ implies $\pi_i \cap \pi_j = \emptyset$, and k is some natural number. G is called propagating if no table contains an erasing production $a \to \lambda$. Similarly, the notion of deterministic system is inherited from the theory of Lindenmayer systems, i.e., G is called deterministic if no table contains two different productions with the same left-hand side. According to G, $x \Rightarrow y$ (for $x, y \in V^*$) iff there is a table P_i and $x = x_0\alpha_1 x_1 \cdots \alpha_n x_n$, $y = x_0\beta_1 x_1 \cdots \beta_n x_n$ such that $\alpha_\nu \to \beta_\nu \in P_i$ for each $1 \leq \nu \leq n$, such that, for each part π_j of the partition of π, either (1) $|\alpha_1 \cdots \alpha_n|_{\pi_j} = k$ or (2) $|\alpha_1 \cdots \alpha_n|_{\pi_j} < k$ and $|x_0 x_1 \cdots x_n|_{\pi_j} = 0$.

As usual, the language generated by G is given by $L(G) = \{w \in V'^* \mid \omega \stackrel{*}{\Rightarrow} w\}$, where $\stackrel{*}{\Rightarrow}$ denotes the reflexive transitive closure of $\Rightarrow$.

As special cases, we define *uniformly k-limited ET0L systems* (introduced by Wätjen and Unruh in [17, 18]) restricting ourselves to partitions of the form $\pi = \{V\}$, and *k-limited ET0L systems* (introduced by Wätjen in [16]) via the restriction to partitions of the form $\pi = \{\{a\} \mid a \in V\}$. This way, we obtain the language families $\mathcal{L}(kl_p\text{E[P][D]T0L})$, $\mathcal{L}(k\text{ulE[P][D]T0L})$, and $\mathcal{L}(k\text{lE[P][D]T0L})$.

In [4], we showed that 1lET0L languages can be non-recursive. By a similar technique, it is possible to show that $\mathcal{L}(k\text{lET0L}) \not\subseteq \mathcal{L}(\text{REC})$ for any $k \geq 1$, which solves the mentioned problem.

We mention the following relations which solve open problems from [1] (Is the inclusion $\mathcal{L}(1\text{lEPT0L}) \subseteq \mathcal{L}(\text{P,CF} - \lambda, \text{ut})$ strict or not? Confer [3] for a similar question on so-called periodically funtion-limited EPT0L systems which is solved in passing.), [8] (Gärtner left open the propagating and deterministic case.), [16] (Wätjen asked the relation between $\mathcal{L}(k\text{lEPT0L})$ and $\mathcal{L}(k'\text{lEPT0L})$. Our theorem solves this question partially for $k' = 1$.), and [18] (The relation between $\mathcal{L}(k\text{ulE[P]T0L})$ and $\mathcal{L}(k\text{lE[P]T0L})$ was asked.).

Theorem 15 *For every $k \geq 1$, we have:*

(1) $\mathcal{L}(kulE[P]T0L) \subseteq \mathcal{L}(P,CF[-\lambda]) \subset \mathcal{L}(P,CF[-\lambda], ac)$,

(2) $\mathcal{L}(klE[P]T0L) \subseteq \mathcal{L}(1lE[P]T0L) = \mathcal{L}(P,CF[-\lambda], ut)$,

(3) $\mathcal{L}(kl_pE[P]T0L) = \mathcal{L}(kl_pE[P]DT0L) = \mathcal{L}(1l_pE[P]DT0L) = \mathcal{L}(P,CF[-\lambda], ac)$,

(4) $\mathcal{L}(kulE[P]T0L) \neq \mathcal{L}(klE[P]T0L)$.

4 Summary and Open Problems

Let us mention the main open question in this area, namely the question whether the restriction of the appearance checking mechanism to unconditional transfer within programmed grammars entails a restricted class of languages or not. In the literature [15], such a separation is claimed, but we are not convinced the argument given there. This question seems to possess an invincibility similar to the question whether the appearance checking feature itself enhances the descriptional power or not, a problem only recently and independently solved in [10] (cf. also [5, 6]) and [9]. Interestingly, such separation result exists in case of leftmost derivations without erasing rules [15].

References

[1] J. Dassow. A remark on limited 0L systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 24(6):287–291, 1988.

[2] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science*. Berlin: Springer, 1989.

[3] H. Fernau. On function-limited Lindenmayer systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 27(1):21–53, 1991.

[4] H. Fernau. Membership for 1-limited ET0L languages is not decidable. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 30(4):191–211, 1994.

[5] H. Fernau. Observations on grammar and language families. Technical Report 22/94, Universität Karlsruhe, Fakultät für Informatik, August 1994. Most of this report will appear in Fundamenta Informaticae in 1996.

[6] H. Fernau. A predicate for separating language classes. *EATCS Bulletin*, 56:96–97, June 1995.

[7] H. Fernau and H. Bordihn. Remarks on accepting parallel systems. *International Journal of Computer Mathematics*, 56:51–67, 1995.

[8] S. Gärtner. *Partitions-limitierte Lindenmayer-Systeme*. Berichte aus der Informatik. Aachen: Shaker-Verlag, 1995. Zugleich: Dissertation Technische Universität Braunschweig.

[9] D. Hauschildt and M. Jantzen. Petri net algorithms in the theory of matrix grammars. *Acta Informatica*, 31:719–728, 1994.

[10] F. Hinz and J. Dassow. An undecidability result for regular languages and its application to regulated rewriting. *EATCS Bulletin*, 38:168–173, 1989.

[11] A. Pascu and Gh. Păun. On the planarity of bicolored digraph grammar systems. *Discrete Mathematics*, 25:195–197, 1979.

[12] D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the Association for Computing Machinery*, 16(1):107–131, 1969.

[13] G. Rozenberg and A. K. Salomaa. Context-free grammars with graph-controlled tables. Technical Report DAIMI PB-43, Institute of Mathematics at the University of Aarhus, Ny Munkegade, DAN–8000 Aarhus C, January 1975.

[14] A. K. Salomaa. *Formal Languages.* Academic Press, 1973.

[15] E. D. Stotskii. Control of the conclusion in formal grammars. *Problemy peredachi informacii; translated: Problems of information transmission*, 7(3):257–270, 1971 (Translation 1973).

[16] D. Wätjen. k-limited 0L systems and languages. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 24(6):267–285, 1988.

[17] D. Wätjen. On k-uniformly-limited T0L systems and languages. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 26(4):229–238, 1990.

[18] D. Wätjen and E. Unruh. On extended k-uniformly-limited T0L systems and languages. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 26(5/6):283–299, 1990.

[19] D. Wood. Bicolored digraph grammar systems. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 1:45–50, 1973.

Contextual Array Grammars

Rudolf Freund
Technical University Wien, Institute for Computer Languages
Resselgasse 3, A-1040 Wien
freund@csdec1.tuwien.ac.at

Gheorghe Păun*
Institute of Mathematics of the Romanian Academy of Sciences
Grivitei Str. 21, PO Box 1-764, RO-70700 Bucureşti

Grzegorz Rozenberg
University of Leiden, Department of Computer Science
Niels Bohrweg 1, NL-2333 CA Leiden

Abstract

We consider the (n-dimensional) array contenant of string contextual grammars in the sense of Marcus and investigate the power of such grammars, also with respect to the degree of connectedness, mainly in comparison with context-free array grammars. The main result is that every array language generated by a context-free array grammar is the coding of a language generated by a contextual array grammar.

1 Introduction

The contextual grammars were introduced in [9] for the string case, with motivations arising from descriptive linguistics. Basically, such

*Research partially carried out when the author was visiting the Technische Universität Wien, Austria.

a grammar consists of a finite set of strings (*axioms*) and a finite set of pairs (s, c), where s is a string and c is a *context*, i. e. a pair of strings, $c = (u, v)$, over the alphabet under consideration. Such a pair (s, c) is called *production,* and s is called its *selector*. Starting from an axiom, iteratively contexts are added as it is indicated by the productions, which yields new strings. There are two modes of adjoining contexts: at the ends of strings only (this case, named *external,* was considered in [9]; infinitely many productions (s, c), with finitely many contexts, are usually used, otherwise the obtained language is finite), or in the interior of strings, bracketting one occurence of a selector with the associated context (this case, named *interior*, was considered in [12]). The generated language consists of all strings obtained in this way, by finitely but arbitrarily many context adjoinings.

Several variants of these grammars were recently considered in [3], [4], [5], [13], [14]. One variant which we shall use also here concerns the way of defining the generated language: for a given grammar we can retain only the set of strings produced by blocked derivations, derivations which cannot be continued. This corresponds to the maximal mode of derivation (called t-mode) in cooperating grammar systems [1].

The fundamental difference between contextual grammars and Chomsky grammars or Lindenmayer systems is that in contextual grammars we do not *rewrite* symbols, but we only *adjoin* symbols to the current string; once introduced, a symbol cannnot be modified, it remains in the string to be finally generated.

In spite of this basic difference, there are surprising connections between languages generated by contextual grammars and Chomsky grammars or Lindenmayer systems. We quote some results of that kind, because they correspond in some sense with the main results of this paper or show significant differences between the string case and the array case:

1. Every regular language can be generated in the internal mode by a contextual grammar [3].
2. Every DOL language can be generated by a contextual grammar

with parallel internal derivations in the sense of [14]; this result is proved in [17].

3. Every linear language is the coding of the intersection of a regular set with a language generated in the external mode by a contextual grammar [5].

4. Every recursively enumerable language is the left quotient by a regular set of the intersection of a regular set with a language generated in the internal mode by a contextual grammar (with regular choice [3], or with one-sided contexts and context-free choice [4]).

For precise formulations and proofs of these results as well as related results, the reader is referred to the papers mentioned above.

The contextual style of generating strings can be naturally extended to arrays; in some sense, this case is even more natural than that of strings, because we have no longer to consider internal and external derivations, or pairs of adjoined strings, one to the left and one to the right of the selector: Basically, we start from a given finite set of arrays (axioms) and apply to them productions (s, c) where both s and c are finite patterns. If in the current array we can identify a subarray identical with s in such a way that the places corresponding to c are empty, then c is adjoined, producing a new array. In the generated language we may retain either all the arrays produced in this way or only the arrays to which no production is applicable (maximal derivations).

The main results of the paper concern the relations between the families of array languages generated in this way and array languages generated by context-free array grammars. It is proved that every array language generated by a context-free grammar is the coding of a language generated by a contextual array grammar in the maximal mode. This is somewhat surprising, because it looks much stronger than the result quoted above: we reach now the power of context-free grammars using a coding only (in [3], for equalizing the power of linear string grammars one uses both a coding and an intersection with a regular set); yet observe that here we use maximal derivations and not free derivations.

Some problems about contextual array grammars remain open, e. g. an interesting question concerns the influence of the number of used symbols on the generative power of contextual array grammars, when we consider only the shape of the produced arrays (and not the marking of the cells).

2 Definitions and examples

In the main part of this section we will introduce the definitions and notations for arrays and sequential array grammars ([7], [15], [18]) and give some explanatory examples, but first we recall some basic notions from the theory of formal languages (for more details the reader is referred to [16]).

Definition 2.1. For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation; the *empty string* is denoted by λ, and $V^* - \{\lambda\}$ is denoted by V^+. Any subset of V^+ is called a *λ-free (string) language.*

A *(string) grammar* is a quadruple

$$G = (V_N, V_T, P, S)\,,$$

where V_N and V_T are finite sets of *non-terminal* respectively *terminal symbols* with $V_N \cap V_T = \emptyset$, P is a finite set of *productions* $\alpha \to \beta$ with $\alpha \in V^+$ and $\beta \in V^*$, where $V = V_N \cup V_T$, and $S \in V_N$ is the *start symbol.* For $x, y \in V^*$ we say that *y is directly derivable from x in G*, denoted by $x \vdash_G y$, if and only if for some $\alpha \to \beta$ in P and $u, v \in V^*$ we get $x = u\alpha v$ and $y = u\beta v$. Denoting the reflexive and transitive closure of the derivation relation $\vdash_G$ by $\vdash_G^*$, the *(string) language generated by G* is

$$L(G) = \{w \in V_T^* \mid S \vdash_G^* w\}\,.$$

A production $\alpha \to \beta$ in P is called

- *monotonic* if the string β is at least as long as the string α,
- *context-free* if $\alpha \in V_N$.

The grammar G is called *monotonic* respectively *regular,* if every production in P is monotonic respectively context-free. The families of λ-free (string) languages generated by arbitrary, monotonic, respectively context-free grammars are denoted by $L(enum)$, $L(mon)$, respectively $L(cf)$. The following relations are known as the

CHOMSKY-hierarchy ([16]):

$$L(cf) \subsetneq L(mon) \subsetneq L(enum).$$

Definition 2.2. Let Z denote the set of integers, let N denote the set of positive integers, i.e. $N = \{1, 2, ...\}$, and let $n \in N$. Then an n-*dimensional array* $\mathcal{A}$ over an alphabet V is a function $\mathcal{A} : Z^n \to V \cup \{\#\}$, where

$$shape(\mathcal{A}) = \{v \in Z^n \mid \mathcal{A}(v) \neq \#\}$$

is finite and $\# \notin V$ is called the *background* or *blank symbol.* We usually shall write

$$\mathcal{A} = \{(v, \mathcal{A}(v)) \mid v \in shape(\mathcal{A})\}.$$

The set of all n-dimensional arrays over V shall be denoted by V^{*n}. The *empty array* in V^{*n} with empty shape shall be denoted by Λ_n. Moreover, we define $V^{+n} = V^{*n} - \{\Lambda_n\}$. Any subset of V^{+n} is called a Λ-free n-dimensional array language.

Definition 2.3. Let $v \in Z^n$. Then the *translation* $\tau_v : Z^n \to Z^n$ is defined by $\tau_v(w) = w + v$ for all $w \in Z^n$, and for any array $\mathcal{A} \in V^{*n}$ we define $\tau_v(\mathcal{A})$, the corresponding n-dimensional array translated by v, by

$$(\tau_v(\mathcal{A}))(w) = \mathcal{A}(w - v) \text{ for all } w \in Z^n.$$

The vector $(0, ..., 0) \in Z^n$ shall be denoted by Ω_n, the vector $(1, ..., 1)$ shall be denoted by E_n.

Usually ([2], [15], [18], and [19]) arrays are regarded as equivalence classes of arrays with respect to linear translations, i.e. only the relative positions of the symbols different from # in the plane are taken into account:

The equivalence class $[\mathcal{A}]$ of an array $\mathcal{A} \in V^{*n}$ is defined by

$$[\mathcal{A}] = \{\mathcal{B} \in V^{*n} \mid \mathcal{B} = \tau_v(\mathcal{A}) \text{ for some } v \in Z^n\}.$$

The set of all equivalence classes of n-dimensional arrays over V with respect to linear translations shall be denoted by $[V^{*n}]$ etc.

In order to be able to define the notion of connectedness of n-dimensional arrays, we need the following definitions:

Definition 2.4. An (undirected) *graph* g is an ordered pair (K, E), where K is a finite set of nodes and E is a set of undirected edges $\{x, y\}$ with $x, y \in K$. A sequence of different nodes $x_0, x_1, ..., x_m$, $m \in N$, is called a path of length m in g with the starting-point x_0 and the ending-point x_m, if for all i with $1 \leq i \leq m$ an edge $\{x_{i-1}, x_i\}$ in E exists. A graph g is said to be *connected*, if for any two nodes $x, y \in K$, $x \neq y$, a path in g with starting point x and ending point y exists. Observe that a graph $(\{x\}, \{\})$ with only one node and an empty set of edges is connected, too.

Let W be a non-empty finite subset of Z^n. For any $k \in N \cup \{0\}$, a graph $g_k(W) = (W, E_k)$ can be assigned to W such that E_k for $v, w \in W$ contains the edge $\{v, w\}$ if and only if $0 < \|v - w\| \leq k$, where the norm $\|u\|$ of a vector $u \in Z^n$, $u = (u(1), ..., u(n))$, is defined by

$$\|u\| = \max\{|u(i)| \mid 1 \leq i \leq n\}.$$

Then W is said to be *k-connected* if $g_k(W)$ is a connected graph. Observe that W is 0-connected if and only if $card(W) = 1$, where $card(W)$ denotes the number of elements in the set W.

Now let V be a finite alphabet and $\mathcal{A}$ an n-dimensional array over V, $\mathcal{A} \neq \Lambda_n$. Then $\mathcal{A}$ (respectively $[\mathcal{A}]$)is said to be *k-connected* if $g_k(shape(\mathcal{A}))$ is a connected graph. Obviously, if $\mathcal{A}$ is k-connected then $\mathcal{A}$ is m-connected for all $m > k$, too. The *norm of* $\mathcal{A}$ is the smallest number $k \in N \cup \{0\}$ such that $\mathcal{A}$ is k-connected, and is denoted by $\|\mathcal{A}\|$. Observe that $\|\mathcal{A}\| = 0$ if and only if $card\,(shape\,(\mathcal{A})) = 1$. For any $\mathcal{A} \in V^{+n}$ we define $\|[\mathcal{A}]\| = \|\mathcal{A}\|$.

Example 2.1. Obviously, the n-dimensional array

$$\mathcal{E}\,(n,k) = \{(\Omega_n, a)\,, (kE_n, a)\}$$

in $\{a\}^{*n}$ is m-connected only for every $m \geq k$, and therefore

$\|\mathcal{E}\,(n,k)\| = \|[\mathcal{E}\,(n,k)]\| = k.$ □

Definition 2.5. An *n-dimensional array production p* over V is a triple

$$(W, \mathcal{A}_1, \mathcal{A}_2)\,,$$

where $W \subseteq Z^n$ is a finite set and $\mathcal{A}_1$ and $\mathcal{A}_2$ are mappings from W to $V \cup \{\#\}$. We say that the array $\mathcal{C}_2 \in V^{*n}$ is *directly derivable* from the array $\mathcal{C}_1 \in V^{*n}$ by the n-dimensional array production $(W, \mathcal{A}_1, \mathcal{A}_2)$ if and only if there exists a vector $v \in Z^n$ such that $\mathcal{C}_1\,(w) = \mathcal{C}_2\,(w)$ for all $w \in Z^n - \tau_v\,(W)$ as well as $\mathcal{C}_1\,(w) = \mathcal{A}_1\,(\tau_{-v}\,(w))$ and $\mathcal{C}_2\,(w) = \mathcal{A}_2\,(\tau_{-v}\,(w))$ for all $w \in \tau_v\,(W)$, i.e. the subarray of $\mathcal{C}_1$ corresponding to $\mathcal{A}_1$ is replaced by $\mathcal{A}_2$, thus yielding $\mathcal{C}_2$; we also write $\mathcal{C}_1 \vdash_p \mathcal{C}_2$. Moreover we say that the array $\mathcal{B}_2 \in [V^{*n}]$ is *directly derivable* from the array $\mathcal{B}_1 \in [V^{*n}]$ by the n-dimensional array production $(W, \mathcal{A}_1, \mathcal{A}_2)$ if and only if there exist $\mathcal{C}_1 \in \mathcal{B}_1$ and $\mathcal{C}_2 \in \mathcal{B}_2$ such that $\mathcal{C}_1 \vdash_p \mathcal{C}_2$; we also write $\mathcal{B}_1 \vdash_p \mathcal{B}_2$.

In contrast to the representation of an array $\mathcal{A} \in V^{*n}$, which is uniquely described by

$$\{(v, \mathcal{A}(v)) \mid v \in shape\,(\mathcal{A})\}\,,$$

(i.e. by listing all positions v in Z^n occupied by a non-blank symbol $\mathcal{A}\,(v) \in V$ together with this symbol $\mathcal{A}\,(v)$), in an n-dimensional array production $(W, \mathcal{A}_1, \mathcal{A}_2)$ over V all positions in W together with their associated symbols must be listed for representing $\mathcal{A}_1$ respectively $\mathcal{A}_2$ by

$$\mathcal{A}_i = \{(v, \mathcal{A}_i(v)) \mid v \in W\}, \ 1 \leq i \leq 2.$$

Therefore the norm of the n-dimensional array production $(W, \mathcal{A}_1, \mathcal{A}_2)$ is defined to be the norm of W, i. e.

$$\|(W, \mathcal{A}_1, \mathcal{A}_2)\| = \|W\|.$$

On the other hand, also for the mappings $\mathcal{A}_1$ and $\mathcal{A}_2$ in the n-dimensional array production $(W, \mathcal{A}_1, \mathcal{A}_2)$ we can define their shapes by

$$\mathit{shape}(\mathcal{A}_i) = \{v \in Z^n \mid \mathcal{A}_i(v) \neq \#\}, \ 1 \leq i \leq 2.$$

As can already be seen from the definitions of an n-dimensional array production, the conditions for an application to an n-dimensional array $\mathcal{B}$ and the result of an application to $\mathcal{B}$, an n-dimensional array production

$$(W, \mathcal{A}_1, \mathcal{A}_2)$$

is a representative for the infinite set of equivalent n-dimensional array productions of the form

$$(\tau_v(W), \tau_v(\mathcal{A}_1), \tau_v(\mathcal{A}_2))$$

with $v \in Z^n$. Hence, without loss of generality, in the sequel we shall assume $\Omega_n \in W$. Moreover, we often will omit the set W, because it is uniquely reconstructible from the description of the two mappings $\mathcal{A}_1$ and $\mathcal{A}_2$ by

$$\mathcal{A}_i = \{(v, \mathcal{A}_i(v)) \mid v \in W\}, \ 1 \leq i \leq 2.$$

Thus in the sequel we will represent the n-dimensional array production $(W, \mathcal{A}_1, \mathcal{A}_2)$ also by writing $\mathcal{A}_1 \to \mathcal{A}_2$, i. e.

$$\{(v, \mathcal{A}_1(v)) \mid v \in W\} \to \{(v, \mathcal{A}_2(v)) \mid v \in W\}.$$

Definition 2.6. An *n-dimensional array grammar* is a quintuple

$$G = (n, V_N, V_T, \#, P, S),$$

where V_N is the alphabet of *non-terminal symbols*, V_T is the alphabet of *terminal symbols*, $V_N \cap V_T = \emptyset$, $\# \notin V_N \cup V_T$; P is a finite non-empty set of n-dimensional array productions over $V_N \cup V_T$ and S is the *start symbol*.

We say that the array $\mathcal{B}_2 \in [V^{*n}]$ is *directly derivable* from the array $\mathcal{B}_1 \in [V^{*n}]$ in G, denoted $\mathcal{B}_1 \vdash_G \mathcal{B}_2$, if and only if there exists an n-dimensional array production $p = (W, \mathcal{A}_1, \mathcal{A}_2)$ in P such that $\mathcal{B}_1 \vdash_p \mathcal{B}_2$. Let $\vdash_G^*$ be the reflexive transitive closure of $\vdash_G$. Then the *(n-dimensional) array language generated by G*, $[L(G)]$, is defined by

$$[L(G)] = \{\mathcal{B} \in [V_T^{*n}] \mid [\{(\Omega_n, S)\}] \vdash_G^* \mathcal{B}\}.$$

The norm of the n-dimensional array grammar G is defined to be the maximum of the norms of the n-dimensional array productions in P, i. e.

$$\|G\| = \max\{\|p\| \mid p \in P\}.$$

An n-dimensional array production $p = (W, \mathcal{A}_1, \mathcal{A}_2)$ in P is called

- *monotonic*, if $shape(\mathcal{A}_1) \subseteq shape(\mathcal{A}_2)$;
- *context-free*, if p is monotonic and moreover
  $card(shape(\mathcal{A}_1)) = 1$ and $\mathcal{A}_1(\Omega_n) \in V_N$ as well as
  $card(shape(\mathcal{A}_2)) = card(W)$;
  the condition $\mathcal{A}_1(\Omega_n) \in V_N$ allows the representation of a context-free array production as
  $$(A, \{(v, \mathcal{A}_2(v)) \mid v \in W\}) \text{ or } A \to \{(v, \mathcal{A}_2(v)) \mid v \in W\}$$
  instead of
  $$\begin{aligned}&(W, \{(\Omega_n, \mathcal{A})\} \cup \{(v, \#) \mid v \in W - \{\Omega_n\}\},\\ &\{(v, \mathcal{A}_2(v)) \mid v \in W\});\end{aligned}$$
  if $card(W) = 1$, we only write $A \to \mathcal{A}_2(\Omega_n)$.
  Observe that $card(shape(\mathcal{A}_2)) = card(W)$ implies $\mathcal{A}_2(v) \in V_N \cup V_T$ for all $v \in W$.

G is called (recursively) enumerable, monotonic, respectively context-free, if every array production in P is an arbitrary, monotonic, respectively context-free array production. The corresponding families of Λ-free n-dimensional array languages of equivalence classes of arrays shall be denoted by $[L(n, enum)]$, $[L(n, mon)]$, and $[L(n, cf)]$.

Remark 2.1. Let $G = (n, V_N, V_T, \#, P, S)$ be an n-dimensional array grammar. If G is monotonic, then $\|[\mathcal{A}]\| \leq \|G\|$ for all $[\mathcal{A}] \in [L(G)]$. In the case of arbitrary n-dimensional array grammars, additional restrictive conditions on the n-dimensional array productions in P are required in order to guarantee every n-dimensional array in $[L(G)]$ to be $\|G\|$-connected or even to be 1-connected as it is often required in the literature ([2], [15], [18], and [19]).

Like in the string case, the families of array languages defined above form a strict hierarchy (compare with the results stated in [7] and [8]).

Proposition 2.1. *(CHOMSKY-Hierarchy of n-dimensional array languages)*

$$[L(n, cf)] \subsetneq [L(n, mon)] \subsetneq [L(n, enum)] \text{ for all } n \in N.$$

With respect to the norms of monotonic respectively context-free array grammars, for each $k \geq 1$ we can distinguish different families of array languages:

Definition 2.7. For all $n, k \in N$ the family of n-dimensional array languages (of equivalence classes of arrays) that can be generated by some monotonic respectively context-free array grammar G with $\|G\| \leq k$ is denoted by $[L(n, mon, k)]$ (respectively $[L(n, cf, k)]$).

Remark 2.2. According to the preceding definition

$$\bigcup_{k \in N} [L(n, mon, k)] = [L(n, mon)] \text{ and } \bigcup_{k \in N} [L(n, cf, k)] = [L(n, cf)]$$

and therefore we will also use the notations $[L(n, mon, \infty)]$ respectively $[L(n, cf, \infty)]$ for $[L(n, mon)]$ respectively $[L(n, cf)]$.

Example 2.2. Consider the n-dimensional context-free array grammar

$$G(n,k) = (n, \{S\}, \{a\}, \#, \{S \to \mathcal{E}(n,k)\}, S)$$

where the $\mathcal{E}(n,k) = \{(\Omega_n, a), (kE_n, a)\}$ are the arrays already considered in Example 2.1 with $\|[\mathcal{E}(n,k)]\| = k$. As

$$[L(G(n,k))] = \{[\mathcal{E}(n,k)]\},$$

we obviously obtain

$$[L(G(n,k))] \in ([L(n, cf, k)] - [L(n, mon, k-1)])$$

for all $k \geq 2$.

The following results are obvious from the definitions and from the previous example:

Proposition 2.2. For all $n, k \in N$

a) $[L(n, mon, k)] \subsetneq [L(n, mon, k+1)]$ and
$[L(n, cf, k)] \subsetneq [L(n, cf, k+1)]$;

b) $[L(n, cf, k)] \subsetneq [L(n, mon, k)]$.

As many results for n-dimensional arrays for a special n can be taken over immediately for higher dimensions, we introduce the following notion:

Definition 2.8. Let $n, m \in N$ with $n \leq m$. For $n < m$, the natural embedding $i_{n,m} : Z^n \to Z^m$ is defined by $i_{n,m}(v) = (v, \Omega_{m-n})$ for all $v \in Z^n$; for $n = m$ we define $i_{n,n} : Z^n \to Z^n$ by $i_{n,n}(v) = v$ for all $v \in Z^n$.

To an n-dimensional array $\mathcal{A} \in V^{+n}$ with

$$\mathcal{A} = \{(v, \mathcal{A}(v)) \mid v \in shape(\mathcal{A})\}$$

we assign the m-dimensional array

$$i_{n,m}(\mathcal{A}) = \{(i_{n,m}(v), \mathcal{A}(v)) \mid v \in shape(\mathcal{A})\}.$$

3 Contextual array grammars: definitions and examples

First we introduce the concept of contextual grammars for arrays:

Definition 3.1. An *n-dimensional contextual array grammar* ($n \in N$) is a construct

$$G = (n, V, \#, P, A),$$

where V is an alphabet not containing the blank symbol $\#$, A is a finite set of axioms, i. e. of n-dimensional arrays in $[V^{+n}]$ and P is a finite set of rules of the form $(U_\alpha, \alpha, U_\beta, \beta)$, where

(i) $U_\alpha, U_\beta \subseteq Z^n$, $U_\alpha \cap U_\beta = \emptyset$, and U_α, U_β are finite and at least U_α is non-empty.

(ii) $\alpha : U_\alpha \to V \cup \{\#\}$, $\beta : U_\beta \to V$ such that $card(shape(\alpha)) \neq \emptyset$, where in the usual way we define

$$shape(\alpha) = \{w \in W \mid \alpha(w) \in V\}.$$

(U_α, α) corresponds with the *selector* and (U_β, β) with the *context* of the production $(U_\alpha, \alpha, U_\beta, \beta)$; U_α is called the *selector area*, and U_β is the *context area*. As the sets U_α and U_β are uniquely determined by α and β, we will also represent $(U_\alpha, \alpha, U_\beta, \beta)$ by (α, β) only.

For $\mathcal{C}_1, \mathcal{C}_2 \in V^{*n}$ we say that $\mathcal{C}_2$ is directly derivable from $\mathcal{C}_1$ by the contextual array production $p \in P$, $p = (U_\alpha, \alpha, U_\beta, \beta)$ (we write $\mathcal{C}_1 \vdash_p \mathcal{C}_2$), if there exists a vector $v \in Z^n$ such that

- $\mathcal{C}_1(w) = \mathcal{C}_2(w) = \alpha(\tau_{-v}(w))$ for all $w \in \tau_v(U_\alpha)$,
- $\mathcal{C}_1(w) = \#$ for all $w \in \tau_v(U_\alpha)$,
- $\mathcal{C}_2(w) = \beta(\tau_{-v}(w))$ for all $w \in \tau_v(U_\beta)$,
- $\mathcal{C}_1(w) = \mathcal{C}_2(w)$ for all $w \in Z^n - \tau_v(U_\alpha \cup U_\beta)$.

Hence, if in $\mathcal{C}_1$ we find a subpattern that corresponds with the selector α and only blank symbols at the places corresponding with β, we can add the context β thus obtaining $\mathcal{C}_2$.

For $\mathcal{B}_1, \mathcal{B}_2 \in [V^{*n}]$ we say that $\mathcal{B}_2$ is directly derivable from $\mathcal{B}_1$ by the contextual array production $p \in P$, $p = (U_\alpha, \alpha, U_\beta, \beta)$, denoted $\mathcal{B}_1 \vdash_p \mathcal{B}_2$, if and only if there exist $\mathcal{C}_1 \in \mathcal{B}_1$ and $\mathcal{C}_2 \in \mathcal{B}_2$ such that $\mathcal{C}_1 \vdash_p \mathcal{C}_2$. $\mathcal{B}_2$ is said to be directly derivable from $\mathcal{B}_1$in G, denoted $\mathcal{B}_1 \vdash_G \mathcal{B}_2$, if and only if $\mathcal{B}_1 \vdash_p \mathcal{B}_2$ for some $p \in P$.

For any rule $(U_\alpha, \alpha, U_\beta, \beta)$ we define

$$\|(U_\alpha, \alpha, U_\beta, \beta)\| = \|U_\alpha \cup U_\beta\|;$$

the norm of the set of contextual array productions in G, $\|P\|$, is defined by

$$\|P\| = \max\{\|U_\alpha \cup U_\beta\| \mid (U_\alpha, \alpha, U_\beta, \beta) \in P\}.$$

Moreover, we define the norm of the set of axioms, $\|A\|$, by

$$\|A\| = \max\{\|\mathcal{A}\| \mid \mathcal{A} \in A\}$$

as well as the norm of the contextual array grammar G, $\|G\|$, by

$$\|G\| = \max\{\|P\|, \|A\|\}.$$

By $\vdash_G^*$ we denote the reflexive transitive closure of $\vdash_G$ and by $\vdash_G^t$ we denote the relation which, for arbitrary arrays $\mathcal{A}, \mathcal{B} \in [V^{+n}]$, is defined by $\mathcal{A} \vdash_G^t \mathcal{B}$ if and only if $\mathcal{A} \vdash_G^* \mathcal{B}$ and there is no $\mathcal{C} \in [V^{+n}]$ such that $\mathcal{B} \vdash_G \mathcal{C}$.

In that way, two array languages can be associated with G :

$$\begin{aligned}[L_*(G)] &= \{\mathcal{B} \in [V^{+n}] \mid \mathcal{A} \vdash_G^* \mathcal{B} \text{ for some } \mathcal{A} \in A\}, \\ [L_t(G)] &= \{\mathcal{B} \in [V^{+n}] \mid \mathcal{A} \vdash_G^t \mathcal{B} \text{ for some } \mathcal{A} \in A\}.\end{aligned}$$

Definition 3.2. The family of n-dimensional array languages generated by contextual array grammars of the form $G = (n, V, \#, A, P)$ with $\|G\| \leq k$ in the mode $f \in \{*, t\}$ is denoted by $[L(n, cont, k, f)]$; when no bound on the degree of connectedness is considered, we replace k by ∞, thus obtaining $[L(n, cont, \infty, f)]$.

In order to elucidate these definitions above, let us consider some examples:

Example 3.1. Let L be a finite subset of $[V^{+n}]$. For the n-dimensional contextual array grammar

$$G_L = (n, V, \#, \emptyset, L)$$

we obviously obtain

$$[L_*(G_L)] = [L_t(G_L)] = L \text{ and } \|G_L\| = \|L\|.$$

As a special example, consider $L(n,k) = \{[E(n,k)]\}$, where

$E(n,k) = \{(\Omega_n, a), (kE_n, a)\}$ as in Example 2.2. Then

$$\left[L_*\left(G_{L(n,k)}\right)\right] = \left[L_t\left(G_{L(n,k)}\right)\right] = L(n,k)$$

and, as $\left\|G_{L(n,k)}\right\| = \|L(n,k)\| = k$, we immediately obtain

$$\begin{aligned}L(n,k) \in \quad & (([L(n, cont, k, *)] \cap [L(n, cont, k, t)]) - \\ & ([L(n, cont, k-1, *)] \cup [L(n, cont, k-1, t]))\end{aligned}$$

for all $k \geq 2$. □

Example 3.2. Let $G = (\{a\}, \#, P, A)$ with A containing the axiom

$$[\{((0,0), a), ((0,1), a), ((1,0), a)\}],$$

which in a more depictive way can be described by

$$\begin{array}{cc} a & \\ a & a \end{array}$$

and P consisting of four productions p_1, p_2, p_3, p_4 :

$$\begin{aligned}
p_1 &= (\{(0,0),(0,1),(1,0)\},\{((0,0),a),((0,1),a),((1,0),a)\}, \\
&\quad \{(1,1),(1,2),(2,1)\},\{((1,1),a),((1,2),a),((2,1),a)\}), \\
p_2 &= (\{(0,0),(0,1),(1,0)\},\{((0,0),a),((0,1),a),((1,0),a)\}, \\
&\quad \{(1,1)\},\{((1,1),a)\}), \\
p_3 &= (\{(0,0),(1,0),(1,1)\},\{((0,0),a),((1,0),a),((1,1),a)\}, \\
&\quad \{(0,1)\},\{((0,1),a)\}), \\
p_4 &= (\{(0,0),(0,1),(1,1)\},\{((0,0),a),((0,1),a),((1,1),a)\}, \\
&\quad \{(1,0)\},\{((1,0),a)\}).
\end{aligned}$$

As the selector area U_α and the context area U_β in a contextual array production of the form $(U_\alpha, \alpha, U_\beta, \beta)$ are disjoint, both α and β can be represented within only one pattern, i. e. p_1, p_2, p_3, p_4 in a more depictive way can be represented by the following patterns (the symbols of the selector are enclosed in boxes):

$$p_1 = \begin{array}{ccc} & a & \\ \boxed{a} & a & a \\ \boxed{a} & \boxed{a} & \end{array}, \quad p_2 = \begin{array}{cc} \boxed{a} & a \\ \boxed{a} & \boxed{a} \end{array},$$

$$p_3 = \begin{array}{cc} a & \boxed{a} \\ \boxed{a} & \boxed{a} \end{array}, \quad p_4 = \begin{array}{cc} \boxed{a} & \boxed{a} \\ \boxed{a} & a \end{array}.$$

Let us examine the maximal derivation in G : The rule p_1 cannot be used after using p_2, the rules p_3, p_4 can be used as long as the array contains North-West or South-East "angles". We start from an array without angles in the South-West direction; the use of all rules preserves this property. After using p_2 no angle exists in the North-East direction. In sum, what we obtain by maximal derivations must be a rectangle. As when using p_1 and p_2 we go on a diagonal of such a rectangle, we obtain in fact a square. In conclusion, $L_t(G)$ is the language of all solid squares of side length at least 2; adding the square $\{((0,0),a)\}$ to $\mathcal{A}$ we can also get the 1×1 square, i.e. we obtain the set of all solid squares (compare [19]). Let us consider three maximal derivations in G :

1. $$\begin{array}{cc} a & \\ a & a \end{array} \vdash_G \begin{array}{cc} a & a \\ a & a \end{array};$$

2. $$\begin{array}{cc} & \\ a & \\ a & a \end{array} \vdash_G \begin{array}{ccc} & a & \\ a & a & a \\ a & a & \end{array} \vdash_G \begin{array}{ccc} & a & a \\ a & a & a \\ a & a & \end{array} \vdash_G$$
$$\begin{array}{ccc} a & a & a \\ a & a & a \\ a & a & \end{array} \vdash_G \begin{array}{ccc} a & a & a \\ a & a & a \\ a & a & a \end{array};$$

3. $$\begin{array}{cc} & \\ & \\ a & \\ a & a \end{array} \vdash_G \begin{array}{ccc} & & \\ & a & \\ a & a & a \\ a & a & \end{array} \vdash_G \begin{array}{cccc} & & a & \\ & a & a & a \\ a & a & a & \\ a & a & & \end{array} \vdash_G$$
$$\begin{array}{cccc} & & a & a \\ & a & a & a \\ a & a & a & \\ a & a & & \end{array} \vdash_G \begin{array}{cccc} & & a & a \\ a & a & a & a \\ a & a & a & \\ a & a & & \end{array} \vdash_G$$
$$\begin{array}{cccc} & a & a & a \\ a & a & a & a \\ a & a & a & \\ a & a & & \end{array} \vdash_G \begin{array}{cccc} a & a & a & a \\ a & a & a & a \\ a & a & a & \\ a & a & & \end{array} \vdash_G$$
$$\begin{array}{cccc} a & a & a & a \\ a & a & a & a \\ a & a & a & a \\ a & a & & \end{array} \vdash_G \begin{array}{cccc} a & a & a & a \\ a & a & a & a \\ a & a & a & a \\ a & a & a & \end{array} \vdash_G \begin{array}{cccc} a & a & a & a \\ a & a & a & a \\ a & a & a & a \\ a & a & a & a \end{array}.$$

The sequences of rules used in these maximal derivations in G are described by the strings p_2, $p_1p_2p_3p_4$, and $p_1^2p_2p_3^3p_4^3$, respectively.

In general, a derivation in G described by the sequence of rules $p_1^n p_2 p_3^{n(n+1)/2} p_4^{n(n+1)/2}$ generates a square of side length $n+2$. □

Definition 3.3. Two n-dimensional arrays $\mathcal{A}, \mathcal{B}$ (in V^{*n} respectively $[V^{*n}]$) are called shape-equivalent, if and only if $shape(\mathcal{A}) = shape(\mathcal{B})$. Two n-dimensional array languages L_1 and L_2 are called shape-equivalent, if and only if

$$\{shape(\mathcal{A}) \mid \mathcal{A} \in L_1\} = \{shape(\mathcal{B}) \mid \mathcal{B} \in L_2\}.$$

4 The generative power of contextual array grammars

As n-dimensional contextual array productions never change a non-blank symbol, we obviously have

Lemma 4.1. Every n-dimensional array in an n-dimensional array language from $[L(n, cont, k, f)]$ is k-connected, where $f \in \{*, t\}$ as well as $k \in N$.

According to Example 3.1 we immediately obtain

Lemma 4.2. $[L(n, cont, k, f)] \subsetneq [L(n, cont, k+1, f)]$ for every $f \in \{*, t\}$ as well as all $k \in N$.

Theorem 4.1. For every $n \in N$ and every $k \in N \cup \{\infty\}$

each language in $[L(n, cont, k, *)]$ is

the coding of a language in $[L(n, cont, k, t)]$.

Proof. Let $G = (n, V, \#, P, A)$ be an n-dimensional contextual array grammar, and let us assume the contextual array productions in P

to be labelled by numbers $1, ..., r$, where $r = card\,(P)$, i. e.

$$P = \{i : (U_i, \alpha_i, W_i, \beta_i) \mid 1 \leq i \leq r\} .$$

Now consider the new alphabet

$$\begin{aligned} V' &= V \times \{0,1\}^F \quad \text{with} \\ F &= \{(i,v) \mid 1 \leq i \leq r,\ v \in U_i\} \end{aligned}$$

The idea is to mark each cell of a picture (generated in the $*$ mode by G) by a symbol (a, f), where a is the symbol from V marking the cell in the original picture and f is a function assigning the values 0 or 1 to the pairs $(i, v) : f((i, v)) = 1$ indicates that the underlying cell may be used by a selector α_i at the position $v \in U_i$ of the contextual array production i, whereas $f((i, v)) = 0$ does not allow to use the cell in this way. Thus the function f associated with a cell controls the adjoining of contexts, blocking selectively the use of the contextual array production i depending on the values of the $f(i, v)$, $v \in U_i$.

Take the finite substitution

$$\varphi : V^* \to 2^{V'^*}$$

defined by

$$\varphi(a) = \{a\} \times \{0,1\}^F ,\ a \in V,$$

and consider the new contextual array grammar

$$G' = (n, V', \#, P', A')$$

with

$$\begin{aligned} A' &= \textstyle\bigcup_{\alpha \in A} \varphi(\alpha) , \\ P' &= \{(U_i, \alpha'_i, W_i, \beta'_i) \mid 1 \leq i \leq r,\ \beta'_i \in \varphi(\beta_i) ,\ \alpha'_i \in \varphi(\alpha_i) , \\ &\qquad \alpha'_i = \{(\alpha_i(v), f_v) \mid v \in U_i\} ,\ f_v \in F \text{ for } v \in U_i, \\ &\qquad \text{and for all } v \in U_i \quad f_v((i,v)) = 1\} \end{aligned}$$

Defining the coding $h : V' \to V$ by

$$h((a, f)) = a, \ a \in V, \ f \in \{0, 1\}^F,$$

we appearently obtain $[L_* (G)] = [h (L_t (G'))]$, i.e. $[L_* (G)]$ is a coding of $[L_t (G')]$.

The type of G, i.e. the degree of connectedness k, plays no rôle, because G' is of the same type, which completes the proof. $\square$

Remark 4.1. For $n \geq 2$, the converse of Theorem 4.1 is not true, e.g. according to Example 3.2 the two-dimensional array language Sq of solid squares is in $[L(2, cont, 2, t)]$, but no language in $[L(2, cont, \infty, *)]$ is shape-equivalent with Sq: In any contextual grammar we start from a finite set of axioms, and in order to generate an arbitrarily large square we must pass through stages when the current array cannot be a square; yet such arrays would belong to $[L_* (G)]$ for any contextual array grammar G.

Moreover, no array language in $[L(n, cont, \infty, *)]$ is shape equivalent with $i_{2,n} (Sq)$.

We now show that the generative power of contextual array grammars can exceed that of context-free array grammars:

Lemma 4.3. There is a language L in $[L(2, cont, 2, t)]$ such that there is no context-free array grammar L' which is shape-equivalent with L.

Proof. Consider the set R_H of hollow rectangles with arbitrary side lengths $p, q \geq 3$. R_H can be generated by the contextual array grammar

$$G = (\{a, b\}, \#, P, \{\mathcal{A}\})$$

with

$$\mathcal{A} = \begin{array}{ll} a & \\ a & a \end{array}$$

and P containing the contextual array productions

$$p_1 = \begin{array}{c} a \\ \boxed{a} \\ \boxed{a} \end{array}, \quad p_2 = \begin{array}{cc} b & b \\ \boxed{a} & \\ \boxed{a} & \end{array}, \quad p_3 = \begin{array}{ccc} \boxed{b} & \boxed{b} & b \\ & \boxed{\#} & \end{array},$$

$$p_4 = \begin{array}{ccc} \boxed{a} & \boxed{a} & a \end{array}, \quad p_5 = \begin{array}{ccc} & & b \\ \boxed{a} & \boxed{a} & b \end{array}, \quad p_6 = \begin{array}{cc} & b \\ \boxed{\#} & \boxed{b} \\ & \boxed{b} \end{array}.$$

Starting form the axiom $\mathcal{A}$, we can go up using the rule p_1 $p-3$ times and go to the right using the rule p_4 $q-3$ times, where $p, q \geq 3$ can be chosen arbtirarily. Then we turn to the right from the vertical line by using once the rule p_2 respectively turn up from the horizontal line by using once the rule p_5; in both cases symbols b are introduced at the ends of the growing lines, whereafter the rules p_1, p_2, p_3, and p_4 cannot be applied any more.

As long as we can find two neighbouring cells occupied by symbols b in a horizontal line, where the right one of the two cells has an empty cell below it, we can use the rule p_3. As long as we can find two neighbouring cells occupied by symbols b in a vertical line, where the upper one has an empty cell on its left side, we can use the rule p_6.

These rules cannot be used any more whenever the rectangle has been completed, and this is the only case where the derivation is blocked. As each rectangle of side lengths p and q with $p, q \geq 3$ can be generated in that way, we conclude

$$R_H \in [L(2, cont, 2, *)].$$

In the following we give a typical example of a derivation yielding the rectangle with side lenghts 5 and 4:

$$
\begin{array}{cc} & \\ a & \\ a & a \end{array}
\vdash_G
\begin{array}{cc} a & \\ a & \\ a & a \end{array}
\vdash_G
\begin{array}{ccc} a & & \\ a & & \\ a & a & a \end{array}
\vdash_G
\begin{array}{cccc} a & & & \\ a & & & \\ a & a & a & a \end{array}
\vdash_G
$$

$$
\begin{array}{cccc} b & b & & \\ a & & & \\ a & & & \\ a & a & a & a \end{array}
\vdash_G
\begin{array}{ccccc} b & b & & & \\ a & & & & \\ a & & & & b \\ a & a & a & a & b \end{array}
\vdash_G
\begin{array}{ccccc} b & b & b & & \\ a & & & & \\ a & & & & b \\ a & a & a & a & b \end{array}
\vdash_G
$$

$$
\begin{array}{ccccc} b & b & b & b & \\ a & & & & \\ a & & & & b \\ a & a & a & a & b \end{array}
\vdash_G
\begin{array}{ccccc} b & b & b & b & \\ a & & & & b \\ a & & & & b \\ a & a & a & a & b \end{array}
\vdash_G
\begin{array}{ccccc} b & b & b & b & b \\ a & & & & b \\ a & & & & b \\ a & a & a & a & b \end{array}
$$

Conversely, R_H cannot be generated by a context-free array grammar, which can easily be shown by extending arguments used in [6] for similar problems; a detailed elaboration of these arguments can be left to the reader. □

As contextual array grammars cannot rewrite symbols, hence cannot replace them with #, we obviously have

Lemma 4.4. For every $n \in N$ and all $k \in N \cup \{\infty\}$,

$[L(n, cont, k, *)] \subseteq [L(n, mon, k)]$.

Proof. Starting from an n-dimensional contextual array grammar $G = (n, V, \#, P, A)$, consider the n-dimensional monotonic array grammar

$$G' = (n, V_N, V_T, \#, P', S)$$

with

$$V_N = \{a' \mid a \in V\} \cup \{S\}$$

and P' constructed as follows: Take the coding φ defined by $\varphi(a) = a'$, $a \in V$. Then P' contains the following array productions:

1. $(S, \varphi(\alpha))$ for each $\alpha \in A$.
2. $(\varphi(\alpha) \cup \beta', \varphi(\alpha) \cup \varphi(\beta))$ for each $(\alpha, \beta) \in P$, where $\beta' = \{(v, \#) \mid v \in U\}$ for $\beta = \{(v, b_v) \mid v \in U\}$,
3. $(a', \{(\Omega_n, a)\})$ for all $a \in V$.

The equality $[L_*(G)] = [L(G')]$ is obvious; moreover, by construction $\|G'\| = \|G\|$. □

Lemma 4.5. For every $n \in N$ and all $k \in N \cup \{\infty\}$,

$[L(n, cont, k, t)] \subseteq [L(n, mon, k)]$.

Proof. Take an n-dimensional contextual array grammar

$$G = (n, V, \#, P, A)$$

with $\|G\| = k$, and construct the n-dimensional monotonic array grammar

$$G' = (V_N, V, \#, P', S)$$

with P' containing the following groups of rules:

1. $S \to \alpha'$ for each $\alpha \in A$, with α' obtained from α by replacing each $a \in V$ with a' except for replacing one occurrence of some symbol $b \in V$ with $[b', X]$.
2. $(\{\Omega_n, v\}, \{(\Omega_n, [a', X]), (v, b')\}, \{(\Omega_n, a'), (v, [b', X])\})$ for each $v \in Z^n$ with $\|v\| \leq k$ and all $a, b \in V$. Using these rules, the symbol X freely circulates on the second component of markings of the current array. In every moment, exactly one occurrence of X is present and all symbols in V are primed (hence are non-terminal symbols from the point of view of G').
3. $(U_\alpha \cup U_\beta, \alpha' \cup \beta'', \alpha' \cup \beta')$ for $(U_\alpha, \alpha, U_\beta, \beta) \in P$, where α', β' are obtained from α, β, respectively, by replacing each $a \in V$ with a' except for replacing one occurrence of some $b \in V$ in

α with $[b, X]$ and one occurance of $c \in V$ in β with $[c, X]$; moreover, β'' is equal to

$(v, \#) \mid v \in U_\beta$ for $\beta = \{(v, \beta(v)) \mid v \in U_\beta\}$.

In this way, the productions of G are simulated in the presence of X.

4. $[a', X] \rightarrow [a', Y]$.

   In some moment, X is replaced by Y; from now on no rule of G can be simulated in G' any more.

5. $(\{\Omega_n, v\}, \{(\Omega_n, [x, Y]), (v, y)\}, \{(\Omega_n, x), (v, [y, Y])\})$
   for each $v \in Z^n$ with $\|v\| \leq k$ and all $x, y \in V \cup \{a' \mid a \in V\}$.

   The symbol Y freely circulates, paired both with symbols in V or with primed symbols.

6. If the cell marked with $[a', Y]$, $a \in V$, cannot be part of an area where a rule from P could be applied, we want to replace a' by a. The symbol a can appear at different positions in the selector area of a contextual array production $(U_\alpha, \alpha, U_\beta, \beta)$ in P. Hence we consider a sequence $\mathcal{R}(a)$ of triples

$$\mathcal{R}(a, i) = (U_\alpha(a, i), \alpha(a, i), U_\beta(a, i)), 1 \leq i \leq m(a)$$

   such that

   - for each $(U_\alpha(a, i), \alpha(a, i), U_\beta(a, i))$ in the sequence $\mathcal{R}(a)$ there exists a contextual array production $(U_\alpha, \alpha, U_\beta, \beta)$ in P and some $u_0(a, i) \in U_\alpha$ such that
     $U_\alpha(a, i) = U_\alpha$, $\alpha(a, i)(u_0(a, i)) = [a', Y(i)]$,
     $\alpha(u_0(a, i)) = a$, $\alpha(a, i)(u) = \alpha(u)$
     for all $u \in U_\alpha - \{u_0(a, i)\}$, and $U_\beta(a, i) = U_\beta$;
   - all possibilities of such triples corresponding to a contextual array production in P are covered by this sequence $\mathcal{R}(a)$ (observe that the sequence may also be empty).

   Therefore for each $a \in V$ we take the following array productions for P' :

(a) $[a', Y] \rightarrow [a', Y(1)]$;

(b) $[a', Y(m+1)] \rightarrow [a, Y]$;

(c) $(U_\alpha(a,i) \cup U_\beta(a,i), \alpha(a,i) \cup \{(v,\#) \mid v \in U_\beta(a,i)\},$
$\alpha''(a,i) \cup \{(v,\#) \mid v \in U_\beta(a,i)\})$, where
$\alpha'(a,i)(u_0(a,i)) = [a', Y(i)]$, yet $\alpha'(a,i) \neq \alpha(a,i)$,
$\alpha''(a,i)(u_0(a,i)) = [a', Y(i+1)]$, and
$\alpha''(a,i)(u) = \alpha'(a,i)(u)$ for $u \in U_\alpha(a,i) - \{u_0(a,i)\}$,
for every i with $1 \leq i \leq m(a)$,
where we take all such possible patterns $\alpha'(a,i)$.

7. Using such rules we can check whether some rule of P can be used "around" the cell marked with the symbol Y. If no rule of P can be used, then a' from the cell marked with $[a', Y]$ can be replaced by the corresponding terminal symbol a. The marker Y circulates across the current array and only when no rule of P can be used we obtain an array marked with terminals only except for one cell marked with $[a, Y]$.

8. $[a, Y] \rightarrow a$ for all $a \in V$.

In conclusion, the derivation in G' terminates if and only if it corresponds to a maximal derivation in G, hence $[L_t(G)] = [L(G')]$.

$$
\begin{aligned}
V_N = \; & \{S\} \cup \{a' \mid a \in V\} \cup \\
& \{[a, Z], [a', Z] \mid a \in V, Z \in \{X, Y\}\} \cup \\
& \{[a, Z], [a', Z] \mid a \in V, Z \in \{Y(i) \mid 1 \leq i \leq m(a) + 1\}\}.
\end{aligned}
$$

By construction, $\|G'\| = \|G\|$. □

Before proving the main result of our paper, i.e. that every context-free array language in $[L(n, cf, k)]$ is the coding of a contextual array language in $[L(n, cont, k, t)]$, we establish a special normal form for context-free array grammars, which is also very interesting for its own (in some sense it corresponds to the Greibach normal form for context-free string grammars).

Theorem 4.2. For every context-free array grammar

$$G = (n, V_N, V_T, \#, P, S)$$

there exists an equivalent context-free array grammar

$$G' = (n, V_N', \#, P', S')$$

with $[L(G')] = [L(G)]$ and $\|G'\| = \|G\|$ such that each of the context-free array productions in P' is of one of the following forms:

- $S' \to a,\ a \in V_T$;
- $S' \to \alpha,\ card\,(shape\,(\alpha)) \geq 2$,

  $\alpha = \{(u, \alpha(u)) \mid u \in U\},\ \alpha(u) \in V_N \cup V_T - \{S'\}$ for all $u \in U$, $\alpha(\Omega_n) \in V_T$;
- $A \to \alpha,\ A \in V_N - \{S'\};\ card\,(shape\,(\alpha)) \geq 2$,

  $\alpha = \{(u, \alpha(u)) \mid u \in U\},\ \alpha(u) \in V_N \cup V_T - \{S'\}$ for all $u \in U$, $\alpha(\Omega_n) \in V_T$.

Proof. We will elaborate the desired normal form in 3 steps:

Step 1.

We first eliminate all chain rules of the form $A \to B,\ A, B \in V_N$, from P, in order to obtain a context-free array grammar

$$G_1' = (n, V_N, V_T, \#, P_1', S)$$

with $[L(G_1')] = [L(G)]$, where we define

$$\begin{aligned}
P_1' \quad = \quad & P - \{A \to B \mid A, B \in V_N\} \cup \\
& \{A \to a \mid A \in V_N,\ a \in V_T, \\
& \quad \text{and there is a non-recurrent derivation of the form} \\
& \quad \{(\Omega_n, A)\} = \{(\Omega_n, B_1)\} \vdash_{p_1} \ldots \\
& \qquad\quad \{(\Omega_n, B_k)\} \vdash_{p_k} \{(\Omega_n, a)\} \\
& \quad \text{with } p_i \in P,\ B_i \in V_N,\ 1 \le i \le k,\ k \ge 1\} \cup \\
& \{A \to \alpha \mid A \in V_N,\ card\,(shape\,(\alpha)) \ge 2 \\
& \quad \text{and there is a non-recurrent derivation of the form} \\
& \quad \{(\Omega_n, A)\} = \{(\Omega_n, B_1)\} \vdash_{p_1} \ldots \{(\Omega_n, B_k)\} \vdash_{p_k} \alpha \\
& \quad \text{with } p_i \in P,\ B_i \in V_N,\ 1 \le i \le k,\ k \ge 1\}.
\end{aligned}$$

As a consequence of this modification we find that no rule $A \to \alpha$ from P_1' can be used twice at the same cell of an array. It is obvious from this construction that $[L(G_1')] = [L(G)]$.

Step 2.

$$G_2' = (n, V_N, V_T, \#, P_2', S)$$

such that $[L(G_2')] = [L(G_1')]$ and for each rule in $A \to \alpha$ in P_2' we have $\alpha(\Omega_n) \in V_T$.

Indeed, if P_1' contains a rule $A \to \alpha$ with $\alpha(\Omega_n) \in V_N$, then the cell marked with A where we have applied $A \to \alpha$ now is marked with the non-terminal symbol $\alpha(\Omega_n)$; to this cell again a rule $\alpha(\Omega_n) \to \beta$ has to be applied etc. until finally a terminal symbol is assigned to the cell under consideration. However, at each step, where we use a rule of the form $B \to \gamma$ with $\gamma(\Omega_n) \in V_N$, in the neighbourhood of this cell at least one new non-blank symbol is generated, because P_1' contains no chain rules any more. As the number of cells in the $\|G\|$-neighbourhood of a cell is bounded by $(2\|G\| + 1)^n - 1$, after at most $(2\|G\| + 1)^n$ derivation steps affecting the chosen cell a terminal symbol must be assigned to this cell (or otherwise, the considered sequence of rules applied in this derivation makes no sense as a part of a derivation in G' leading to a terminal array).

Hence, P_2' is obtained from P_1' by removing all rules of the form $A \to \alpha$ with $\alpha(\Omega_n) \in V_N$ from P_1', but adding all rules $A \to \beta$ with $\beta(\Omega_n) \in V_T$, where β is the result of a derivation described above, i. e. of the form

$$\alpha_1 \vdash_{p_1} ...\alpha_k \vdash_{p_k} \beta,$$

where $\alpha_1 = \{(\Omega_n, A)\}$ and for $1 \leq i \leq k$, p_i is applied to the cell at the origin Ω_n, $\alpha_i(\Omega_n) \in V_N$, and $\beta(\Omega_n) \in V_T$.

The grammar G_2' obtained in this way is equivalent with G_1' : Each new rule in P_2' corresponds to a sequence of rules from P_1'; the application of a rule from P_2' and the application of the corresponding sequence of rules from P_1' to an arbitrary array over $V_N \cup V_T$ yields the same result, hence $[L(G_2')] \subseteq [L(G_1')]$.

On the other hand, any derivation in G_1' not directly corresponding to a derivation in G_2' can be represented by a sequence of rules from P_1' in such a way that the rules rewriting the same cell can be put together in groups, yet each such block then must correspond to a rule in P_2', i. e. we obtain $[L(G_1')] \subseteq [L(G_2')]$,too.

In conclusion, $[L(G_1')] = [L(G_2')]$.

The context-free array grammar G_2' has the useful property that in every derivation in G_2' each cell once marked with a non-terminal symbol in an intermediate array is used exactly once as the left-hand member of an array production during the whole derivation.

Step 3.

In order to obtain the desired normal form for G', from P_2' all rules of the form $A \to b$, $A \in V_N$, $b \in V_T$, have to be eliminated. In order to be able to include terminal arrays of the form $[\{(\Omega_n, a)\}]$ with $a \in V_T$, we introduce a new start symbol S', i. e. we define

$$G' = (n, V_N \cup \{S'\}, V_T, \#, P', S')$$

with

$$
\begin{aligned}
P' \quad = \quad & P_2' - \{A \to b \mid A \in V_N,\ b \in V_T\} \cup \\
& \{S' \to a \mid S \to a \in P_2'\} \cup \\
& \{S' \to \alpha \mid S \to \alpha \in P_2',\ card\,(shape\,(\alpha)) \geq 2\} \cup \\
& \{A \to \alpha^* \mid A \to \alpha \in P_2',\ A \in V_N,\ card\,(shape\,(\alpha)) \geq 2, \\
& \quad \alpha = \{(v, \alpha\,(v)) \mid v \in U\},\ \alpha^* = \{(v, \alpha^*\,(v)) \mid v \in U\}, \\
& \quad \text{where for all } v \in U,\ \alpha^*\,(v) = \alpha\,(v) \text{ or } \alpha^*\,(v) = c \\
& \quad \text{for some } c \in V_T \text{ with } \alpha\,(v) \to c \in P_2'\}.
\end{aligned}
$$

The equality $[L\,(G')] = [L\,(G_2')]$ is obvious.

In conclusion, we have constructed a context-free array grammar G' with $[L(G')] = [L(G)]$ and $\|G'\| = \|G\|$, where G' has the desired properties. $\square$

In the following we prove the main result of our paper:

Theorem 4.3. Every context-free array language in $[L\,(n, cf, k)]$ is the coding of a contextual array language in $[L\,(n, cont, k, t)]$.

Proof. Let $L \in [L\,(n, cf, k)]$ be given by a context-free array grammar

$$G' = (n, V_N', V_T, \#, P', S'),$$

which, without loss of generality, we can assume to be in the normal form established in Theorem 4.6 and to have norm k.

We now construct a contextual array grammar

$$G = (n, V, \#, P, A)$$

with

$$
\begin{aligned}
A \quad = \quad & \{[a;0] \in V_T \mid S' \to a \in P'\} \cup \\
& \{\alpha' \mid S' \to \alpha \in P',\ \alpha = \{\alpha\,(v) \mid v \in U\}, \\
& \quad \alpha' = \{[\alpha\,(v)\,;0] \mid v \in U,\ \alpha\,(v) \in V_T\} \cup \\
& \quad\quad \{[\alpha\,(v)\,, c; 0] \mid v \in U,\ \alpha\,(v) \in V_N' - \{S'\}, \\
& \quad\quad\quad \text{and there is a rule } \alpha\,(v) \to \beta \text{ in } P' \\
& \quad\quad\quad \text{with } \beta\,(\Omega_n) = c\}\}.
\end{aligned}
$$

and

$$\begin{array}{rcl} V & = & \{[a;v] \mid a \in V_T,\ v \in Z^n,\ 0 \le \|v\| \le k\} \cup \\ & & \{[A,c;v] \mid A \in V_N' - \{S'\},\ c \in V_T,\ v \in Z^n,\ 0 \le \|v\| \le k\} \end{array}$$

as well as P containing the following contextual array productions:

1. $(\{(\Omega_n, [A,b;u])\}$,
   $\{(v, [\alpha(v); v]) \mid v \in U - \{\Omega_n\},\ \alpha(v) \in V_T\} \cup$
   $\{(v, [\alpha(v), c(v); v]) \mid v \in U,\ \alpha(v) \in V_N' - \{S'\},\ c(v) \in V_T,$
   and there is a rule $\alpha(v) \to \beta_v$ in P' with $\beta_v(\Omega_n) = c\}$

   for every context-free array production $A \to \alpha$ in P', where $\alpha = \{\alpha(v) \mid v \in U\}$ for some U with $\{\Omega_n\} \subsetneq U \subseteq Z^n$ and $\|U\| \le k$, with $\alpha(\Omega_n) = b$, $b \in V_T$.

2. $(\{(\Omega_n, [A,b;u])\} \cup$
   $\{(v, \alpha(v)) \mid v \in Z^n,\ 0 < \|v\| \le k,$
   and for all v with $0 < \|v\| \le k$
   $\alpha(v) \notin \{[c;v], [B,c;v] \mid c \in V_T,\ B \in V_N' - \{S'\}\}, \emptyset)\}$

   for all $A \in V_N' - \{S'\}$, $b \in V_T$, and $u \in Z^n$ with $0 \le \|u\| \le k$.

The contextual array productions in the first group of rules allow the simulations of the context-free array productions from P'. The non-blank cells in the arrays obtained in a derivation in G are marked with $[b;v]$ or $[A,b;v]$, where in the triples $[A,b;v]$ only those pairs (A,b) can be introduced that are possible according to the context-free array productions in P', i.e. there must be a corresponding rule $A \to \alpha$ in P' with $\alpha(\Omega_n) = b$.

A contextual array production from the second group of rules can only be applied to an array where we find a cell marked with $[A,b;u]$ where no cell in the k-neighbourhood at the relative position v is marked with some $[c;v]$ or$[B,c;v]$ indicating that we have already simulated some array production $A \to \alpha$ from P' to this cell, i.e. this cell corresponds to a cell still marked with the non-terminal symbol A in the corresponding derivation in G'.

As the rules from this second group of contextual array productions in P do not change the underlying array, these rules can be applied again and again, hence we can never obtain a maximal derivation in G as long as such a rule can be applied. On the other hand this shows that a maximal derivation in G ending up with an array $\mathcal{A}$, where each non-blank cell is marked with some $[b;v]$ or $[A,b;v]$, corresponds with a derivation in G' yielding a terminal array $\mathcal{A}'$, where $\mathcal{A}$ is the coding of $\mathcal{A}'$ with respect to the coding $\varphi : V \to V_T$ defined by

$$\varphi([b;v]) = b \text{ and } \varphi([A,b;v]) = b$$

for all $A \in V_N' - \{S'\}$, $b \in V_T$, and $v \in Z^n$ with $0 \leq \|v\| \leq k$. Hence we obtain the desired result; i.e.

$$\varphi([L_t(G)]) = [L(G')].$$

□

Corollary 4.1. Every context-free array language in $[L(n, cf, k)]$ is shape–equivalent with a contextual array language in $[L(n, cont, k, t)]$.

Proof. The contextual array grammar G constructed in the proof of Theorem 4.3 in the maximal derivation mode yields an array language which is shape-equivalent with the array language generated by the given context-free array grammar. □

References

[1] E. Csuhaj-Varjù, J. Dassow, J. Kelemen, and Gh. Păun, *Grammar Systems* (Gordon and Breach, London, 1994).

[2] C. R. Cook and P. S.-P. Wang, A Chomsky hierarchy of isotonic array grammars and languages, *Computer Graphics and Image Processing* **8** (1978) pp. 144-152.

[3] A. Ehrenfeucht, Gh. Păun, and G. Rozenberg, On representing recursively enumerable languages by internal contextual languages, *Theoretical Computer Science, to appear.*

[4] A. Ehrenfeucht, A. Mateescu, Gh. Păun, G. Rozenberg, and A. Salomaa, On representing RE languages by one-sided internal contextual languages, Technical Report 95-04, Leiden University, 1995.

[5] A. Ehrenfeucht, Gh. Păun, and G. Rozenberg, The linear landscape of external contextual languages, *Acta Informatica, to appear.*

[6] J. Dassow, R. Freund, and Gh. Păun, Cooperating array grammar systems, *International Journal of Pattern Recognition and Artificial Intelligence, to appear.*

[7] R. Freund, Aspects of n-dimensional Lindenmayer systems. In: G. Rozenberg, and A. Salomaa (eds.) *Developments in Language Theory* (World Scientific Publ., Singapore, *to appear).*

[8] R. Freund and Gh. Păun, One-dimensional matrix array grammars, *J. Inform. Process. Cybernet.* **EIK 29 (6)** (1993) pp 1-18.

[9] S. Marcus, Contextual grammars, *Rev. Roum. Math. Pures Appl.,* 14 (1969) pp. 1525-1534.

[10] H. Maurer, G. Rozenberg, and E. Welzl, Using string languages to describe picture languages, *Information and Control,* 54 (1982), 155-185.

[11] Gh. Păun, *Contextual Grammars* (The Publ. House of the Romanian Academy of Science, Bucharest, 1982, in Romanian).

[12] Gh. Păun and X. M. Nguyen, On the inner contextual grammars, *Rev. Roum. Math. Pures Appl.,* 25 (1980) pp. 641-651.

[13] Gh. Păun, G. Rozenberg, and A. Salomaa, Contextual grammars: erasing, determinism, one-sided contexts. In: G. Rozenberg and A. Salomaa (eds.), *Developments in Language Theory (*World Scientific Publ., Singapore, 1994) pp. 370-388.

[14] Gh. Păun, G. Rozenberg, and A. Salomaa, Contextual grammars: parallelism and blocking of derivation, *Fundamenta Informaticae, to appear.*

[15] A. Rosenfeld, *Picture Languages* (Academic Press, Reading, MA, 1979).

[16] A. Salomaa, *Formal Languages* (Academic Press, Reading, MA, 1973).

[17] S. Vicolov-Dumitrescu, On parallel contextual grammars. In: Gh. Păun (ed.), *Mathematical Linguistics and Related Topics* (The Publ. House of the Romanian Academy of Science, Bucharest, 1982).

[18] P. S.-P. Wang, Some New Results on Isotonic Array Grammars, *Information Processing Letters* **10** (1980) pp. 129-131.

[19] Y. Yamamoto, K. Morita, and K. Sugata, Context-sensitivity of two-dimensional regular array grammars. In: P. S.-P. Wang (ed.), *Array Grammars, Patterns and Recognizers*, WSP Series in Computer Science, Vol. 18 (World Scientific Publ., Singapore, 1989) pp. 17-41.

Minimierung von Muller-Automaten

Norbert Gutleben

Theoretische Informatik
BTU Cottbus
Postfach 101 344, 03013 Cottbus

gutleben@informatik.tu-cottbus.de

Zusammenfassung

Für deterministische Muller–Automaten ist bislang noch kein effizienter Algorithmus bekannt, der einen äquivalenten zustandsminimalen Automaten erzeugt. Im allgemeinen gibt es keinen eindeutig bestimmten äquivalenten zustandsminimalen Automaten. Für reguläre Sprachen, die in der Borel–Hierarchie in $F_\sigma \cap G_\delta$ liegen, wies Staiger (1993) jedoch nach, daß der Minimalautomat der eindeutig bestimmte zustandsminimalen Automaten ist. Wagner zeigte 1979, daß jeder deterministische Muller–Automat, der eine Sprache aus $F_\sigma \cap G_\delta$ erkennt, eine recht einfache Struktur hat. Wir nutzen diese Eigenschaften, um die Minimierung von Muller–Automaten auf die Minimierung von deterministischen finiten Automaten zurückzuführen.

1 Definitionen

Definition 1 $\mathcal{A} = (Q, X, q_0, \delta, \mathcal{T})$ *heißt Muller–Automat, falls*

- *Q eine endliche Menge von Zuständen ist,*

- *X ein endliches Alphabet ist,*
- *$q_0 \in Q$ der Anfangszustand ist,*
- *$\delta : Q \times X \longrightarrow Q$ die Zustandsübergangsfunktion ist und*
- *$\mathcal{T} \subseteq \mathcal{P}(Q)$ eine Menge von Finalmengen ist.*

Für ein ω–Wort $\xi \in X^\omega$ ist $inf(\mathcal{A}, \xi) := \{q \mid \exists^\omega u \preceq \xi : \delta(q_0, u) = q\}$. Ein ω–Wort ξ wird von $\mathcal{A}$ erkannt, falls $inf(\mathcal{A}, \xi) \in \mathcal{T}$. Die Menge aller von $\mathcal{A}$ erkannten Wörter wird mit $L(\mathcal{A})$ bezeichnet.
Eine Menge $T \subseteq Q$ heißt wesentlich, falls es ein $\xi \in X^\omega$ gibt mit $inf(\mathcal{A}, \xi) = T$.

Definition 2 *Sei $G := \{WX^\omega \mid W \subseteq X^*\}$. Dann ist G eine Topologie auf X^ω. Wir bezeichnen mit F die Menge aller abgeschlossenen Mengen, mit G_δ die Menge aller abzählbaren Durchschnitte von offenen Mengen und mit F_σ die Menge aller abzählbaren Vereinigungen von abgeschlossenen Mengen.*

Bekanntlich liegt jede reguläre ω–Sprache E im boolschen Abschluß von G_δ. Laut Wagner(1979) gilt für jeden Automaten $\mathcal{A} = (Q, X, q_0, \delta, \mathcal{T})$ mit $L(\mathcal{A}) \in F_\sigma \cap G_\delta$ und jede Zusammenhangskomponente Z von (Q, δ):

$$
\begin{array}{lcl}
Z \in \mathcal{T} & \Longleftrightarrow & \text{Jede wesentliche Teilmenge von } Z \text{ gehört zu } \mathcal{T} \text{ und} \\
Z \notin \mathcal{T} & \Longleftrightarrow & \text{Keine wesentliche Teilmenge von } Z \text{ gehört zu } \mathcal{T}.
\end{array}
$$

Definition 3 *Für einen Muller–Automaten $\mathcal{A} = (Q, X, q_0, \delta, \mathcal{T})$ und $q \in Q$ sei $\mathcal{A}(q) := (Q, X, q, \delta, \mathcal{T})$.*

Definition 4 *Für einen Muller–Automaten $\mathcal{A} = (Q, X, q_0, \delta, \mathcal{T})$ seien $\mathcal{A}^+ := (Q, X, q_0, \delta, F^+)$ und $\mathcal{A}^- := (Q, X, q_0, \delta, F^-)$ deterministische finite Automaten mit $F^+ := \{q \mid \forall T \subseteq Q \quad q \in T \in \mathcal{T} \Longrightarrow T \in \mathcal{T}\}$ und $F^- := \{q \mid \forall T \subseteq Q \quad q \in T \in \mathcal{T} \Longrightarrow T \notin \mathcal{T}\}$.*

Definition 5 *Sei* $\mathcal{A} := (Q, X, q_0, \delta, \mathcal{T})$ *ein Muller–Automat mit* $L(\mathcal{A}) \in F_\sigma \cap G_\delta$ *und* $P := (q_1, \ldots, q_n)$ *ein Pfad durch* Q. *P heißt* $+$*Pfad [*$-$*Pfad] der Länge* k, *falls es* $i_1, \ldots, i_k$ *gibt, mit*

1. $1 \leq i_1 < \ldots < i_k \leq n$,
2. *Für ungerade [gerade]* j *liegt* q_{i_j} *in einer akzeptierenden Teilmenge von* Q,
3. *Für gerade [ungerade]* j *liegt* q_{i_j} *in einer verwerfenden Teilmenge von* Q.

Definition 6 *Sei* $\mathcal{A} := (Q, X, q_0, \delta, \mathcal{T})$ *ein Muller–Automat mit* $L(\mathcal{A}) \in F_\sigma \cap G_\delta$ *und* $q \in Q$.*Dann sei*
$n^+(q)[n^-(q)] := max\{k \mid$ *in* q *beginnt ein* $+$*Pfad [*$-$*Pfad] der Länge* $k\}$.

Definition 7 *Sei* $\mathcal{A} := (Q, X, q_0, \delta, \mathcal{T})$ *ein Muller–Automat mit* $L(\mathcal{A}) \in F_\sigma \cap G_\delta$. *Die Relationen* $\sim, \sim^+$ *und* $\sim^-$ *auf* Q *seien definiert durch*

$$\begin{aligned} p \sim q &\iff L(\mathcal{A}(p)) = L(\mathcal{A}(q)), \\ p \sim^+ q &\iff L((Q, X, p, \delta, F^+)) = L((Q, X, q, \delta, F^+)), \\ p \sim^- q &\iff L((Q, X, p, \delta, F^-)) = L((Q, X, q, \delta, F^-)). \end{aligned}$$

2 Das Ergebnis

Lemma 1 (Wagner, 1979) *Sei* $\mathcal{A} := (Q, X, q_0, \delta, \mathcal{T})$ *ein Muller–Automat mit* $L(\mathcal{A}) \in F_\sigma \cap G_\delta$. *Seien* $p, q \in Q$ *und* T *eine wesentliche Teilmenge von* Q *mit* $q \in T$. *Dann gilt:*

1. $p \sim q \Longrightarrow n^+(p) = n^+(q), n^-(p) = n^-(q)$,
2. $T \in \mathcal{T} \Longrightarrow n^+(q) = n^-(q) + 1$,
3. $T \notin \mathcal{T} \Longrightarrow n^-(q) = n^+(q) + 1$.

Lemma 2 *Sei* $\mathcal{A} := (Q, X, q_0, \delta, \mathcal{T})$ *ein Muller–Automat mit* $L(\mathcal{A}) \in F_\sigma \cap G_\delta$. *Seien* $p, q \in Q$. *Dann gilt:*

$$p \sim^+ q \vee p \sim^- q \Longrightarrow p \sim q.$$

Beweis: Sei $p \sim^+ q \vee p \sim^- q$. Sei $\xi \in X^\omega$. Seien $T := inf(\mathcal{A}(p), \xi)$ und $T' := inf(\mathcal{A}(q), \xi)$. Dann gibt es ein $u \preceq \xi$ mit $\delta(p, u) \in T$ und $\delta(q, u) \in T'$. Wegen $p \sim^+ q \vee p \sim^- q$ folgt daraus $T \in \mathcal{T} \Longleftrightarrow T' \in \mathcal{T}$. Also gilt $p \sim q$. $\square$

Satz 1 *Sei* $\mathcal{A} := (Q, X, q_0, \delta, \mathcal{T})$ *ein Muller–Automat mit* $L(\mathcal{A}) \in F_\sigma \cap G_\delta$. *Dann gilt*

$$\mathcal{A} \text{ minimal } \Longleftrightarrow \mathcal{A}^+, \mathcal{A}^- \text{ minimal.}$$

Beweis: $\Longrightarrow$**:**Sei $\mathcal{A}$ minimal. Wäre $\mathcal{A}^+$ oder A^- nicht minimal, so gäbe es Zustände $p, q \in Q$ mit $p \sim^+ q \vee p \sim^- q$. Mit Lemma 2 folgt daraus $p \sim q$, im Widerspruch zu $\mathcal{A}$ ist minimal.
$\Longleftarrow$**:**Seien $\mathcal{A}^+, \mathcal{A}^-$ minimal. Angenommen, $\mathcal{A}$ ist nicht minimal.
Fall 1: Es gibt eine Zusammenhangskomponente Z des Graphen (Q, δ), die maximal bezgl. Erreichbarkeit ist und einen Zustand p enthält, zu dem es einen äquivalenten Zustand q mit $p \neq q$ gibt.
Dann gilt für alle $\xi \in X^\omega \quad inf(\mathcal{A}(p), \xi) \subseteq Z$. Wegen $L(\mathcal{A}) \in F_\sigma \cap G_\delta$ folgt daraus $L(\mathcal{A}(p)) \in \{\emptyset, X^\omega\}$. Wegen $L(\mathcal{A}(q)) = L(\mathcal{A}(p))$ folgt daraus

$$L((Q, X, p, \delta, F^+)) = L((Q, X, q, \delta, F^+))$$

oder

$$L((Q, X, p, \delta, F^-)) = L((Q, X, q, \delta, F^-)),$$

im Widerspruch zur Minimalität von $\mathcal{A}^+$ und $\mathcal{A}^-$.
Fall 2: Der Graph (Q, δ) besitzt keine solche Zusammenhangskomponente.
Sei $\mathcal{Z}$ die Menge aller Zusammenhangskomponenten, die einen Zustand p enthalten, zu dem es einen äquivalenten Zustand q mit $p \neq q$ gibt. Da es nur endlich viele Zusammenhangskomponenten gibt, gibt es eine Komponente $Z \in \mathcal{Z}$, die in $\mathcal{Z}$ bezgl. Erreichbarkeit

maximal ist. Seien $p \in Z$ und $q \in Q$ mit $p \neq q$ und $p \sim q$. Da Z unter allen Zusammmenhangskomponenten bezgl. Erreichbarkeit nicht maximal ist, gibt es ein $u \in X^*$ mit $\delta(p,u) \notin Z$. Es gilt dann auch $\delta(q,u) \sim \delta(p,u)$. Wegen der Maximalität von Z in $\mathcal{Z}$ ist dann $\delta(p,u) = \delta(q,u)$. Sei v das längste Präfix von u mit $\delta(p,v) \neq \delta(q,v)$ und $x \in X$ mit $vx \preceq u$. Da $\mathcal{A}^+, \mathcal{A}^-$ minimal sind, gilt $\delta(p,v) \not\sim^+ \delta(q,v)$ und $\delta(p,v) \not\sim^- \delta(q,v)$. Also gibt es $w_1, w_2 \in X^*$ mit $\delta(p,vw_1) \in F^+ \iff \delta(q,vw_1) \notin F^+$ und $\delta(p,vw_2) \in F^- \iff \delta(q,vw_2) \notin F^-$. Wegen der Maximalität von Z in $\mathcal{Z}$ folgt daraus $\delta(p,vw_1), \delta(p,vw_2) \in Z$.
Ist Z unwesentlich, so ist $p \in F^+ \cap F^-$ und $v = w_1 = w_2 = \epsilon$. Also ist $q \notin F^+ \cup F^- = Q$, was nicht sein kann.
Ist dagegen $Z \in \mathcal{T}$, so ist $\delta(q,vw_1) \notin F^+$ und damit

$$\begin{aligned} n^+(\delta(p,vw_1)) &= n^-(\delta(p,vw_1)) + 1 = n^-(\delta(q,vw_1)) + 1 \\ &= n^+(\delta(q,vw_1)) + 2 = n^+(\delta(p,vw_1)) + 2, \end{aligned}$$

was nicht sein kann.
Ist schließlich $Z \notin \mathcal{T}$, aber Z wesentlich, so ist $\delta(q,vw_2) \notin F^-$ und damit

$$\begin{aligned} n^-\delta(p,vw_2)) &= n^+(\delta(p,vw_2)) + 1 = n^+(\delta(q,vw_2)) + 1 \\ &= n^-(\delta(q,vw_2)) + 2 = n^-(\delta(p,vw_2)) + 2, \end{aligned}$$

was nicht sein kann. $\square$

On the Complexities of Deterministic Indexed Grammars*

(Extended Abstract)

Markus Holzer and Klaus-Jörn Lange
Wilhelm-Schickard Institut für Informatik
Universität Tübingen
Sand 13, D-72076 Tübingen
{holzer,lange}@informatik.uni-tuebingen.de

There are many close connections between complexity theory and the area of formal languages and automata. Typical relations of this kind are completeness results for families of formal languages. Well-known examples are:

1. *REG*, the family of regular languages, is NC^1-complete (Barrington [3]),
2. *LIN*, the family of linear context-free languages, is *NSpace*(log)-complete (Sudborough [7]),
3. *CFL*, the family of context-free languages is $\mathit{NAuxPDASpaceTime}(\log n,\ \mathit{poly})$-complete (Sudborough [8]),
4. *IND*, the family of indexed languages, is *NP*-complete (Fischer [4]).

*Part of the work was done during the authors worked at Institut für Informatik, Technische Universität München, Arcisstr. 21, D-80290 München, Germany.

Several restrictions of the family of context-free languages, which are motivated from compiler construction, have been extensively studied in the literature. Here we are interested in LL(1) and LR(1) restrictions only.

The idea underlying both LL(1) and LR(1) conditions is the same, i.e., to make a language deterministic parsable. But the crucial difference between the conditions is how to obtain this determinism. In the case of LL(1) grammars the parsing process works in a top-down manner, whereas the process works bottom-up for LR(1) grammars. A third method, which is in fact the most natural, is to define determinism via the corresponding automaton model, and not via a grammar type.

In formal languages it is well-known that context-free LR(1) grammars generate exactly the deterministic context-free languages and form a proper superset of the context-free LL(1) languages (see Hopcroft and Ullman [6]). Nevertheless all these classes are complete for $DAuxPDASpaceTime(\log n, poly)$ (Sudborough [8]).

Here, one could ask whether determinism of the automaton model, LL(1), and LR(1) conditions always behaves similar with respect to the complexity theoretical point of view. Unfortunately, such a general statement is not true, which was shown by Holzer and Lange [5] considering linear context-free languages. It was shown that linear context-free LR(1) grammars generate exactly the deterministic linear context-free languages and that the complexity theoretical equivalence of LL(1) and LR(1) conditions as in the context-free language case are no longer valid in the presence of linear context-free grammars, since here their equivalence would imply $NC^1 = DSpace(\log n)$.

Based on context-free grammars Aho [1] introduced indexed grammars, languages, respectively. The main difference between context-free grammars, indexed grammars, respectively, is that a nonterminal can be viewed as a function call without parameter, with parameter (evaluated in a call-by-value like way), respectively. For a precise definition see Aho [1]. In addition, Aho [2] also found a nice characterization of indexed languages in terms of stack automata. This automaton provides a mathematical model for a restricted form of embedded list structure, and is therefore called nested stack automaton.

The situation for deterministic versions of indexed languages is more involved as in the context-free case. It is easy to define determinism via the automaton and via LL(1) conditions on the grammar type. A condition playing the role of LR(1) as in the context-free case is up to now not known. Here bottom-up parsing is much more complicated, because of the above mentioned function calls with parameters.

We show that both deterministic indexed languages, i.e., languages accepted by deterministic nested stack automata, and indexed LL(1) languages are complete for the deterministic counterpart of *NP*, i.e., we show P-completeness. In addition we study variations of indexed languages. One variation consists of restricting the underlying grammar to regular or linear indexed grammars. The other variation is to restrict the usage of the indices, i.e., the values of the parameters during a function call (see Aho [1]).

In most cases, the complexity of indexed and indexed LL(1) languages is well determined, and behaves as we expect. This means that whenever a class of indexed languages is $\mathcal{A}$-complete, then the LL(1) counterpart is complete for the deterministic counterpart of $\mathcal{A}$.

References

[1] A. V. Aho. Indexed grammars—an extension of context-free grammars. *Journal of the ACM*, 15:647–671, 1968.

[2] A. V. Aho. Nested stack automata. *Journal of the ACM*, 16(3):383–406, July 1969.

[3] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in $\mathcal{NC}^1$. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[4] M. J. Fischer. Grammars with macro-like productions. Ph.D. thesis, Harvard University, 1968.

[5] M. Holzer and K.-J. Lange. On the complexities of linear LL(1) and LR(1) grammars. In *Proceedings of the 9th International Conference on Fundamentals of Computation Theory*, number 710 in LNCS, pages 299–308. Springer, August 1993.

[6] J. E. Hopcroft and J. D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley, 1968.

[7] I. H. Sudborough. A note on tape-bounded complexity classes and linear context-free languages. *Journal of the ACM*, 22(4):499–500, October 1975.

[8] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25:405–414, 1978.

Effiziente Speicherstrukturen für Turingmaschinen mit *einem* Kopf

Martin Hühne

Lehrstuhl Informatik II
Universität Dortmund
44221 Dortmund

huehne@ls2.informatik.uni-dortmund.de

Die Effizienz von Turingmaschinen hängt sehr stark von der Anzahl und der Struktur ihrer Arbeitsspeicher ab. Wir untersuchen den Einfluß der Speicherstruktur auf die Effizienz von deterministischen Turingmaschinen mit genau einem Schreib/Lesekopf und geben obere Zeitschranken für die Simulation von Mehrband Turingmaschinen auf solchen Ein-Kopf Turingmaschinen an.

Die Struktur des Arbeitsspeichers ist fest und wird durch einen Graphen beschrieben [Re90]. Die abzählbar unendlich vielen Knoten des Graphen repräsentieren die Speicherzellen. Der Graph hat gefärbte, gerichtete Kanten, die die möglichen Bewegungen des Kopfes repräsentieren. Die Anzahl der ausgehenden Kanten pro Knoten ist beschränkt.

Bekannte und vielfach untersuchte Speicherstrukturen sind das Band, das Gitter, und der Baum. Ein-Kopf Turingmaschinen mit der Speicherstruktur "Band" sind sehr ineffizient, da für die Simulation von t Schritten einer Mehrband Turingmaschine Zeit $O(t^2)$ benötigt wird

[HS65]. Auf Ein-Kopf Turingmaschinen mit der Speicherstruktur "d-dimensionales Gitter", $d \geq 2$, kann die Simulation in Zeit $O(t^{1+1/d} / \log t)$ erfolgen [DH94]. Ein-Kopf Turingmaschinen mit der Speicherstruktur "binärer Baum" können Mehrband Turingmaschinen sogar in Zeit $O(t \log t)$ simulieren [PR81]. Diese Zeitschranken sind scharf. Schnellere deterministische Simulationen von Mehrband Turingmaschinen sind nur auf Modellen bekannt, die mehr als einen Kopf besitzen.

Gibt es Speicherstrukturen für Ein-Kopf Turingmaschinen, sodaß Mehrband Turingmaschinen schneller als in Zeit $t \log t$ simuliert werden können?

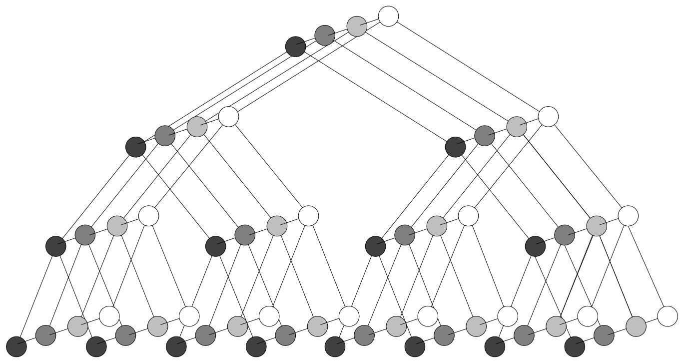
Man kann leicht eine Speicherstruktur definieren, in deren Kanten alle möglichen Berechnungen von Turingmaschinen kodiert sind. Sei $\mathcal{C} = \{L_1, L_2, L_3, \ldots\}$ eine abzählbare Menge von Sprachen über dem Alphabet $\{0,1\}$. Betrachte die baumartige Speicherstruktur mit Knotenmenge $\{0,1\}^*$, in der neben den Baumkanten für jedes Wort $x_1 \ldots x_j \in L_i$ eine Kante von der Speicherzelle $1^i\, 0\, x_1 \ldots x_j$ zur Wurzel des Baumes vorhanden ist. Eine Ein-Kopf Turingmaschine mit dieser Speicherstruktur kann auf einfache Weise alle Sprachen in $\mathcal{C}$ entscheiden — und zwar selbst dann, wenn einzelne oder gar alle Sprachen in $\mathcal{C}$ nicht rekursiv sind.

Daher ist es offensichtlich notwendig, die Menge der betrachteten Speicherstrukturen geeignet einzuschränken. Reischuk nennt eine Speicherstruktur *konstruierbar*, falls jede Speicherzelle eine kurze, eindeutige Adresse hat und falls es eine linear zeitbeschränkte Zwei-Band Turingmaschine gibt, die bei Eingabe der Adresse einer Speicherzelle die Adressen aller Nachbarzellen sowie die Färbung der zugehörigen Kanten berechnet [Re90].

Im Vortrag werden verschiedene konstruierbare Speicherstrukturen vorgestellt, auf denen Mehrband Turingmaschinen schneller als auf dem Baum simuliert werden können.

Die erste solche Speicherstruktur ist die "Hecke". Die Hecke läßt sich als das kartesische Produkt von Baum und Band beschreiben. Auf einer Ein-Kopf Turingmaschine mit Speicherstruktur "Hecke" können t

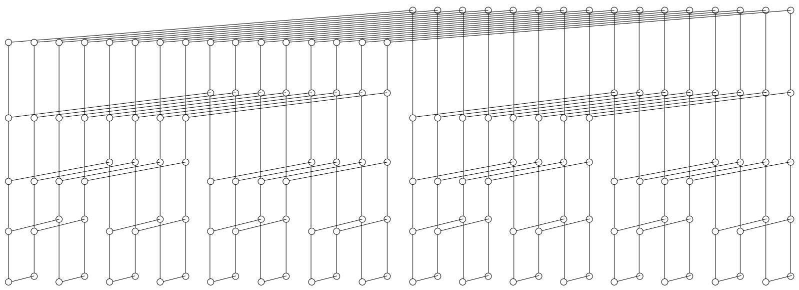
Schritte von Mehrband Turingmaschinen in Zeit $O(t \log t / \log\log t)$ simuliert werden. Grundlage für die Simulation ist ein schneller Kopieralgorithmus. Üblicherweise benötigt eine Ein-Kopf Turingmaschine Zeit $\ell \cdot \Delta$, um einen String der Länge ℓ an eine andere Stelle im Speicher zu kopieren, die Δ Speicherzellen entfernt ist. Wir benutzen eine zuerst auf dem Gitter entwickelte Technik [Ti87, DH94], mit der diese Kopieraufgabe mit einem Kopf in Zeit $O(\ell \cdot \Delta / \log \Delta)$ gelöst werden kann. Allerdings ist vor dem Kopieren eine einmalige Vorbereitungszeit erforderlich ($O(\Delta^{3/2})$). Diese Technik kann nicht auf dem Baum oder auf dem Band angewendet werden.



Ein Ausschnitt aus der Speicherstruktur "Hecke".

Die Speicherstruktur "Cube Connected Cycles" (CCC) ist von der gleichnamigen Parallelrechnerarchitektur abgeleitet, mit der verteilt arbeitende Prozessoren effizient gekoppelt werden können [PV81, MadH83]. Ein-Kopf Turingmaschinen mit der Speicherstruktur "CCC" sind besonders effizient. Sie können Mehrband Turingmaschinen in Zeit $O(t\sqrt{\log t})$ simulieren. Auch diese Simulation basiert auf einer schnellen Kopiertechnik. (Kopierzeit: $O(\Delta + \ell^2)$, Vorbereitungszeit: $O(\Delta^2\, 2^{\sqrt{\Delta}})$.)

Auf der Hecke und auf dem CCC können auch Turingmaschinen mit mehreren anderen Speicherstrukturen effizient simuliert werden. So können t Schritte von Turingmaschinen mit mehreren d-dimensionalen Gittern auf einer Turingmaschine mit einer einzelnen Hecke in Zeit



Ein Ausschnitt aus der Speicherstruktur "Cube Connected Cycles".

$O(t\ \log t\,/\,\sqrt[d]{\log\log t})$ und auf einer Turingmaschine mit einem CCC in Zeit $O(t\ \log t\,/\,\sqrt[2d]{\log t})$ simuliert werden. Eine Hecke simuliert Turingmaschinen mit mehreren Bäumen oder Hecken in Zeit $O(t\ \log t\,/\,\log\log\log t)$.

Wir beschreiben eine konstruierbare Speicherstruktur, auf der t Schritte von Mehrband Turingmaschinen in Zeit $O(t)$ simuliert werden können. Die simulierende Maschine nutzt ihre Speicherstruktur in einer ungewohnten Weise. Anstatt die Beschriftung der Bänder in den Zellen ihres Speichers abzulegen, merkt sie sich die Beschriftung der Bänder allein durch die *Position* ihres Kopfes. Die Linearzeit-Simulation funktioniert auch dann, wenn die simulierende Maschine ihren Speicher gar nicht beschreiben darf, sondern nur einen Pebble im Speicher verschieben kann.

Wir beschreiben eine weitere konstruierbare Speicherstruktur, auf der t Schritte von Zwei-Band Turingmaschinen, die Platz s benutzen, in Zeit $O(\sqrt{t \cdot s} + n^2)$ simuliert werden können. (Hierbei bezeichnet n die Eingabelänge.) Auch hier werden Bandbeschriftungen vollständig durch die Position des Kopfes dargestellt. Mit einer einzelnen Bewegung des Kopfes in dieser Speicherstruktur werden gleich mehrere Schritte der Zwei-Band Turingmaschine simuliert. Nach einem Resultat von Hartmanis benutzen Ein-Band Turingmaschinen in Zeit t maximal Platz $O(n + t\,/\,\log t)$ [Ha68]. Damit können t Schritte von Ein-Band Turingmaschinen, $t = \Omega(n^2 \log n)$, in sublinearer Zeit $O(t\,/\,\sqrt{\log t})$ auf dieser konstruierbaren Speicherstruktur simuliert werden. Dies ist möglicherweise ein Hinweis darauf, daß Reischuks Definition konstruierbarer Speicher zu weit gefaßt ist.

Ich danke Martin Dietzfelbinger für wertvolle Hinweise und für die Gelegenheit zu ausführlichen Diskussionen. Diese Arbeit wurde durch die Deutsche Forschungsgemeinschaft unterstützt (DFG Projekt Di 412/2-2). Die Ergebnisse über die Speicherstruktur "Hecke" wurden auf der letzten MFCS vorgestellt [Hü95].

Literatur

[DH94] M. Dietzfelbinger und M. Hühne, Matching upper and lower bounds for simulations of several tapes on one multidimensional tape. In *Proc. 14th FST & TCS*, LNCS 880, 1994, 24–35.

[Ha68] J. Hartmanis, Computational complexity of one-tape Turing computations. *Journal of the ACM* **15** (1968), 325–339.

[HS65] J. Hartmanis und R.E. Stearns, On the computational complexity of algorithms. *Transactions of the American Mathematical Society* **117** (1965), 285–306.

[Hü95] M. Hühne, The hedge: An efficient storage device for Turing machines with one head. In *Proc. 20th MFCS*, LNCS 969, 1995, 247–256.

[MadH83] F. Meyer auf der Heide, Infinite cube-connected cycles. *Information Processing Letters* **16** (1983), 1–2.

[PR81] W.J. Paul und K.R. Reischuk, On time versus space II. *Journal of Computer and Systems Sciences* **22** (1981), 312–327.

[PV81] F. Preparata und J. Vuillemin, The cube-connected cycles: A versatile network for parallel computation. *Communications of the ACM* **24** (1981), 300–309.

[Re90] K.R. Reischuk, *Einführung in die Komplexitätstheorie*. B.G. Teubner, Stuttgart, 1990.

[Ti87] P. Tiwari, Lower bounds on communication complexity in distributed computer networks. *Journal of the ACM* **34** (1987), 921–938.

Introducing a Two-Level Grammar Concept for Design*

Klaus P. Jantke†

Hochschule für Technik, Wirtschaft und Kultur Leipzig
FB Informatik, Mathematik und Naturwissenschaften
Postfach 30066, 04251 Leipzig

jantke@informatik.th-leipzig.de

1 Introduction

The present papers focusses on the introduction of a novel concept of formal grammars. According to the case-based target application sketched below, the ultimate goal is to develop, investigate, and apply the new concept introduced in the area of formal graph languages. Nevertheless, the key ideas will be developed for string languages first. This should enable the reader to easily understand the essentials of the new approach. The paper on hand is intended to be a launching pad for in-depth investigations focussed on learning.

*This work has been partially supported by the German Federal Ministry for Education, Science, Research and Technology (BMBF) within the BMBF Joint Project Fabel under grant 413–4001–01 IW 101. The learning-theoretic aspects have been investigated within the project Ind–Cbl supported by the Deutsche Forschungsgemeinschaft (DFG) under grant Ja 566/2-1.

†http://www.informatik.th-leipzig.de

1.1 Case-Based Reasoning

Case-based reasoning is deemed an important technology to alleviate the bottleneck of knowledge acquisition in Artificial Intelligence. Publications are mushrooming. Hence, it is almost impossible to give a reasonable overview. [RS89] and [Kol93] are widely accepted textbooks. There are several introductory papers like [Kol92], e.g. In case-based reasoning, knowledge is represented in the form of particular cases or episodes with an appropriate similarity measure relating them to each other rather than any form of generalized or abstracted rules. One of the underlying key motivations is to avoid unjustified generalization and abstraction during knowledge acquisition. Instead, emphasis is put on adequately representing cases in a so-called case base. This is taken as a basis for further automated reasoning. When new problems are to be solved, a case-based reasoning system may search its case base for similar problems considered in the past. If more or less similar cases are found, the system may try to adapt them to the needs of the recent problem on hand. Thus, similarity-based retrieval and adaptation are essential steps of case-based reasoning.

Cognitive psychology is providing sufficient evidence of the importance of case-based reasoning in automating particular processes of human reasoning adequately. George Lakoff's book [Lak87] is a very nice source putting emphasis on categorization. The early work by Eleanor Rosch (cf. [Ros75], e.g.) contains much details motivating elaborated approaches towards automated case-based reasoning.

Recently, there is an enormous amount of approaches and applications. The majority of publications adopts an attribute-value-pair descrition of cases. In a sense, there is assumed some n-dimensional space for case representation. Every particular case is characterized by some possibly incomplete vector of attribute values. This is appropriate for a huge amount of application domains. But in many interesting domains posing exciting problems to Artificial Intelligence, representing cases in an attribute-value-pair style means already substantial abstraction and generalization (cf. [EOSW94] and

[Sch95], e.g.). This essentially misses the original motivation of case-based reasoning approaches. Episodic knowledge should be kept in a form as close as possible to its appearence. Knowledge elicitation is widely underestimated, and investigations like [Gor92], [PAG92], and [RH92] are frequently disregarded in computation-oriented Artificial Intelligence.

The inappropriateness of the attribute-value-pair approach bears abundant evidence of the need of structural case representations together with structural concepts of similarity. The author's paper [Jan94] is intended to be a launching pad for investigations of this type. In response, KATY BÖRNER (cf. [Bör94]) is striving hard for developing and implementing structural similarity concepts in architectural design, e.g.

There is a common motivation for approaches like in [Jan94] and [Bör94]) drawn from the authors' work within the BMFT joint project FABEL (cf. [FC93]). Although the FABEL project is quite application-oriented, it is continuously generating exciting theoretical questions. The present paper is exclusively focussed on those theoretical problems.

Another aim of the present paper is to develop certain fundamentals for applying inductive learning techniques in case-based reasoning. A formal framework for case-based learning has recently been developed by [Jan92], [JL93], and [SJL94] in an inductive inference manner.

1.2 Case-Based Learning

The crux of case-based reasoning is that the intended automated reasoning process provided by a case-based reasoning system should be mainly based on prior experience found somehow similar to a certain problem on hand. This similarity has to be determined in an early reasoning phase, whereas its success can only be verified finally. The unavoidable incompleteness of information based on earlier experience makes case-based reasoning a process of inductive hypothesizing.

Therefore, the intelligent behaviour of case-based reasoning systems can be substantially improved by inductive learning techniques. Those techniques are invoked within several application scenarios of case-based reasoning. An in-depth investigation of inductive learning in case-based reasoning needs a number of prerequisites like well-defined case concepts and similarity approaches.

From the overwhelming scenarios of embedding inductive learning into case-based reasoning, a prototypical example should do for illustration. Assume an interactive design support system possessing its case base together with some similarity measure. Whenever a user is designing a new case, (s)he may ask the system for formerly solved problems similar to a specific problem presented. Usually, the system will respond to the request with one or more cases of maximal similarity. Let us assume that there is always some case among the most similar cases presented which fits the users needs, but which is usually not the most similar one. The user may whish to correct the underlying similarity concept in such a way that his preferred cases would be presented as proposals of highest similarity. This means to change this similarity concept by inductive learning. The input to the assumed learning device is quite clear. First of all, there is a current similarity concept to be modified incrementally. Second, there is a permanently changing case base collecting the system's experience. Third, there is the user's input consisting of pairs containing a recent problem together with a desired output case to be of highest similarity. A sequence of user inputs should result in a sequence of concept changes. In the best case, this may converge to an a priori unknown target similarity concept.

The scenario sketched is a typical instance of the general inductive inference scenario (cf. [AS83], [KW80], [Jan89]) of learning in the limit from usually incomplete information. In the present paper, we are more particularly faced to problems of language learning (cf. [Vid94]). The approach is even more particular, as we are interested in two-level grammar concepts for generating graphs (see below). There are several further motivations of investigating two-level grammars, especially in computational linguistics (cf. [Kos83], e.g.). Inductive learning of two-level grammars is restricted two word languages, so far, like in [Ita94].

The present paper does not deal with the development of inductive learning for advanced case-based reasoning. Its intention is to introduce an appropriate graph-theoretic concept flexible and expressive enough to reflect essential cases of the FABEL application domain. The precision of the introduced concepts and their relation to other formal language concepts should provide a firm background for the intended development and application of inductive learning methods.

2 Structural Case Representation in Architectural Design

The present papers addresses case-based reasoning applications where the main objects of reasoning are highly structured. In those domains, exploiting the internal structures of objects is usually crucial.

Many computer science applications are substantially suffering from what the author calls the *paradoxon of levelling.* Although objects in the real world are usually very complex, have a certain hierarchical structure or internal topology, knowledge representation on computers is initially levelling this structure down, in most cases. In later phases of knowledge processing, enormous efforts are necessary to reconstruct the structural information levelled before. This seems to be somehow paradox.

Our underlying FABEL project is an excellent example in this regard. Typical target objects considered within the FABEL application are fresh air supply nets of buildings, for instance. In reality, these nets are highly structured, consist of certain levels both due to stages of architectural design and due to types of constructive parts, have an obvious topology, and are reasonably understood as graphs. However, as an effect of using a particular CAD system during design, these objects are levelled down to lists of tupels representing a bunch of attribute/value pairs. This leads to the somehow crazy necessity to rebuild structured objects from flat representations. In [Bör94] there is sketched the problem of transforming tupel representations back into structured objects for an appropriate case-based knowledge processing. This is described in chapter 7.2 of [BF95] in some more detail.

The present paper focusses on structural concepts for appropriate case-based reasoning, particularly in architectural design. It is an implementation decision whether or not to make a detour via flat case representations when invoking the concepts developed. This may essentially depend on the available hard- and software tools. However, those considerations are beyond the present investigation.

2.1 Case Structures in FABEL

In the Fabel approach (cf. [FK93] and [FC93]), we are dealt with certain architectural design problems in industrial building. Typical applications are designing complete air supply systems of buildings. The net structures to be created interactively may also represent electricity or water supply, for instance. Characteristically, those nets are huge structures of a high regularity which may be reasonably understood as graphs.

A typical example is depicted on the following page. This example will be used for a more detailed discussion subsequently.

2.2 Graph Objects in Architectural Design

The recent scientific literature yields an extremely inhomogeneous impression of the area. The early book [SG78] has set the stage for shape grammars in architecture. There are a score or more approaches like the one in [Bro94] which sound quite promising, but do not provide any precision. An extreme example is [Lee94] which announces “the formal basis ...”, but does not contain any formalism.

In contrast, recent publications which adopt a formal language approach are much more precise and well-based. A quite comprehensive publication is [Shi94]. But in [Shi94], [MG87], and [Mac91], the grammars under consideration are essentially string grammars, though the semantics of strings may be of higher dimension. This applies to [GLK94] as well. However, [MG87] and [GLK94] address learning problems which are also motivating the present publication.

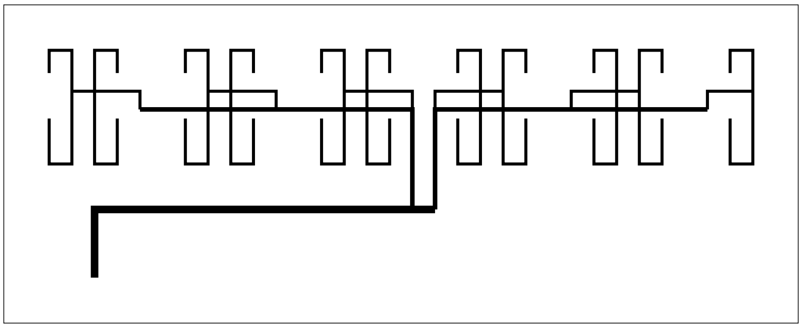


Figure 1: Part of a Typical Fresh-Air Supply Structure

In a minority of papers, one may find well-explained concepts of a certain precision. A valuable introduction leading to grammars for defining topological changes may be found in [HSF+92]. There, the authors have developed a clearly motivated two-level concepts of grammars for architectural design. The languages in use are essentially regular. It is one of the key intentions of the present paper to generalize this approach considerably. Similarly, the grammatical concepts in [GLK94] are both well-formalized and understandable. But this paper focusses on genetic algorithms for evolutionary grammar learning. This is beyond the scope of the present paper.

3 Two-Level Design Grammars

The present chapter is devoted to the introduction of two-level graph grammar concepts.

On the upper level as illustrated by the figure below, classical concepts of the CHOMSKY hierarchy are invoked. This figure illustrates the skeleton of some target graph to be described by conventional grammar rules.

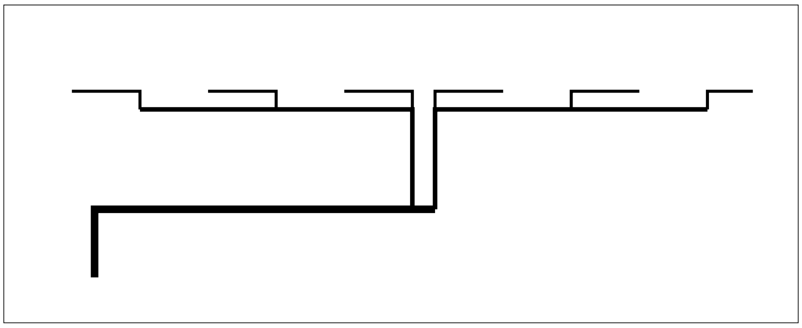


Figure 2: The CHOMSKY Part of a Target Graph

On the lower level, the idea of string patterns is adopted. For the classical formal language concepts, there is a large amount of topical publications. [HU79] is an appropriate reference. The basic concepts of pattern languages have been introduced by DANA ANGLUIN in [Ang80].

In dependence on the underlying application, those rules may be regular or context free. The resulting graph usually possesses some clearly marked nodes which are to be completed by attaching certain subgraphs taken from a class of more or less well-defined prototypes. This process of completion is somehow similar to the process of variable substitution for pattern languages (cf. [Ang80]). The next figure is intended to display the "substitution" of "variables" implicitly contained in the figure above.

The ultimate goal of the overall approach behind the present paper is to learn appropriate grammatical concepts from actual design processes. This needs clear formalisms to be outlined here.

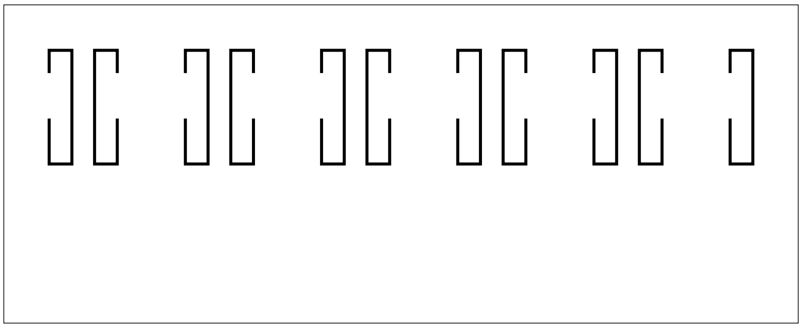


Figure 3: The Pattern Part of a Target Graph

For illustration, the reader may overlay the two structures of Figure 2 and Figure 3 to get the target fresh air supply net of Figure 1. Formally, this overlapping may be interpreted as variable substitution to be formalized below.

First, our basic ideas are developed for string languages, for readability. It is assumed that most readers are familiar with the fundamentals of classical formal language theory. Thus, the introduction of our key concepts may look like an easy tour beyond the boundaries of classical theory. Second, they will be straightforwardly generalized to cover graph concepts

3.1 String Languages

The presentation follows classical concepts as tight as possible. When introducing graph grammars below, it seems reasonable to restrict our attention to only context-free substitutions. Therefore, we will apply the same restriction to the novel concepts of the string language case discussed first.

Definition 1

A two-level context-free design grammar $\mathcal{DG} = [N, T_1, V, T_2, R, s]$ is given by the following parameters:

- N is a non-empty set (the so-called nonterminals).
- T_1 is a (possibly empty) set (the so-called first-level terminals).
- V is a non-empty set (the so-called pattern variables).
- T_2 is a non-empty set (the so-called second-level terminals).
- For R it holds $R \subseteq N \times (N \cup V \cup T_1)^+$ (the rule system of the grammar).
- For s it holds $s \in N$ (the start symbol of the grammar).

A design grammar $\mathcal{DG} = [N, T_1, V, T_2, R, s]$ is said to be finite, if all sets N, T_1, V, T_2, and R are finite. $\square$

Usually, grammars are used to define languages. We will introduce a formal semantics specifing for every grammar $\mathcal{DG}$ its formal language $L(\mathcal{DG})$. The standard concepts of the CHOMSKY hierarchy are assumed. Additionally, we invoke the basic concepts of pattern languages (cf. [Ang80]).

In particular, the following notions and notations are in use:

- For every CHOMSKY grammar $\mathcal{G}$ on any alphabet A, $L(\mathcal{G})$ denotes the formal language generated by $\mathcal{G}$ over A.
- For every pattern p on any alphabet A and for any set of variables X disjoint with the alphabet A, $L(p)$ denotes the pattern language generated by p via non-erasing substitutions over A.

Definition 2

The formal semantics of any context-free design grammar $\mathcal{DG} = [N, T_1, V, T_2, R, s]$ is some formal language $L(\mathcal{DG})$ on the alphabet $T_1 \cup T_2$ defined by

$$L(\mathcal{DG}) \;=\; \bigcup_{p \in L(\mathcal{G})} L(p)$$

where

- $\mathcal{G} = [N, V \cup T_1, R, s]$ is a CHOMSKY grammar and
- every string $p \in L(\mathcal{G})$ is a pattern with variables in V varying over T_2 .

For any given alphabet A, the notation $\mathcal{L}_{DG}$ means the class of all formal languages defined by any context-free design grammar over $A = T_1 \cup T_2$. □

For notational convenience, we call $\mathcal{G}$ the CHOMSKY part of $\mathcal{DG}$.

One could easily introduce a couple of requirements of naturalness. For instance, if $V \neq \emptyset$ and $L(\mathcal{G}) \setminus T_1^* \neq \emptyset$, one should reasonably assume $T_2 \neq \emptyset$. Vice versa, if V is empty, one does not need any letter in T_2. The investigation of dependencies of this type is left to the reader as an introductory excercise.

For extremely simple special cases, one gets well-known concepts. For instance, $V = \emptyset$ implies that $L(\mathcal{DG})$ is a usual CHOMSKY language. If $L(\mathcal{G})$ is a singleton set, $L(\mathcal{DG})$ is a standard pattern language. Already in the simple case where $L(\mathcal{G})$ is finite, we are faced to a type of formal languages not investigated so far.

Throughout the paper, we will present some initial results which are quite simple. Thus, we call them *propositions* instead of *theorems*. Proofs are sketched, only.

Proposition 1

There are context-free design grammars with a CHOMSKY part which is regular, but which generate formal languages which are not context-free.

Proof: One may trivially choose $\mathcal{G}$ to generate the singleton language $\{ axaxax \}$ with $T_1 = \{ a \}$ $V = \{ x \}$. For any singleton alphabet T_2 with a letters different from a (say b, without loss of generality), the resulting language $L(axaxax)$ turns out to be *not* context-free. Interestingly, there is no proof based on the pumping theorem for context-free languages (cf. [HU79]), as the pattern language $L(axaxax)$ satisfies the corresponding pumping condition. However, the existence of a stack automaton for accepting $L(axaxax)$ would imply the existence of a similar automaton for the language $\{ a^n b^n c^n \mid n = 1, 2, 3, \dots \}$. But this is a standard counter-example known to be not context-free (cf. [HU79]). ♣

There is another somehow complimentary result, where we take into account even more general design grammars:

Proposition 2

There are design grammars with a CHOMSKY part which is even more complex than context-sensitive, but which generate formal languages which are regular.

Proof: One may choose any formal grammar $\mathcal{G}$ not being context-sensitive. Without loss of generality, we assume the start symbol s. Furthermore, one may assume V to be some sufficiently large, but finite subset of $\{ x_1, x_2, x_3, \dots \}$. There is some rule "$s \longrightarrow x_1 x_2 \cdots x_k$" such that the grammar $\mathcal{G}'$ resulting from adding this particular rule to the rule set of $\mathcal{G}$ is still not context-sensitive. $\mathcal{DG}'$ denotes the (generalized) design grammar with its CHOMSKY part $\mathcal{G}'$. Trivially, $\mathcal{L}(\mathcal{DG}')$ is co-finite, as it contains all words of length not smaller than the constant k. Consequently, this language is regular, since all finite and co-finite languages are regular. ♣

There is a couple of reasonable generalizations and refinements of the basic approach introduced above. For example, one may generalize the approach to multi-layer grammars. Furthermore, one may refine the approach by assigning particular alphabets to different variables in V, or by guarding substitutions. In the author's opinion, one should stick with the basic approach for some time. First, this approach seems to meet the needs of the driving application described above quite well. There is no obvious need to develop more than two levels. Second, one should try to solve some of the open problems in the simplest setting before attacking more general cases. Hence, the approach will not be generalized throughout this paper.

A few investigations may illustrate the differences of our new concepts from known hierarchies.

As we know already fron prpervation 1 above, even if $L(\mathcal{G})$ is regular, $L(\mathcal{DG})$ may be not context-free. We present a few further trivialities to warm up:

Proposition 3

In case $|V| = 1$, $L(\mathcal{DG})$ is regular if and only if $L(\mathcal{G})$ is regular.

If T_1 and T_2 are disjoint, it holds:

1. Membership for $L(\mathcal{DG})$ is uniformly decidable.
2. Emptyness for $L(\mathcal{DG})$ is uniformly decidable.
3. Finiteness for $L(\mathcal{DG})$ is uniformly decidable.

Proof: These results are immediately inherited from classical results in formal language theory (cf. [HU79]). ♣

Further properties seem to require particular assumptions. For the intended application domain, languages which satisfy $T_1 \setminus T_2 \neq \emptyset$ are of a particular importance. It may also be of a special interest to consider languages with a certain rate of letters from $T_1 \setminus T_2$ in terminal words.

3.2 Graph Grammars

The area of graph grammars is quite rich (cf. [KR90], e.g., and [SG78], [GLK94], for shape grammars). First, one needs to specify whether the generation process should be based on node replacements or edge replacement (or even hyperedge replacements). The FABEL application does no govern this decision in any way. Throughout this paper, we deal with node substitutions, exclusively.

A key question is how to embed graphs when inserted for a node of a given host graph. The embedding rule essentially determines the peculiarities of the whole approach. In the case of undirected graphs, there is an enormous number of concurring ideas.

In the present paper, we assume directed acyclic graphs. This approach has two basic roots. First, it is a direct generalization of approaches for two-level word grammars as sketched in [HSF+92]. Second, it is based on graph grammar concepts mainly proposed by OKSANA ARNOLD in [AJ94b], [AJ94c] and [JA95]. Although these grammatical concepts are originally designed to meet the necessities of therapy plan generation for complex dynamic processes, they fit our present needs quite well.

The non-terminal symbols of the above definitions collected in N correspond to compound vertices in C_i below. Analoguously to the above set of variables V, there will be sets of variables X_i in every graph.

For the following definitions, there are two basic alternatives for representing what corresponds to the start symbol of CHOMSKY grammars: One may either choose a node or a graph. The choice of a graph would directly correspond to the underlying approach in [AJ94b], [AJ94a] and [JA95]. However, some reader may find it more intuitive to have a start object which is properly atomic. As this decision is not essential, we have chosen the syntactically simpler second alternative.

Definition 3

A two-level context-free graph design grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$ is given as follows:

1. $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$ is a set of so-called pin graphs.
2. $\mathcal{H} = \{\mathcal{G}_{m+1}, \ldots, \mathcal{G}_n\}$ is another set of pin graphs.
3. Every pin graph in $\mathcal{F}$ has the form $\mathcal{G}_i = [V_i, E_i, P_i^{in}, P_i^{out}, C_i, sub_i, X_i]$.
4. Every pin graph in $\mathcal{H}$ has the form $\mathcal{G}_i = [V_i, E_i, P_i^{in}, P_i^{out}]$.
5. Every pin graph $\mathcal{G}_i$ in $\mathcal{F} \cup \mathcal{H}$ with $i \in \{1, \ldots, n\}$ meets the following conditions:

   5.1 $\mathcal{G}'_i = [V_i, E_i]$ is a finite, directed, acyclic graph with the set V_i of vertices and the set E_i of edges.

   5.2 $P_i^{in} \cup P_i^{out}$ contains the so-called input pins and output pins of $\mathcal{G}_i$ defined as follows:

   5.3 $P_i^{in} = \{ v \mid v \in V_i \wedge \neg\exists u \in V_i \, ((u, v) \in E_i) \}$

   5.4 $P_i^{out} = \{ v \mid v \in V_i \wedge \neg\exists u \in V_i \, ((v, u) \in E_i) \}$
6. Every pin graph $\mathcal{G}_i$ in $\mathcal{F}$ with $i \in \{1, \ldots, m\}$ meets the following conditions:

   6.1 The vertices in $C_i \subseteq V_i$ are called compound.

   6.2 $sub_i : C_i \to 2^{\{1,\ldots,m\}} \setminus \emptyset$ defined the admissible substitutions of graphs $\mathcal{G}_j$ for compound vertices in C_i.

   6.3 The vertices in $X_i \subseteq V_i$ are called variables. For all $i \in \{1, \ldots, m\}$ it holds $C_i \cap X_i = \emptyset$.
7. $c_0 \in \bigcup_{i=1\ldots m} C_i$ is some start vertex. $\square$

Obviously, the specification of P_i^{in} and P_i^{out} is redundant. It is given exclusively for clarity.

The pin graphs in $\mathcal{H}$ can be understood as special cases of graphs in $\mathcal{F}$ with $C_i = \emptyset$ and $X_i = \emptyset$. The property $C_i = \emptyset$ implies that the mapping sub_i (understood as a relation) must be empty. Intuitively, the graphs in $\mathcal{F}$ are to be substituted one into another according to $\{sub_i\}_{i=1\ldots m}$, whereas the graphs in $\mathcal{H}$ are used for replacing the varaibles in $\{X_i\}_{i=1\ldots m}$.

For illustration, recall figure 3 above. It depicts 2 pin graphs (one occurs in 5 copies) of $\mathcal{H}$. The graph of figure 2 belonging to $\mathcal{F}$ contains variables in 6 places. A possible substitution yields the graph originally displayed in figure 1.

Decisions about the substitution mechanism, i.e. embedding, define the underlying formal semantics. As introduced before, there isa two-level mechanism. For understandability, it is split into the following concepts:

- a rewrite relation,
- a language of pattern graphs,
- graph substitution, and
- a language of ground graphs.

There are some useful notations. For example, there is some desire to avoid name conflicts during substitution. Therefore, the following concentions will be adopted:

- If c and d are names of vertices (perhaps, in different graphs), then $c.d$ is an acceptable name of a vertex as well.
- Similarly, this notation is applied to pairs of vertices: $c.(d_1, d_2) = (c.d_1, c.d_2)$
- If c is a vertex and C a set of vertices, then holds $c.C = \{c.d \mid d \in C\}$.

Names of vertices of the form "c.d" are said to be structured. Originally, there are no structured names.

Definition 4

Assume any two-level context-free graph design grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$. For any graph $\mathcal{G}_i \in \mathcal{F}$, any vertex $c \in C_i$ and any index $j \in sub_i(c)$, the *Substitution of* $\mathcal{G}_j$ *in* $\mathcal{G}_i$ *at position* $c \in C_i$ yields a graph $\mathcal{G}_i[c \hookleftarrow \mathcal{G}_j]$ where $\mathcal{G}_i[c \hookleftarrow \mathcal{G}_j] = [V, E, P^{in}, P^{out}, C, sub, X]$ is defined by:

1. $V \;=\; (\,V_i \setminus \{c\}\,) \cup c.V_j$
2. $E \;=\; (\,(E_i \cup c.E_j) \setminus (V_i \times \{c\} \cup \{c\} \times V_i)\,) \cup ((\{v \mid (v,c) \in E_i\} \times c.P_j^{in}) \cup (c.P_j^{out} \times \{v \mid (c,v) \in E_i\}))$

   3.1 $P^{in} \;=\; P_i^{in}$ if $c \notin P_i^{in}$

   3.2 $P^{in} \;=\; (\,P_i \setminus \{c\}\,) \;\cup\; c.P_j^{in}$ if $c \in P_i^{in}$

   4.1 $P^{out} \;=\; P_i^{out}$ if $c \notin P_i^{out}$

   4.2 $P^{out} \;=\; (\,P_i \setminus \{c\}\,) \;\cup\; c.P_j^{out}$ if $c \in P_i^{out}$

5. $C \;=\; (\,C_i \setminus \{c\}\,) \cup c.C_j$
6. $sub \;=\; sub_{i/C_i \setminus \{c\}} \;\cup\; c.sub_j$
7. $X \;=\; X_i \;\cup\; c.X_j$ $\square$

The resulting pin graph $\mathcal{G} = \mathcal{G}_i[c \hookleftarrow \mathcal{G}_j]$ is of the same structure as the graphs in $\mathcal{F}$. Therefore, the definition above applies to those generated host graphs $\mathcal{G}$, for arbitrary of their vertices c, and for all graphs $\mathcal{G}_j$ in $\mathcal{F}$ as well. This allows to write $\mathcal{G}[c \hookleftarrow \mathcal{G}_j]$ even if $\mathcal{G}$ is the result of any preceding substitution process.

For illustration, one may consider the graph shown by the following figure. For simplicity, the name of only one compound node is displayed. Furthermore, there are three nodes with the suffix letter x displayed. For the node c, there may be substituted several graphs according to the grammatical knowledge given. This figure together with the following one illustrate one rewrite step.

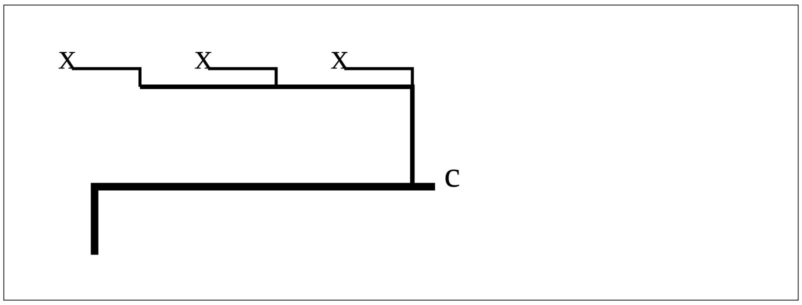


Figure 4: Graph before Substitution at Compound Node c

The result of substitution is displayed below.

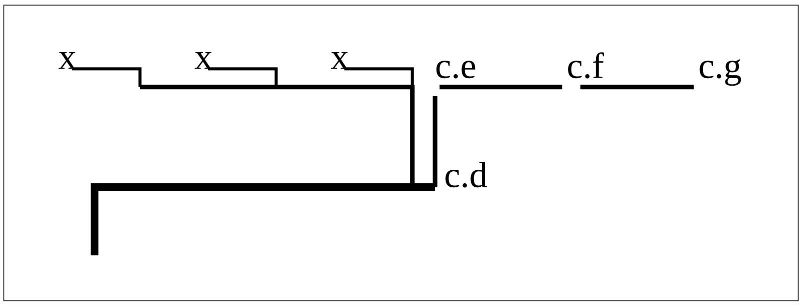


Figure 5: Graph after Substitution at Compound Node c

To indicate the possibilities of further steps of rewriting, the new nodes $c.d$, $c.e$, $c.f$, and $c.g$ resulting from the nodes d, e, f, and g, respectively, of the inserted graph are shown.

Definition 5

Among pin graphs of the structure $[V, E, P^{in}, P^{out}, C, sub]$, one defines a rewrite relation $\Rightarrow_{\mathcal{F}}$ by

1. $\mathcal{G} \;\Rightarrow_{\mathcal{F}}\; \mathcal{G}'$ if and only is $\exists c\, \exists j\; (\, \mathcal{G}' = \mathcal{G}[c \hookleftarrow \mathcal{G}_j]\,)$

One may interpret the vertex c_0 as a graph consisting of only the one (compound) vertex c_0 for justifying the notation $c_0 \;\Rightarrow_{\mathcal{F}}\; \mathcal{G}$.

2. $c_0 \;\Rightarrow_{\mathcal{F}}\; \mathcal{G}$ gilt if and only if
   $\exists i \in \{1, \ldots, m\}\; \exists j \in sub_i(c_0)\; (\, c_0 \in C_i \;\wedge\; \mathcal{G} = \mathcal{G}_j\,)$

$\Rightarrow_{\mathcal{F}}^{+}$ denotes the transitive closure of $\Rightarrow_{\mathcal{F}}$ and the irreducibility of any graph $\mathcal{G}$ with respect to $\Rightarrow_{\mathcal{F}}$ is indicated by $\mathcal{G}\!\downarrow_{\mathcal{F}}$. □

This is sufficient for defining the language of patterns generated by any given two-level context-free graph design grammar. The graph depicted in figure 2 is a typical element of such a language, for instance.

Definition 6

Assume any two-level context-free graph design grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$. This defines a language $\mathcal{P}(\mathcal{GDG})$ as follows:

- $\mathcal{P}(\mathcal{GDG}) \;=\; \{\, \mathcal{G} \mid c_0 \;\Rightarrow_{\mathcal{F}}^{+}\; \mathcal{G} \;\wedge\; \mathcal{G}\!\downarrow_{\mathcal{F}} \}$ □

Graphs belonging to $\mathcal{P}(\mathcal{GDG})$ describe the macro-structure of some design object under consideration. This macro-structure is, from a certain perspective, taken as a pattern for further refinement. In technical terms of the Fabel application domain, a fresh air supply net of order 6 is completed by adding the connections of order 8. This almost final design step follows a number of regularities. Thus, it may be reasonably understood as a substitution process similar to variable substitution in pattern languages (cf. [Ang80]). The following technical concepts seem quite appropriate.

Definition 7

Assume any grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$. The set of its variables is

1. $X_{\mathcal{GDG}} \ = \ \bigcup_{i=1\ldots m} X_i$

The final segment relation $\sqsupseteq$ of vertices is introduced as follows.

2. $\forall d, e \ (\ e \sqsupset_1 d \iff \exists c \ (\ e = c.d\ )\ )$
3. $\sqsupseteq_1$ denotes the reflexive closure of $\sqsupset_1$.
4. $\forall d, e \ (\ e \sqsupseteq d \iff e = d \ \vee \ \exists c \ (\ e \sqsupset_1 c \ \wedge \ c \sqsupseteq d\ )\ )$

ν computes the last letter of any vertex name via $\sqsupseteq$.

5. $\forall c, d \ (\ d = \nu(c) \iff c \sqsupseteq d \wedge \ \nexists e\ (\ d \sqsupset e\ )\ )$. $\square$

Structured vertices c which have been constructed during stepwise substitution are finally identified with its last letter $\nu(c)$. First, the substitution concept is introduced.

Definition 8

Assume any two-level context-free graph design grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$ with the language of patterns $\mathcal{P}(\mathcal{GDG})$. A substitution σ is some mapping

1. $\sigma \ : \ X_{\mathcal{GDG}} \ \longrightarrow \ \{m+1, \ldots, n\}$

Every substitution σ specifies a non-deterministic rewrite operator $\Rightarrow_\sigma$ on $\mathcal{P}(\mathcal{GDG})$.

2. $\mathcal{G} \Rightarrow_\sigma \mathcal{G}'$, if $\exists c\ (\, \mathcal{G}' = \mathcal{G}[c \hookleftarrow \mathcal{G}_{\sigma(\nu(c))}]\,)$

3. $\Rightarrow_\sigma^+$ denotes the transitive closure and $\Rightarrow_\sigma^*$ denotes the reflexive and transitive closure of $\Rightarrow_\sigma$.

4. $\mathcal{G}\!\downarrow_\sigma$ means that $\mathcal{G}$ is irreducible w.r.t. $\Rightarrow_\sigma$. □

The target concept to be developed in the present paper is the formal graph language generated by any given two-level context-free graph design grammar.

Definition 9

Assume any two-level context-free graph design grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$ with the language of patterns $\mathcal{P}(\mathcal{GDG})$. $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$ defines a formal language $\mathcal{L}(\mathcal{GDG})$ by

$$\mathcal{L}(\mathcal{GDG}) \;=\; \bigcup_{\substack{\mathcal{G} \in \mathcal{P}(\mathcal{GDG}) \\ \sigma \in X_{\mathcal{GDG}}}} \sigma(\mathcal{G})$$

□

The two levels of the derivation process are easy to see. For generating any graph of $\mathcal{L}(\mathcal{GDG})$, one first generates a pattern graph describing the skeleton of the target graph which belongs to $\mathcal{P}(\mathcal{GDG})$. Second, the final graph is constructed by an appropriate variable substitution.

This process suggests several similarity concepts. The simplest one is to consider two (or even more) graphs as somehow similar, exactly if they have the same skeleton. Another interesting approach is to choose the largest common subgraph of two skeletons as the similarity of two given graphs. A quite opposite approach compares the similarities of final substitutions.

Furthermore, the novel formalism developed may be taken as a firm basis for learning-theoretic investigations.

4 Learning Scenarios

This very brief chapter is aimed at presenting a few introductory remarks, only. The paper is intended as a launching pad for learning-theoretic investigations, but it is not intended to provide the learning theory itself.

A typical inductive inference scenario assumes any target grammar to be identified. Example graphs and, possibly, counter-examples are presented step by step. An inductive learning device has to identify the underlying grammar from suffiently many positive (and, possibly, negative) examples. In architectural design, it is quite unrealistic to assume sufficiently many negative examples. Because of GOLD's basic results about the limitations of learning from positive examples, only, this approach seems not sufficiently promising (cf. [Gol67]).

As a consequence, one may try to learn only one level of the grammar under the assumption of the other one. In case the CHOMSKY part is given, this is quite promising (cf. [Ang80]).

Another idea is case-based learning. One may try to learn similarity concepts for given two-level context-free graph design grammars. Recently, there have been developed similarity concept of graphs in architectural design, which are particularly taylored towards the needs of inductive learning (cf. [TSSM95]).

[GLK94] have developed an idea of evolutionary learning of design grammars. They are starting with some initially given grammar which is changed incrementally by some genetic algorithm. As we have a two-level approach invoked in the present paper, comparable idea of evolutionary learning may be even finer than in [GLK94]. A key idea is to generate "populations" of variable arrangements with respect to a given CHOMSKY part of a two-level design grammar.

References

[AJ94a] Oksana Arnold and Klaus P. Jantke. Therapy plan generation as program synthesis. In Setsuo Arikawa and Klaus P. Jantke, editors, *Algorithmic Learning Theory, AII'94 & ALT'94*, volume 872 of *LNAI*, pages 40–55. Springer–Verlag, 1994.

[AJ94b] Oksana Arnold and Klaus P. Jantke. Therapy plans as hierarchically structured graphs. WISCON Report 02/94, HTWK Leipzig (FH), Fachbereich IMN, April 1994.

[AJ94c] Oksana Arnold and Klaus P. Jantke. Therapy plans as hierarchically structured graphs. In *Fifth International Workshop on Graph Grammars and their Application to Computer Science, Williamsburg, Virginia, USA*, November 1994.

[Ang80] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.

[AS83] Dana Angluin and Carl H. Smith. A survey of inductive inference: Theory and methods. *Computing Surveys*, 15:237–269, 1983.

[BF95] Katy Börner and Roland Faßauer. Chapter 7: Analogical layout design (SYN). In Katy Börner, editor, *Modules for Design Support*, number 35 in FABEL Report, pages 59–68. GMD St. Augustin, 1995.

[Bör94] Katy Börner. Structural similarity as guidance in case-based reasoning. In Stefan Wess, Klaus-Dieter Althoff, and Michael M. Richter, editors, *1st European Workshop on Case-Based Reasoning (EWCBR'93), November 1-5, 1993, Selected Papers*, volume 837 of *Lecture Notes in Artificial Intelligence*, pages 197–208. Springer-Verlag, 1994.

[Bro94] Ken N. Brown. Shape grammars and semantics: Formalising the link between geometry, features and the design process. In *Third International Conference on Artificial Intelligence in Design, Lausanne, Switzerland, August 15–18, 1994, Workshop on Reasoning with Shapes in Design, Workshop Notes*, pages 1–4, 1994.

[EOSW94] Dieter Ehrenberg, Thomas J. Olbrich, Ralf Schulz, and Herbert Wiesner. Distributed reasoning support systems – a case-based approach to solving ill-structured decision problems. Arbeitsbericht 5, Universität Leipzig, Wirtschaftswissenschaftliche Fakultät, Lehrstuhl Wirtschaftsinformatik, May 1994.

[FC93] FABEL-Consortium. Survey of FABEL. FABEL–Report 2, Gesellschaft für Mathematik und Datenverarbeitung mbH, Forschungsbereich Künstliche Intelligenz, February 1993.

[FK93] Fabel-Konsortium. FABEL im Überblick. FABEL–Report 1, Gesellschaft für Mathematik und Datenverarbeitung mbH, Forschungsbereich Künstliche Intelligenz, February 1993.

[GLK94] John S. Gero, S.J. Louis, and S. Kundu. Evolutionary learning of novel grammars for design improvement. *AIEDAM*, 8(3):83–94, 1994.

[Gol67] E Mark Gold. Language identification in the limit. *Information and Control*, 14:447–474, 1967.

[Gor92] Dallie E. Gordon. Implication of cognitive theory for knowledge acquisition. In Robert R. Hoffman, editor, *The Psychology of Expertise. Cognitive Research of Empirical AI*, pages 99–120. Springer-Verlag, 1992.

[HSF+92] K. Hua, I. Smith, B. Faltings, S. Shih, and G. Schmitt. Adaptation of spatial design cases. In John S. Gero, editor, *Artificial Intelligence in Design'92*, pages 559–575. Kluwer Academic Publ., 1992.

[HU79] John E. Hopcroft and Jeffrey. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[Ita94] Alon Itai. Learning morphology – practice makes good. In Rafael C. Carrasco and Jose Oncina, editors, *Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, September 21–23, 1994*, volume 862 of *Lecture Notes in Artificial Intelligence*, pages 5–15. Springer–Verlag, 1994.

[JA95] Klaus P. Jantke and Oksana Arnold. Graphgrammatik-Konzepte in der Therapieplanung für komplexe dynamische Prozesse. In Markus Holzer, editor, *4. GI Theorietag "Automaten und Formale Sprachen", 1994, Proc., Herrsching*, pages 42–47. Universität Tübingen, Wilhelm-Schickard-Institut, 1995.

[Jan89] Klaus P. Jantke. Algorithmic learning from incomplete information: Principles and problems. In J. Dassow and J. Kelemen, editors, *Machines, Languages, and Complexity*, Lecture Notes in Computer Science, pages 188–207. Springer-Verlag, 1989.

[Jan92] Klaus P. Jantke. Case based learning in inductive inference. In *Proc. 5th Annual ACM Workshop on Computational Learning Theory, (COLT'92), July 27-29, 1992, Pittsburgh, PA, USA*, pages 218–223. ACM Press, 1992.

[Jan94] Klaus P. Jantke. Nonstandard concepts of similarity in case-based reasoning. In Hans-Hermann Bock, Wolfgang Lenski, and Michael M. Richter, editors, *Information Systems and Data Analysis: Prospects – Foundations – Applications, Proceedings of the 17th Annual Conference of the GfKl, Univ. of Kaiserslautern, 1993*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 28–43. Springer-Verlag, 1994.

[JL93] Klaus P. Jantke and Steffen Lange. Case–based representation and learning of pattern languages. In K.P. Jantke, S. Kobayashi, E. Tomita, and T. Yokomori, editors, *4th Intern. Workshop on Algorithmic Learning Theory (ALT'93), Proc., November 1993, Tokyo, Japan*, volume 744 of *Lecture Notes in Artificial Intelligence*, pages 87–100. Springer–Verlag, 1993.

[Kol92] Janet L. Kolodner. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6:3–34, 1992.

[Kol93] Janet L. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann, 1993.

[Kos83] K. Koskienniemi. Two-level morphology: A general computational model for word-form recognition and production. Technical report, Department of General Linguistics, University of Helsinki, Publication No. 11, 1983.

[KR90] Hans-Jörg Kreowski and Grzegorz Rozenberg. On structured graph grammars. I. *Information Sciences*, 52:185–210, 1990.

[KW80] Reinhard Klette and Rolf Wiehagen. Research in the theory of inductive inference by GDR mathematicians - a survey. *Information Sciences*, 22:149–169, 1980.

[Lak87] George Lakoff. *Women, Fire, and Dangerous Things*. The University of Chicago Press, 1987.

[Lee94] John R. Lee. The formal basis of "reasoning with shapes". In *Third International Conference on Artificial Intelligence in Design, Lausanne, Switzerland, August 15–18, 1994, Workshop on Reasoning with Shapes in Design, Workshop Notes*, pages 30–34, 1994.

[Mac91] C.A. Mackenzie. *Function and Structure Relationships and Transformations in Design Processes.* PhD thesis, University of Sydney, Department of Architectural and Design Science, Sydney, Australia, 1991.

[MG87] C.A. Mackenzie and John S. Gero. Learning design rules from decisions and performances. *Artificial Intelligence in Engineering*, 2(1):2–10, 1987.

[PAG92] David S. Prerau, Mark R. Adler, and Alan S. Gunderson. Eliciting and using experiential knowledge and general expertise. In Robert R. Hoffman, editor, *The Psychology of Expertise. Cognitive Research of Empirical AI*, pages 137–148. Springer-Verlag, 1992.

[RH92] Stephen B. Regoczei and Graeme Hirst. Knowledge and knowledge acquisition in the computational context. In Robert R. Hoffman, editor, *The Psychology of Expertise. Cognitive Research of Empirical AI*, pages vii–ix. Springer-Verlag, 1992.

[Ros75] Eleanor Rosch. Cognitive reference points. *Cognitive Psychology*, 7:531–547, 1975.

[RS89] Christopher K. Riesbeck and Roger C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Assoc., 1989.

[Sch95] Ralf Schulz. Handling similarity in case-based systems for ill-structured problems. Arbeitsbericht 7, Universität Leipzig, Wirtschaftswissenschaftliche Fakultät, Lehrstuhl Wirtschaftsinformatik, January 1995.

[SG78] G. Stiny and J. Gips. *Algorithmic Aesthetics: Computer Models for Criticism and Design in the Arts.* University of California Press, 1978.

[Shi94] Shen-Guan Shih. *The Use of String Grammars in Architectural Design.* PhD thesis, ETH Zürich, Department of Architecture, Switzerland, 1994.

[SJL94] Yasubumi Sakakibara, Klaus P. Jantke, and Steffen Lange. Learning languages by collecting cases and tuning parameters. In Setsuo Arikawa and Klaus P. Jantke, editors, *Algorithmic Learning Theory, AII'94 & ALT'94*, volume 872 of *LNAI*, pages 533–547. Springer-Verlag, 1994.

[TSSM95] Elisabeth-Ch. Tammer, Kathleen Steinhöfel, Siegfried Schönherr, and Daniel Matuschek. Anwendung des Konzeptes der Strukturellen Ähnlichkeit zum Fallvergleich mittels Term- und Graph-Repräsentationen. FABEL–Report 38, Gesellschaft für Mathematik und Datenverarbeitung mbH, Forschungsbereich Künstliche Intelligenz, September 1995.

[Vid94] Enrique Vidal. Grammatical inference: An introductory survey. In Rafael C. Carrasco and Jose Oncina, editors, *Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, September 21–23, 1994*, volume 862 of *Lecture Notes in Artificial Intelligence*, pages 1–4. Springer–Verlag, 1994.

Properties of Formal Languages of Therapy Plans created by Graph Grammars (overview)

Daniel Kirsten

Hochschule für Technik, Wirtschaft und Kultur Leipzig
Fachbereich IMN
Postfach 30066, 04251 Leipzig

kirsten@imn.th-leipzig.de

1 Motivation

The paper deals with a special type of graph grammars used in therapy planning in complex dynamic systems in the Joint Project (BMFT-Verbundprojekt) WISCON. The word therapy is often related with medicine, but we concern with plans which allow to avoid averages in complex dynamic systems like industrial equipments.
Plans are represented as finite, directed, acyclic graphs, called pin graphs. The semantic of a vertex is an action, the semantic of an edge is the time relation later than or simultaneous.
A planner (a program) has to work on plans: creation, evaluation, simulation, revision... That means the planner has to create and to evaluate pin graphs.
Therefore Dipl.-Ing. Oksana Arnold has developed a graph grammar concept suitable for therapy planning. The core is the *rooted family*

[AJ94a][AJ94b].
It is based on vertex rewriting, i.e. single vertices are replaced by complex graphs like deriving in context-free languages.
The basic idea of planning is the following: The planner starts at a pin graph called starting graph. It contains two kinds of vertices: elementary vertices and compound vertices like terminals and nonterminals in CHOMSKY-grammars. The planner replaces compound vertices by complex pin graphs which may also contain compound vertices. Usually there are several pin graphs which can replace a compound vertex. The planner inserts and evaluates pin graphs. So the search space of the planner is a set of pin graphs created by a graph grammar, or better it is a language of graphs.
The usage of graph grammars leads also to problems. Is the rooted family really suitable for therapy planning? May be there is an optimal plan to solve a problem, but the graph grammar is only able to create worse plans. May be there are several plans solving a problem, but the graph grammar is unable to create one of them. We deal therapy planning, so human life may be dependant on a graph grammar.
Therefore a basic research on languages creatable by rooted families was necessary! We call these languages context-free. May be this is a bad choose because the term context-free is already defined in other graph grammar concepts. Otherwise it reflects the intution of this concept.

- What is the *power* of context-fee languages? I.e. are all recursively enumerable languages context-free, or only special cases?

- Are there interesting languages which are not context-free?

- Are there plans which can not be created by a rooted family?

- Are context-free languages stable under complement, intersection and union?

- How can I proof that a language is not context-free?

2 Introduction

A short introduction to the basic concepts. Actually I revise the entire concept. Several formalisations hinder the investigations of a theorist. I only revise the syntax, I do not revise the semantic. Unfortunately this revision is not ready yet, so I give the version from [AJ94a][AJ94b]. This version was the base of my investigation.

Definition 1.
A *hierarchically structured family of plans* is a finite set of pin graphs where: $\mathcal{G}_i = [V_i, E_i, P_i^{in}, P_i^{out}, C_i, sub_i]\,(i = 1, \ldots, k)$, such that for all $i \in \{1, \ldots, k\}$:

1. $[V_i, E_i]$ is a nonempty, finite, directed, acyclic graph.
2. $P_i^{in} := \{v | v \in V_i \quad \wedge \quad \neg\exists u \in V_i((u, v) \in E_i)\}$
   $P_i^{out} := \{v | v \in V_i \quad \wedge \quad \neg\exists u \in V_i((v, u) \in E_i)\}$
3. $C_i \subseteq V_i$
4. $sub_i : C_i \rightarrow (2^{\{1 \ldots k\}} \setminus \{\emptyset\})$ $\square$

sub_i is called substitution mapping. It yields a nonempty, finite set of of natural numbers for each compound vertex c, determining the pin graphs which we are allowed to substitute on c. The substitution is defined in the following way. Now we have not to worry about sub_i, we deal this later.

Substitutions:
given: $\mathcal{F} := \{\mathcal{G}_1 \ldots \mathcal{G}_k\}$
$\mathcal{G} := \mathcal{G}_i[c \hookleftarrow \mathcal{G}_j]$
Presumption: $c \in V_i$
Definition of $\mathcal{G}$:

1. $V := (V_i \setminus \{c\}) \cup c.V_j$
2. $$\begin{aligned} E :=&((E_i \cup c.E_j) \setminus (V_i \times \{c\}) \setminus (\{c\} \times V_j) \\ &\cup(\{v | (v, c) \in E_i\} \times c.P_j^{in}) \\ &\cup(c.P_j^{out} \times \{v | (c, v) \in E_i\}) \end{aligned}$$

3. P^{in} , P^{out} already defined.
4. $C := (C_i \setminus \{c\}) \cup c.C_j$
5. $sub := sub_{j/C_i \setminus \{c\}} \cup c.sub_j$

We assume that the vertices of pin graphs are words over an alphabet! $c.V$ simply means we set a c in front of each vertex in V. This is necessary to ensure disjoint sets of vertices. So the vertices grow to words. This allows to backtrace the substitution process.
We forbid that a vertex of a pin graph is an initial segment of another vertex in the same pin graph. This property inherits in substitutions.

We may denote several substitutions (paths) by
$\mathcal{G}[c_1 \hookleftarrow \mathcal{G}_3][c_2 \hookleftarrow \mathcal{G}_6][c_1 c_3 \hookleftarrow \mathcal{G}_{42}]$

Theorem 1.
Substitution steps are commutativ and associativ in the following way:

- $\forall \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3 : \forall c, d \in V_1 :$
  $c \neq d \rightarrow \mathcal{G}_1[c \hookleftarrow \mathcal{G}_2][d \hookleftarrow \mathcal{G}_3] = \mathcal{G}_1[d \hookleftarrow \mathcal{G}_3][c \hookleftarrow \mathcal{G}_2]$
- $\forall \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3 : \forall c \in V_1 : \forall d \in V_2 :$
  $\mathcal{G}_1[c \hookleftarrow \mathcal{G}_2][cd \hookleftarrow \mathcal{G}_3] = \mathcal{G}_1[c \hookleftarrow \mathcal{G}_2[d \hookleftarrow \mathcal{G}_3]]$ □

Remind the quantifications of c and d. You can not perform the second step first if the first step creates the vertex which the second one overwrites. But associativity needs this fact.

The deriving relation is defined based on substitution. We have to use sub_i to ensure that the substitution is allowed.

$\mathcal{G}_i \Rightarrow_{\mathcal{F}} \mathcal{G} \quad \Longleftrightarrow \quad \exists c \in \mathcal{G}_i \; \exists j \in sub_i(c) \; : \; \mathcal{G} = \mathcal{G}_i[c \hookleftarrow \mathcal{G}_j]$

The transitive closure of $\Rightarrow_{\mathcal{F}}$ is $\Rightarrow_{\mathcal{F}}^{+}$.

Definition 2.
A *rooted family* is a pair $[\mathcal{F}, \mathcal{G}_i]$ such that $\mathcal{F}$ is a *hierarchically structured family of plans* and $\mathcal{G}_i$ is a pin graph in $\mathcal{F}$. $\mathcal{G}_i$ is the starting pin graph.

The *language of a rooted family* is defined by:
$L(\mathcal{R}) := \{\mathcal{G} \mid \mathcal{G}_i \Rightarrow^+_{\mathcal{F}} \mathcal{G} \quad \wedge \quad C = \emptyset\,\}$

These definitions look quite useful as long as you do not have to use it. Main lacks I plan to correct are:

- The term *pin graph* is defined implicitely. Imagine a pin graph such that its substitution mapping (sub) delivers indices 3, 4 and 9. It is not a pin graph if it is in a *hierarchically structured family of plans* with lesser than nine pin graphs, otherwise it is a pin graph.
- The usage of natural numbers as indices renders proofs more difficult, i.e. the ideas are difficult to denote. Often we have to shift indices to ensure that we use all numbers from 1 to k. This causes repairs of the substitution mapping.
- Denoting P_i^{in} and P_i^{out} is redundant, because they are determined by V_i and E_i.
- Defining compound vertices in a pin graph is not useful. A better solution is to let the grammar decide which vertices are compound vertices.

Remember that the vertices of the created pin graphs are words: The renaming from V to $c.V$ produces longer and longer vertices. So vertices consist of several former compound vertices followed by an elementary vertex. The main point for the planning domain are the last letters. E.g. the semantic of the vertices $c_2c_4c_2a$, c_9c_1a and c_8a is the same, it is the semantic of a. You may think the best solution is to remove the substitution history, simply renaming $c_2c_4c_2a$ to a, but this yields to another problem. Vertices with the same semantic will be renamed to one vertex! So we have to live with the problem that the one plan can be represented by different graphs, there are infinite many graphs representing a given plan. These pin graphs are called equivalent. ($last(v)$ is simply the last letter of v.)

Definition 3.
The pin graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent iff there is a bijection

$r : V_1 \rightarrow V_2$ such that :

- $\forall v \in V_1 : last(v) = last(r(v))$
- $\forall (u,v) \in (V_1 \times V_1) : (u,v) \in E_1 \longleftrightarrow (r(u), r(v)) \in E_2$

We denote it by $\mathcal{G}_1 \cong \mathcal{G}_2$.

Equivalence is nothing but equality on the semantic level. Equivalence of pin graphs reflects the intuitive meaning of equality of plans. Equivalence is necessary, but it draws us into a tide of problems. A tasting (You find formal definitions in my paper.) :

- We can not get along with the semantic of $\in$. We have to use $\underset{\sim}{\in}$ (equivalent in) instead of $\in$. $\mathcal{G} \underset{\sim}{\in} L$ means L contains a pin graph which is equivalent to $\mathcal{G}$.
- So we have to define new relations for subset: It is $L_1 \underset{\approx}{\subseteq} L_2$ iff all pin graphs in L_1 are equivalent in L_2 . $\underset{\approx}{\subsetneq}$ is understood accordingly.
- Two languages are equivalent ($L_1 \cong L_2$) iff each language is an equivalent subset of the other language.
- $L_1 \underset{\sim}{\setminus} L_2$ means we remove all pin graphs from L_1 which are equivalent in L_2.
- The most interesting problem occurs if we discuss the intersection of languages. I call it equivalent intersection. Assume two languages L_1 and L_2. Assume a pin graph $\mathcal{G}$ such that $\mathcal{G} \in L_1$. Further is $\mathcal{G} \underset{\sim}{\in} L_2$, but $\mathcal{G} \notin L_2$, i.e. it is an eqvivalent version of $\mathcal{G}$ in L_2, but $\mathcal{G}$ misses in L_2. It is quite clear that $\mathcal{G} \underset{\sim}{\in} L_1 \underset{\sim}{\cap} L_2$, but shall we put $\mathcal{G}$ or its equivalent copy from L_2 into $L_1 \underset{\sim}{\cap} L_2$. So I define $\underset{\sim}{\cap}$ in that way: $L_1 \underset{\sim}{\cap} L_2 := (L_1 \underset{\sim}{\setminus} L_2) \cup (L_2 \underset{\sim}{\setminus} L_1)$

Another problem caused by equivalence is that there are infinite languages which contain only equivalent copies of a finite number of pin graphs. An infinite language which can be reduced to a finite language by removing equivalent copies is called weakly infinite.

Definition 4.
A language L is called *strongly infinite* iff
$\nexists \hat{L}$: $\hat{L}$ is finite $\wedge\ \hat{L} \cong L$ $\square$

Now we are able to define the family of context-free languages:

Definition 5. $\mathcal{PG}_{cf} := \{\ L \mid \exists \mathcal{R}\ :\ L(\mathcal{R}) \cong L\ \}$

The set of all pin graphs over a nonempty finite alphabet Σ is defined as follows:

Definition 6.
$\mathcal{PG}^{+}(\Sigma) := \{\mathcal{G} \mid \mathcal{G}$ is a pin graph $\wedge\ C = \emptyset\ \wedge\ \forall v \in V : last(v) \in \Sigma\}$ $\square$

3 A Pumping Theorem

Context-free languages allow pumping in the following way:
c^1 and c^2 are c with upper indices, so $c^2 \neq cc$!

Theorem 2.
$\forall L \in \mathcal{PG}_{cf}$: $(L$ is strongly infinite$) \longrightarrow$

$$
\begin{aligned}
&\exists n \in \mathbb{N} \quad \forall \mathcal{G} \lessapprox L : \quad |\mathcal{G}| \geq n \rightarrow \\
&\quad \exists \mathcal{G}^1, \mathcal{G}^2, \mathcal{G}^3 : \quad && C^1 = \{c^1\}, |V^1| > 1, \\
& && C^2 = \{c^2\}, |V^2| > 1, \\
& && C^3 = \emptyset, \\
& && \mathcal{G} \cong \mathcal{G}^1[c^1 \hookleftarrow \mathcal{G}^2][c^1 c^2 \hookleftarrow \mathcal{G}^3], \\
& && \forall j \in \mathbb{N} : \mathcal{G}^1[c^1 \hookleftarrow \mathcal{G}^2]^{j,c2}[c^1 (c^2)^j \hookleftarrow \mathcal{G}^3] \lessapprox L
\end{aligned}
$$
$\square$

We can not simply pump like $[c^1c^2 \hookleftarrow \mathcal{G}^2][c^1c^2 \hookleftarrow \mathcal{G}^2][c^1c^2 \hookleftarrow \mathcal{G}^2]\ldots$, because the vertex c^1c^2 will be overwritten to $c^1c^2c^2$ and so on...
We have to pump like $[c^1c^2 \hookleftarrow \mathcal{G}^2][c^1c^2c^2 \hookleftarrow \mathcal{G}^2][c^1c^2c^2c^2 \hookleftarrow \mathcal{G}^2]$.

It is possible to strengthen up this pumping theorem. You may demand any size limit m such that each of the pin graphs $\mathcal{G}^1$, $\mathcal{G}^2$ and

$\mathcal{G}^3$ has at least m vertices, but n is depending on m.
I performed another improvement that allows deducting properties of all rooted families creating L. I combined these two ideas and more explainations in my paper, look there for more information.

Usually the least powerful version of the pumping theorem is strong enough. Often it is not necessary to use the last property (the pumping) to proof that a language is not context-free. This seems to be stupid, using a pumping theorem whitout pumping. Just hide the last line of the pumping theorem and review it. The problem is that many pin graphs are only derivable via $\mathcal{G}^1[c^1 \hookleftarrow \mathcal{G}^2]$ if one of the pin graphs $\mathcal{G}^1$ and $\mathcal{G}^2$ has only one vertex. So many pin graphs are not derivable as the pumping theorem demands.

4 Other main Results

Often examined graphs are the complete graphs. They contain cycles, so they are not pin graphs. But we can define the complete directed pin graphs $\mathcal{G}^x_{cd}$ for each natural number x greater than 0 as follows:

Vertices: $V^x_{cd} := \{c_i a| \quad i := 1, \ldots, x\}$

Edges: $E^x_{cd} := \{(c_i a, c_j a)| \quad i, j := 1, \ldots, x \quad \wedge \quad i < j\}$

Between two different vertices is an edge starting from the vertex with the lesser index. There are not any compound vertices, the input- and outputvertices are clear. L_{cd} is the language of all complete, directed pin graphs.

$L_{cd} = \{\mathcal{G}^x_{cd} \mid x \in I\!N^+\}$

This language is not context-free. It is even worse: Each context-free language contains only a finite count of pin graphs from L_{cd} or it contains infinite many pin graphs from L_{cd} which are only equivalent copies of finite many pin graphs from L_{cd}.

Theorem 3.
$\nexists L \in PG_{cf} \ : \quad (L_{cd} \tilde{\cap} L)$ is strongly infinite $\square$

This theorem can be proved using the pumping theorem whitout pumping as I mentioned above. If we assume that a context-free language L containes infinite many complete directed pin graphs we are unable to show that pumping will exceed L. But we can use the pre-last line of the pumping theorem and show that the complete directed pin graphs are not derivable.
The main lack of the rooted family is the precise rule of drawing new edges in the definition of the substitution.
One conclusion of theorem 3 is that the language of all pin graphs over an alphabet is not context-free.

Theorem 4.
$\forall\, \Sigma : \qquad \mathcal{PG}^+(\Sigma) \notin \mathcal{PG}_{cf}$ $\square$

Such is life.

Context-free languages are closed under union. The idea of the proof is easy: "Merge" the rooted families, create a new single-vertex starting graph from which you can derive the two "old" starting graphs. You get a good taste of the suitability of the definitions from the beginning. To merge the rooted families the pin graphs of one rooted family have to get new indices. So the substitution mappings have to be repaired... Stability under union and theorem 4 show that the complement of a context-free language can not be context-free. However, there are languages which are not context-free and their complements are not context-free, too. To show this, you may use "the half of" L_{cd} and theorem 3. Context-free languages are not stable under intersection. An easy example is the standard example from the CHOMSKY-hierarchy, $a^n b^n c^n$.

Theorem 5.

- $\forall L_1, L_2 \in \mathcal{PG}_{cf} \;:\qquad (L_1 \cup L_2) \in \mathcal{PG}_{cf}$
- $\forall L \subsetneqq \mathcal{PG}^+(\Sigma) \;:\quad L$ is finite $\rightarrow L \in \mathcal{PG}_{cf}$
- $\forall L \subsetneqq \mathcal{PG}^+(\Sigma) \;:\quad L$ is weakly infinite $\rightarrow L \in \mathcal{PG}_{cf}$

- $\forall L \in \mathcal{PG}_{cf} \ : \quad (\mathcal{PG}^{+}(\Sigma) \mathbin{\not\curlywedge} L) \notin \mathcal{PG}_{cf}$
- $\exists L \subseteq \mathcal{PG}^{+}(\Sigma) \ : \quad L \notin \mathcal{PG}_{cf} \quad \wedge \quad (\mathcal{PG}^{+}(\Sigma) \mathbin{\not\curlywedge} L) \notin \mathcal{PG}_{cf}$
- $\exists \bar{L}, \hat{L} \in \mathcal{PG}_{cf} \ : \quad (\bar{L} \mathbin{\tilde{\cup}} \hat{L}) \notin \mathcal{PG}_{cf}$ $\square$

5 Conclusions and Next Steps

The main questions are answered. But I am not able to evaluate the suitability of the rooted family for therapy planning. I figured out and proofed several properties which allow people working in therapy planning a better understanding of the graph grammar concept they created themselves. Theorem 3 looks terrible to theorists, may be practists smile about it, because they hate complete directed graphs.

If you examine the complete, directed pin graphs you may recognize that many edges are not necessary. Remember that the semantic of an edge is a transitive time relation. An edge (a, b) is called redundant iff there is also a chain of edges from a to b. The complete, directed pin graphs are the root of all problems, but the problems of the complete, directed pin graph are the redundant edges.
The practicioners need not any redundant edge, although they defined pin graphs in a way that redundant edges are allowed. So the new question is: Can we avoid the main problems if we confine to pin graphs whitout redundant edges.
My paper "Redundant Edges in Therapy Planning" is nearly ready. The main problems of the rooted family persist.
Interesting problems to investigate are the following:

- The concept of the rooted family has been extended such that vertices carry logical constraints which are inherited by substitutions. The constraints allow a shrinking of the substitution mapping...
- How important is connectivity? I believe to know that it is useful to allow disconnected graphs in rooted families, even if you wish to create only connected graphs.

- (How) can we decide membership, finiteness...

* * * *

References

[AJ94a] Oksana Arnold and Klaus P. Jantke. Therapy plan generation as program synthesis. In Setsuo Arikawa and Klaus P. Jantke, editors, *LNAI*, volume 872, pages 40–55. Springer-Verlag, 1994.

[AJ94b] Oksana Arnold and Klaus P. Jantke. Therapy plans as hierarchically structured graphs. WISCON Report 02/94, HTWK Leipzig (FH), Fachbereich IMN, April 1994.

[AJ95] Oksana Arnold and Klaus P. Jantke. Inductive program synthesis for therapy plan generation. CALG Report 01/95, HTWK Leipzig (FH), Fachbereich IMN, July 1995.

[HU79] John E. Hopcroft and Jeffrey D. Ullmann. *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie.* ADDISON-WESLEY, 1979.

A Pumping Lemma for Output Languages of Attributed Tree Transducers

Armin Kühnemann and Heiko Vogler

Institut für Softwaretechnik I
Technische Universität Dresden
D–01062 Dresden

{kuehne,vogler}@orchid.inf.tu-dresden.de

Abstract

An attributed tree transducer is a formal model for studying properties of attribute grammars. We introduce a pumping lemma for output languages of noncircular, producing, and visiting attributed tree transducers. The pumping lemma can be applied to gain two results: (1) there is no noncircular, producing, and visiting attributed tree transducer which computes the set of all monadic trees with exponential height as output and (2) there is a hierarchy of noncircular, producing, and visiting attributed tree transducers with respect to their number of attributes.

Keywords: Attribute Grammars, Tree Languages, Pumping Lemma

1 Introduction

First Aho and Ullman have inspected pumping lemmata for output languages of translation schemes in [1], namely for generalized syntax directed translations. Perrault and Ésik have introduced in [12] and [7], respectively, pumping lemmata for (nondeterministic) top–down tree transducers (cf. [13, 14, 3]). Engelfriet, Rozenberg, and Slutzki have presented in [5] a pumping lemma for output languages of deterministic top–down tree–to–string transducers which has a structure that is closely related to the pumping lemma for context–free string languages. The proof of this lemma had a big influence on the development of the pumping lemma for output languages of attributed tree transducers, which was introduced in [11]. Recently, Kühnemann has presented in [10] a pumping lemma for output languages of macro tree transducers (cf. [4, 2, 6]).

Attributed tree transducers have been investigated by Fülöp in [8]. They are abstractions of attribute grammars (cf. [9]) in the sense that they take trees over an arbitrary ranked alphabet of input symbols rather than derivation trees as argument, and that the values of the attributes are also trees over a ranked alphabet of output symbols.

Like in attribute grammars, the set of attributes is partitioned into the set of synthesized and inherited attributes which are associated to the input symbols and which compute their values in a bottom–up manner and in a top–down manner, respectively. In contrast to attribute grammars, to every input symbol the whole set of attributes is associated. Roughly speaking, computing the value of a synthesized attribute occurrence of a node x of an input tree, the values of the inherited attribute occurrences of x and of the synthesized attribute occurrences of its sons (if they exist) may be used and, computing the value of an inherited attribute occurrence of x, the values of the inherited attribute occurrences of its father (if it exists) and of the synthesized attribute occurrences of x and of its brothers may be used.

We consider only total deterministic attributed tree transducers: For every node x of an input tree which is labeled by a particular input symbol and for every synthesized attribute s, the computation of the attribute occurrence of s at x is fixed by exactly one rule. Similarly, for every node x which is labeled by a particular input symbol and for every inherited attribute i, the computation of the attribute occurrence of i at the j–th son of x is fixed by exactly one rule.

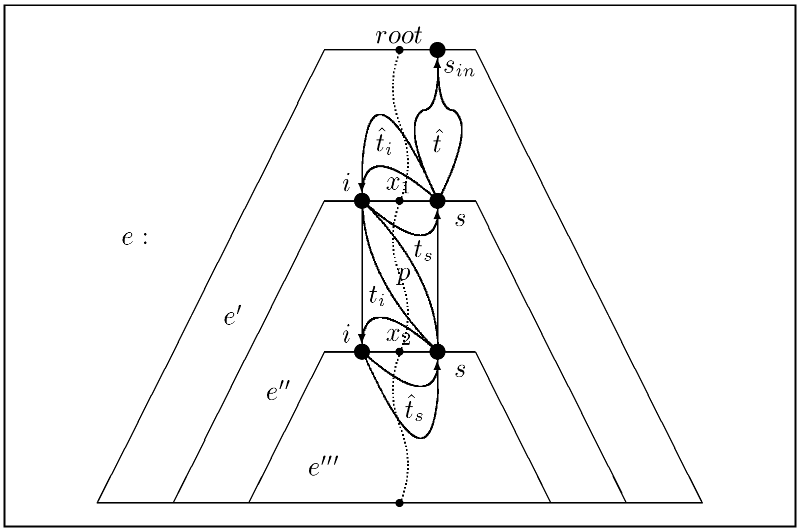
As in attribute grammars, these dependencies can induce circularities among the attribute occurrences. We restrict the attributed tree transducers to be noncircular and we designate a synthesized attribute s_{in} as initial attribute. Thus we designate an initial attribute occurrence at the root of every input tree of which the value will be the output tree. Then every attributed tree transducer M computes a total function from input trees to output trees. The output language of an attributed tree transducer M is defined as the range of this function.

Additionally, we restrict our pumping lemma to producing and visiting attributed tree transducers. An attributed tree transducer is producing, if every rule application delivers at least one new output symbol. An attributed tree transducer is visiting, if for every input tree and for every node x of it, the value of at least one attribute occurrence of x is needed to compute the value of s_{in} at the root.

The main idea of our pumping lemma is adopted from the pumping lemma for context–free string languages. We consider input trees belonging to a sufficiently large output tree to obtain new pumped output trees: For every producing and visiting attributed tree transducer M, a natural number n_M, called the pumping index of M, can be constructed. If we choose an output tree t from the output language of M which has at least n_M nodes, then every input tree e which can be transformed into t has the following property: e is high enough, such that it has a path p, on which two different nodes x_1 and x_2 can be found, which have the same attribute–set, i.e. the same set of attributes, for which the attribute occurrence of x_1 (and of x_2, respectively) is needed to calculate s_{in} at the root of e. Assuming that x_1 is closer to the root than x_2, we can define the following decomposition of e into input patterns. Roughly speaking,

- e' is the tree e without the subtree which has x_1 as root.
- e'' is the tree which has x_1 as root without the subtree which has x_2 as root.

- e''' is the tree which has x_2 as root.



This decomposition of e induces a decomposition of the output tree t into output patterns. Roughly speaking,

- The pattern $\hat{t}$ corresponds to the normal form of the attribute occurrence s_{in} at the root of e, which is calculated only on the nodes of e'.
- For every synthesized attribute s in the attribute–set of the two nodes x_1 and x_2, the pattern t_s (and $\hat{t}_s$) corresponds to the normal form of the attribute occurrence s at x_1 (and at x_2, respectively), which is calculated only on the nodes of e'' (and e''', respectively).
- For every inherited attribute i in the attribute–set of the two nodes x_2 and x_1, the pattern t_i (and $\hat{t}_i$) corresponds to the normal form of the attribute occurrence i at x_2 (and at x_1, respectively), which is calculated only on the nodes of e'' (and e', respectively).

Since the two nodes x_1 and x_2 are compatible with respect to their attribute–sets, we can construct new input trees by repeating e'' arbitrarily many times. The output patterns t_s and t_i must be used for every repetition of e'' to obtain the corresponding new output trees. Thus the pumping process itself can be described by using only the output patterns.

References

[1] A.V. Aho and J.D. Ullman. Translations on a context free grammar. *Inform. and Control*, 19:439–475, 1971.

[2] B. Courcelle and P. Franchi–Zannettacci. Attribute grammars and recursive program schemes. *Theoret. Comput. Sci.*, 17:163–191 and 235–257, 1982.

[3] J. Engelfriet. Bottom–up and top–down tree transformations — a comparison. *Math. Syst. Theory*, 9:198–231, 1975.

[4] J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R.V. Book, editor, *Formal language theory; perspectives and open problems.* New York, Academic Press, 1980.

[5] J. Engelfriet, G. Rozenberg, and G. Slutzki. Tree transducers, L systems, and two–way machines. *J. Comput. Syst. Sci.*, 20:150–202, 1980.

[6] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31:71–145, 1985.

[7] Z. Ésik. Decidability results concerning tree transducers. *Acta Cybernetica*, 5:1–20, 1980.

[8] Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.

[9] D.E. Knuth. Semantics of context–free languages. *Math. Syst. Theory*, 2:127–145, 1968.

[10] A. Kühnemann. A pumping lemma for output languages of macro tree transducers. Technical Report TUD/FI95/08, Technical University of Dresden, 1995.

[11] A. Kühnemann and H. Vogler. A pumping lemma for output languages of attributed tree transducers. *Acta Cybernetica*, 11:261–305, 1994.

[12] C.R. Perrault. Intercalation lemmas for tree transducer languages. *J. Comput. Syst. Sci.*, 13:246–277, 1976.

[13] W.C. Rounds. Mappings and grammars on trees. *Math. Syst. Theory*, 4:257–287, 1970.

[14] J.W. Thatcher. Generalized^2 sequential machine maps. *J. Comput. Syst. Sci.*, 4:339–367, 1970.

On language families of interacting automata

Martin Kutrib

AG Informatik
Universität Gießen
Arndtstr. 2, D–35392 Giessen

kutrib@informatik.uni-giessen.de

Abstract

The language recognition capabilities of some classes of interacting automata are studied. In particular these are the classes collectively called polyautomata, their iterative variants and stack augmented generalizations. We summarize known relationships and show new results concerning languages recognizable by iterative pushdown arrays.

1 Introduction

The language recognition capabilities of some classes of interacting automata are studied. Classical one-dimensional *cellular spaces* and various modifications and generalizations have been investigated for a long time (e.g. [1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 14]). We restrict our considerations on some of such classes Smith [13] collectively called *polyautomata*, their iterative variants and a real generalization obtained by stack augmentation. Since there are such open problems in the theory of polyautomata as the question whether linear-time cellular automata are more powerful than real-time cellular automata or not, we designate real-time acceptors and acceptors without any time restrictions.

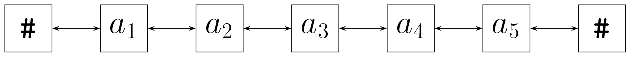


Figure 1: A cellular automaton.

Linear arrays of automata can be understood as models for massively parallel computers. Mainly various types differ in how the automata are interconnected and in how the input is supplied. Here we are investigating arrays with a very simple interconnection pattern. Each node is connected to its (one or two) immediate neighbors only. They are usually called *cellular automata* (CA) if the input is supplied in parallel and *iterative arrays* (IA) in case of sequential input to a designated single automaton. We will use the notion *one-way* (OCA) if the single automata (cells) are connected to their left immediate neighbor only. Moreover, our cells must not be finite-state machines. We study linear arrays of deterministic pushdown automata, too. Accordingly these models are called *pushdown cellular automata* (PDCA) resp. *iterative pushdown arrays* (IPDA). On a first glance they are more complex than their "classical" counterpart. On the other hand if computational universality should be obtained in the classical case we have to consider infinite arrays, sometimes called unbounded cellular automata, whereas in case of pushdown cellular automata arrays of at least length two are sufficient.

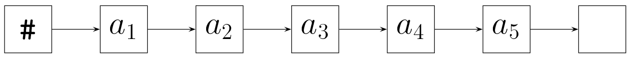


Figure 2: An one-way cellular automaton.

Instead of defining all the models formally we give a formal definition of the iterative pushdown arrays, from which the basic notions and functionality follows. Readers who are interested on some more definitions are suggested to [8, 14].

Definition 1.1 An *iterative pushdown array* (IPDA) is a system $(S, \Sigma, \Gamma, \sigma_0, \sigma, q_0, a_0, g_0,)$, where

a) S is the finite, nonempty set of *states*,
b) Σ is the finite, nonempty set of *input symbols*,
c) Γ is the finite, nonempty set of *stack symbols*,
d) $a_0 \in \Sigma$ is the *end of input symbol*,
e) $g_0 \in \Gamma$ is the *bottom of stack symbol*,
f) $\sigma_0 : S^3 \times \Gamma \times \Sigma \longrightarrow S \times \Gamma^*$ and $\sigma : S^3 \times \Gamma \longrightarrow S \times \Gamma^*$ are the *local transformations* of the cell at the origin and of all other cells respectively, satisfying $\forall s_1, s_2, s_3, s_4 \in S, g \in \Gamma, a \in \Sigma :\ \sigma(s_1, s_2, s_3, g) = (s_4, \gamma) \Longrightarrow$ (resp. $\sigma_0(s_1, s_2, s_3, g, a) = (s_4, \gamma) \Longrightarrow$) $(\gamma \in (\Gamma \setminus \{g_0\})^* \wedge g \neq g_0) \vee (\gamma = \gamma' g_0 \wedge \gamma' \in (\Gamma \setminus \{g_0\})^* \wedge g = g_0)$
g) $q_0 \in S$ is the *quiescent state*, such that $\forall g \in \Gamma : \sigma(q_0, q_0, q_0, g) = (q_0, g)$ holds.

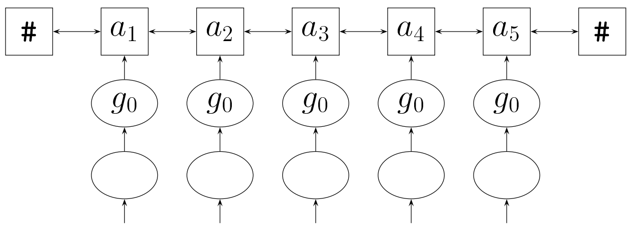


Figure 3: A pushdown cellular automaton.

The local transition functions induce a mapping (*global transformation*) $\mathcal{T} : (S \times \Gamma^+)^+ \times \Sigma^\omega \longrightarrow (S \times \Gamma^+)^+ \times \Sigma^\omega$ according to the following.

Let $\bar{\sigma} : S^3 \times \Gamma^+ \longrightarrow S \times \Gamma^+$ be defined as
$\bar{\sigma}(s_1, s_2, s_3, g_m \cdots g_0) :=$

$$\Big(\pi_1(\sigma(s_1, s_2, s_3, g_m)), \pi_2(\sigma(s_1, s_2, s_3, g_m)) g_{m-1} \cdots g_0\Big),$$

and $\bar{\sigma}_0 : S^3 \times \Gamma^+ \times \Sigma \longrightarrow S \times \Gamma^+$ be defined as

$\bar{\sigma}_0(s_1, s_2, s_3, g_m \cdots g_0, a) :=$
$$\Big(\pi_1(\sigma_0(s_1, s_2, s_3, g_m, a)), \pi_2(\sigma_0(s_1, s_2, s_3, g_m, a))g_{m-1} \cdots g_0\Big),$$
then
$\forall m \in \mathbb{N}, i \in \{-m, \ldots, -1, 0, 1, \ldots, m\} : \forall s_i \in S,\ \gamma_i \in \Gamma^+,\ a \in \Sigma :$
$\mathcal{T}((s_{-m}, \gamma_{-m}) \cdots (s_0, \gamma_0) \cdots (s_m, \gamma_m), aa_1a_2 \cdots) :=$
$$\begin{cases} (\bar{\sigma}(q_0, q_0, s_0, g_0)\bar{\sigma}_0(q_0, s_0, q_0, \gamma_0, a)\bar{\sigma}(s_0, q_0, q_0, g_0), a_1a_2 \cdots) & \text{if } m = 0 \\ (\bar{\sigma}(q_0, q_0, s_{-m}, g_0) \cdots \bar{\sigma}_0(s_{-1}, s_0, s_1, \gamma_0, a) \cdots \\ \qquad \cdots \bar{\sigma}(s_m, q_0, q_0, g_0), a_1a_2 \cdots) & \text{if } m > 0 \end{cases}$$

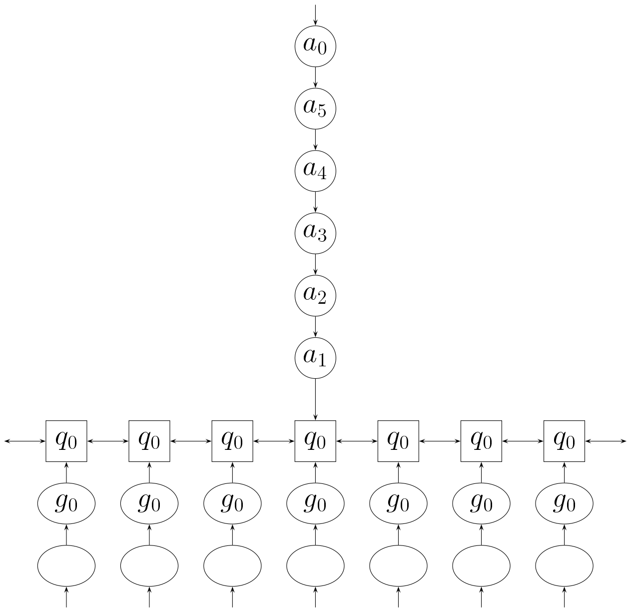


Figure 4: An iterative pushdown array.

Definition 1.2 Let $M = (S, \Sigma, \Gamma, \sigma_0, \sigma, q_0, a_0, g_0,)$ be an IPDA, $L \subseteq (\Sigma \setminus \{a_0\})^+$ a formal language and $t : \mathbb{N} \longrightarrow \mathbb{N}$ a function.
L is *accepted* by M in time t with respect to the *final states* $F \subseteq S$ if

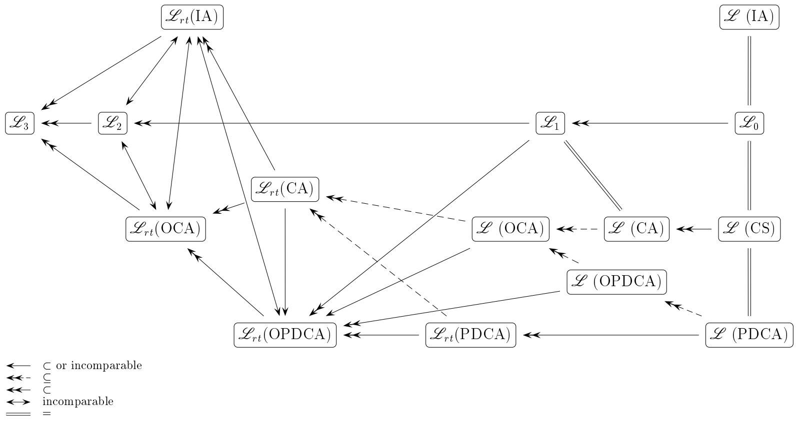
and only if $L = \{w \mid \pi_1(\pi_0(\mathcal{T}^{t(|w|)}((q_0, g_0), wa_0^\omega))) \in F \wedge \forall t' < t(n) : \mathcal{T}^{t'}((q_0, g_0), wa_0^\omega))) \notin F\}$ holds.

$\pi_i(x_1 \cdots x_n) := x_i$ selects the ith component of $x_1 \cdots x_n$ and $\mathcal{T}^k$ denotes the k-fold composition of $\mathcal{T}$.

The family of languages acceptable by a class POLY in time $t(n)$ is denoted by $\mathscr{L}_{t(n)}(\text{POLY})$. Of special interest are the real-time languages which can be accepted within time $rt(n) := n$. In case of unrestricted computation time the subscript is omitted. We denote the regular languages by $\mathscr{L}_3$, the context-free languages by $\mathscr{L}_2$, the deterministic context-sensitive languages by $\mathscr{L}_1$ and the recursively enumerable languages by $\mathscr{L}_0$.

2 What is known

In the following figure some of the known relationships between certain language families are summarized.



Observe for example the well-known Chomsky hierarchy $\mathscr{L}_3 \subset \mathscr{L}_2 \subset \mathscr{L}_1 \subset \mathscr{L}_0$ or the hierarchy $\mathscr{L}_3 \subset \mathscr{L}_{rt}(\text{OCA}) \subset \mathscr{L}_{rt}(\text{OPDCA}) \subset \mathscr{L}_1$ shown in [9].

3 Results

Now we focus our interest on the family $\mathscr{L}_{rt}$(IPDA). Of course, the following lemma holds since $\mathscr{L}$ (IPDA) equals to the recursively enumerable languages.

Lemma 3.1 $\mathscr{L}_{rt}(\text{IPDA}) \subset \mathscr{L}\,(\text{IPDA})$

For structural reasons we immediately obtain:

Lemma 3.2 $\mathscr{L}_{rt}(\text{IA}) \subseteq \mathscr{L}_{rt}(\text{IPDA})$

Theorem 3.1 $\exists L \in \mathscr{L}_2 : L \notin \mathscr{L}_{rt}(\text{IPDA})$

Proof. $L := \{vv' \mid v, v' \in \{\mathtt{a}, \mathtt{b}, \mathtt{0}, \mathtt{1}\}^* \wedge |v'| \geq 2 \wedge v' = v'^R\}$ is context-free. Define $W_n := \{\mathtt{0}w\mathtt{1} \mid w \in \{\mathtt{a}, \mathtt{b}\}^+ \wedge |w| = n\}$ for arbitrary $n \in \mathbb{N}$. Define for all subsets $U = \{u_1, \ldots, u_m\} \subseteq W_n$ a string

$$u := \begin{cases} \varepsilon & \text{if } U = \emptyset \\ u'_m & \text{otherwise} \end{cases},$$

where u'_m is recursively defined:

$$u'_0 := \varepsilon, \qquad u'_{i+1} := u^R_{i+1} u'^R_i u'_i \text{ for } i \in \{0, \ldots, m-1\}.$$

It is easy to see that for different subsets U and V from W_n the strings u and v are different. Furthermore: $\forall u_i \in U : uu_i \in L$ and $\forall \bar{u} \in W_n \setminus U : u\bar{u} \notin L$. Assume there exists an IPDA accepting L in real-time. Choose n such that $2^{2^n} > \max\{|S|, |\Gamma|\}^{(2n+5)^2}$ holds. It follows $|W_n| = 2^n$ and there exist 2^{2^n} different subsets of W_n. Assume now the automaton has fetched its input down to l symbols. In such a situation there are less than $\max\{|S|, |\Gamma|\}^{(2l+1)^2}$ possibilities to obtain a result with respect to the remaining input. Let $l = n + 2$, then there are less than $\max\{|S|, |\Gamma|\}^{(2n+5)^2}$ possibilities.

Since our choice of n there are at least two subsets $\tilde{U}$ and $\hat{U}$ for which the configurations are identical. Without loss of generality there exists a string $u_i \in \tilde{U} \setminus \hat{U}$. Therefore, the automaton has to accept $\tilde{u}u_i \in L$, but would accept $\hat{u}u_i \notin L$, too. $\square$

Theorem 3.2 $\exists L \in \mathscr{L}_1 \setminus \mathscr{L}_2 : L \in \mathscr{L}_{rt}(\text{IPDA})$

Proof. $L = \{\mathtt{a}^{2^n} \mid n \in \mathbb{N}\}$ is a real-time IA and, therefore, a real-time IPDA language. But L is not context-free. $\square$

We conclude that the families $\mathscr{L}_2$ and $\mathscr{L}_{rt}(\text{IPDA})$ are incomparable.

Since the language $\{vv' \mid v, v' \in \{\mathtt{a},\mathtt{b},\mathtt{0},\mathtt{1}\}^* \wedge |v'| \geq 2 \wedge v' = v'^R\}$ belongs to $\mathscr{L}_{rt}(\text{PDCA})$ and for a trivial simulation we obtain:

Lemma 3.3 Either $\mathscr{L}_{rt}(\text{IPDA}) \subset \mathscr{L}_{rt}(\text{PDCA})$ or $\mathscr{L}_{rt}(\text{IPDA})$ and $\mathscr{L}_{rt}(\text{PDCA})$ are incomparable.

Theorem 3.3 $\exists L \notin \mathscr{L}_{rt}(\text{OPDCA}) : L \in \mathscr{L}_{rt}(\text{IPDA})$

Proof. $L = \{\mathtt{a}^{2^n} \mid n \in \mathbb{N}\} \notin \mathscr{L}_{rt}(\text{OPDCA}) \supset \mathscr{L}_{rt}(\text{OCA})$. But $L \in \mathscr{L}_{rt}(\text{IA}) \subseteq \mathscr{L}_{rt}(\text{IPDA})$. $\square$

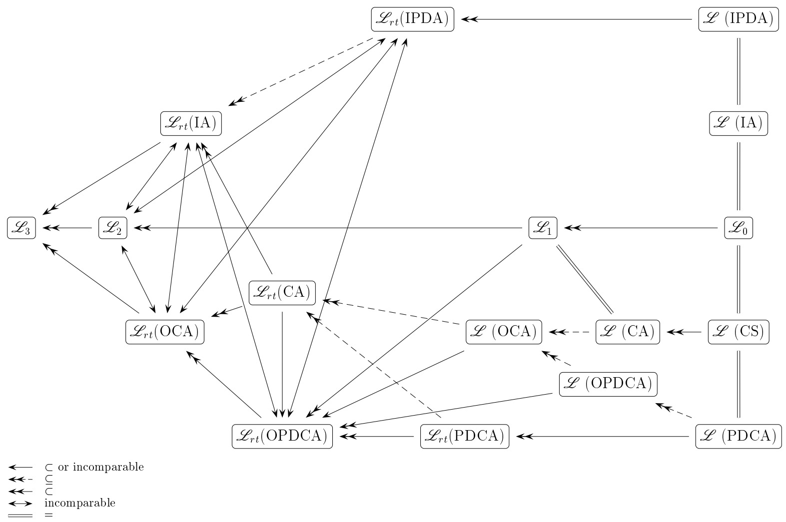
Theorem 3.4 $\exists L \in \mathscr{L}_{rt}(\text{OCA}) : L \notin \mathscr{L}_{rt}(\text{IPDA})$

Proof. $\{vv' \mid v, v' \in \{\mathtt{a},\mathtt{b},\mathtt{0},\mathtt{1}\}^* \wedge |v'| \geq 2 \wedge v' = v'^R\}$ belongs to $\mathscr{L}_{rt}(\text{OCA}) \subset \mathscr{L}_{rt}(\text{OPDCA})$. $\square$

We conclude that the families $\mathscr{L}_{rt}(\text{OPDCA})$ and $\mathscr{L}_{rt}(\text{IPDA})$ as well as the families $\mathscr{L}_{rt}(\text{OCA})$ and $\mathscr{L}_{rt}(\text{IPDA})$ are incomparable.

4 What is known now

The following figure is the updated one of section 2. Now the family $\mathscr{L}_{rt}(\text{IPDA})$ and some of its relationships are depicted, too.



5 What is not known

There are a lot of open problems in the area dealt with. One of the most popular unanswered questions is the question whether one of the inclusions $\mathscr{L}_{rt}(\text{CA}) \subseteq \mathscr{L}\ (\text{OCA}) \subseteq \mathscr{L}\ (\text{CA})$ is a proper one.
Another good question asks for the relationship between $\mathscr{L}_{rt}(\text{PDCA})$ and $\mathscr{L}\ (\text{OPDCA})$. Trivially, $\mathscr{L}_{rt}(\text{PDCA}) \supseteq \mathscr{L}_{rt}(\text{CA})$ holds, but is the inclusion a proper one?
Whether all context-free languages are real-time CA languages (or real-time OPDCA languages) or not is a question Smith III raised in the early seventies.

References

[1] Beyer, W. *Recognition of topological invariants by iterative arrays*. Technical Report MAC TR-66, Massachusetts Institute of Technology, 1969.

[2] Bucher, W. and Čulik II, K. *On real time and linear time cellular automata.* RAIRO Informatique Théorique et Applications. 18 (1984), 307–325.

[3] Choffrut, C. and Čulik II, K. *On real-time cellular automata and trellis automata.* Acta Informatica 21 (1984), 393–407.

[4] Cole, S. N. *Real-time computation by n-dimensional iterative arrays of finite-state machines.* IEEE Conference Record of 7^{th} Annual Symposium on Switching and Automata Theory, 1966, 53–77.

[5] Dyer, C. R. *One-way bounded cellular automata.* Information and Control 44 (1980), 261–281.

[6] Kosaraju, S. R. *On some open problems in the theory of cellular automata.* IEEE Transactions on Computers C-23 (1974), 561–565.

[7] Kutrib, M. *Real-time language recognition by pushdown cellular automata.* Proc. Workshop on Parallel Processing, Lessach, Österreich, TU Clausthal, 1994, 126–140.

[8] Kutrib, M. *On stack-augmented polyautomata.* Report 9501, Arbeitsgruppe Informatik, Universität Gießen, Gießen, 1995.

[9] Kutrib, M. and Richstein, J. *Real-time one-way pushdown cellular automata languages.* Proc. Developments in Language Theory '95, World Scientific, Singapore, 1996.

[10] Kutrib, M. and Worsch, Th. *Investigation of different input modes for cellular automata.* Proc. Parcella '94, Akademie Verlag, Berlin, 1994, 141–150.

[11] Seidel, S. R. *Language recognition and the synchronization of cellular automata.* Technical Report 79-02, Department of Computer Science, University of Iowa, Iowa City, 1979.

[12] Smith III, A. R. *Real-time language recognition by one-dimensional cellular automata.* Journal of Computer and System Sciences 6 (1972), 233–253.

[13] Smith III, A. R. *Introduction to and survey of polyautomata theory*. In Lindenmayer, A. and Rozenberg, G. (eds.), *Automata, Languages, Development*. North-Holland, Amsterdam, 1976, 405–422.

[14] Vollmar, R. *Algorithmen in Zellularautomaten*. Teubner, Stuttgart, 1979.

Automaten mit der Datenstruktur Menge

Klaus-Jörn Lange und Klaus Reinhardt

Wilhelm-Schickard Institut für Informatik
Universität Tübingen
Sand 13, D-72076 Tübingen

{lange,reinhard}@informatik.uni-tuebingen.de

1 Einführung

Ein Hauptgrund für die Betrachtung formaler Sprachen war die Modellierung der Syntax höherer Programmiersprachen, wie es etwa in der Beschreibung von Klammerstrukturen durch kontextfreie Grammatiken geschieht. Dieses Vorgehen findet jedoch schnell seine Grenzen, wenn es um die Berücksichtigung von Nebenbedingungen geht. Ein exemplarischer Fall ist das Problem der Wohldeklariertheit von Programmvariablen. Im einfachsten Fall ohne Berücksichtigung von Blockstruktur oder Verschattung läßt es sich als folgende Forderung formulieren: *Jede im Programm verwendete Variable muß genau einmal deklariert sein.* Formalisiert bedeutet das, daß es eine Abbildung von der Menge aller Variablenvorkommen in die Menge der Deklarationen gibt. Diese Abbildungsforderung läßt sich in die zwei Eigenschaften der Linkstotalität und der Rechtseindeutigkeit zerlegen. Formalisiert führt das auf die beiden Mengen

$$L_t = \left\{ x_1 c x_2 \cdots c x_n d y_1 c y_2 \cdots c y_m \;\middle|\; \begin{array}{l} m, n \geq 1, x_i, y_j \in \{a, b\}^*, \\ \bigwedge_{1 \leq i \leq n} \bigvee_{1 \leq j \leq m} x_i = y_j \end{array} \right\}$$

$$\text{und}\quad L_e = \left\{ y_1 c y_2 \cdots c y_m \;\middle|\; m \geq 1, y_i \in \{a,b\}^*, \bigwedge_{i \neq j} y_i \neq y_j \right\}.$$

Beide Sprachen liegen in AC^0, haben also ein sehr einfaches Wortproblem. Im Unterschied dazu scheint die formalsprachliche Behandlung der beiden Mengen größere Schwierigkeiten zu bereiten. So war Fischer [2] nur in der Lage, L_t als Indexsprache darzustellen. In der Tat liegt L_t sogar in $ETOL$. Aber sowohl $ETOL$ als auch *INDEX* haben ein *NP*-vollständiges Wortproblem.

Ziel dieser Arbeit ist nun die formalsprachliche Behandlung von L_e. Hierbei betrachten wir als typisch formalsprachliche Eigenschaften, daß die Sprachfamilie ein konstruktives Trio bildet, ein entscheidbares Leerheitsproblem besitzt (und damit Pumping- oder Copying Theoreme ermöglicht) und sich darstellen läßt durch Automaten oder Grammatiken.

Zu diesem Zweck führen wir hier *Automaten mit der Datenstruktur Menge*, kurz Mengenautomaten, ein. Diese verfügen über eine einweg–Eingabe, eine endliche Anzahl von Zuständen und ein einweg–beschreibbares *Frageband*. Die möglichen Schritte deines derartigen Automaten sind das Lesen eines Eingabezeichens, das Verlängern des Wortes aus dem Anfrageband und das *Stellen der Anfrage*: der Bandinhalt wird mit der Menge bisher gestellter Anfragen verglichen, die möglicherweise neue Anfrage der Menge der bisherigen Fragen hinzugefügt und schließlich das Frageband gelöscht.

Die Klasse der so definierten *Mengensprachen* bilden ein Trio, sind also abgeschlossen gegenüber Homomorphismen, inversen Homomorphismen und Schnitten mit reguären Sprachen. Als ein Hauptergebnis dieser Arbeit präsentieren wir einen Beweis für die Entscheidbarkeit des Leerheitsproblems für Mengenautomaten. Dieses Ergebnis kann als ein Indiz dahingehend verstanden werden, daß die Mengensprachen eine Familie bilden, die strukturell den kontextfreien Sprachen ähnelt. Desweiteren zeigen wir die *NP*-Härte des (entscheidbaren) Wortproblems.

Bezüglich der verwendeten Begriffe und Notationen verweisen wir auf

Werke wie [3, 4]. Im weiteren bezeichne λ das leere Wort.

2 Automaten mit Mengen

Der Hauptabschnitt dieser Arbeit gliedert sich in eine formale Definition des Begriffs des Mengenautomaten und der Mengensprachen, einen Unterabschnitt über formalsprachliche Aspekte und eine Übersicht von Ergebnissen zu Entscheidbarkeit und Komplexität mengensprachlicher Fragestellungen.

2.1 Definition und Arbeitsweise

Ein Mengenautomat besitzt eine nichtdeterministische, endliche Kontrolle, eine einweg-Eingabe und ein einweg-Ausgabeband, das im weiteren Angfrageband genannt wird. Es gibt drei Arten von elementaren Operationen: das Lesen eines Eingabezeichens (sogenannte δ_r–Transitionen), das Schreiben eines Zeichens auf das Anfrageband (sogenannte δ_b–Transitionen) und das Stellen einer Anfrage (sogenannte δ_q–Transitionen). Hierbei wird das auf dem Anfrageband befindliche Wort mit der Menge der bisher gestellten Anfragen verglichen, die Anfrage (wenn neu) der Menge der bisher gestellten Anfragen hinzugefügt und das Anfrageband gelöscht. Demgemäß gibt es unter den Zuständen der endlichen Kontroll zwei besondere Arten: Antwortzustände, in denen begonnen wird eine neue Anfrage zu komponieren, und Fragezustände, aus denen heraus eine Anfrage gestellt wird. Die Beantwortung der Anfrage erfolgt durch Übergang in positive oder negative Antwortzustände. Endzustände und der Startzustand gelten dabei als spezielle Antwortzustände.

Definition 1 *Ein* Mengenautomat *ist ein X-Tupel* $A = (Z, \Sigma, \Gamma, \delta, Q, S, E, z_0)$*, wobei*

- Z *eine endliche Menge von* Zuständen
- Σ *eine endliches* Eingabealphabet*,*

- Γ *ein endliches* Fragealphabet,
- $Q \subseteq Z$ *eine Menge von* Anfragezuständen
- $S \subseteq Z \setminus Q$ *eine Menge von* Antwortzuständen,
- $E \subseteq S$ *eine Menge von* Endzuständen,
- $z_0 \in S$ *ein* Startzustand *und*
- $\delta = \delta_r \cup \delta_b \cup \delta_q$ *mit*

  $\delta_r \subset (Z\backslash Q)\times\Sigma\times(Z\backslash S)$, $\delta_b \subset (Z\backslash Q)\times\Gamma\times(Z\backslash S)$, $\delta_q \subset Q\times S\times S$

  eine endliche Menge von Transitionen *ist.*

Um Berechnungen von Mengenautomaten definieren zu können, benötigen wir zunächst den Begriff der *Konfiguration*, also des Globalzustandes zwischen zwei Berechnungsschritten. Eine Konfiguration besteht dabei aus dem aktuellen Zustand, dem restlichen Eingabewort, dem bisher gebildeten Teil des Anfragewortes und der (immer endlichen) Menge von vorher gestellten Anfragen.

Definition 2 *Sei $A = (Z, \Sigma, \Gamma, \delta, Q, E, S, z_0)$ ein Mengenautomat.*

a.) *Eine* Konfiguration *von A ist ein Element aus $Z \times \Sigma^* \times \Gamma^* \times 2^{\Gamma^*}$.*

b.) *Eine Konfiguration (z, v, α, M) von A* geht direkt über *in die Konfiguration (z', v', α', M') (im Zeichen $(z, v, \alpha, M) \vdash (z', v', \alpha', M')$ genau dann, wenn gilt:*

Lesen eines Eingabezeichens: *Es gibt eine Transition $(z, a, z') \in \delta_r$ und es gilt $v = av'$ sowie $\alpha = \alpha'$ und $M = M'$,*

Verlängern des Anfragezeichens: *Es gibt eine Transition $(z, x, z') \in \delta_b$ und es gilt $\alpha x = \alpha'$ sowie $v = v'$ und $M = M'$ oder*

Stellen der Anfrage: *Es gibt eine Transition* $(z, z_+, z_-) \in \delta_q$ *und es gilt entweder* $\alpha \in M$, $M = M'$ *und* $z' = z_+$ *oder* $\alpha \notin M$, $M' = M \cup \{\alpha\}$ *und* $z' = z_-$ *sowie* $v = v'$ *und* $\alpha' = \lambda$.

Mit $\vdash^*$ *bezeichen wir die transitive Hülle der Relation* $\vdash$.

c.) *Die von* A akzeptierte Sprache *ist*

$$L(A) = \left\{ w \in \Sigma^* \;\middle|\; \bigvee_{z \in E} \bigvee_{\alpha \in \Gamma^*} \bigvee_{M \subset \Gamma^*} (z_0, w, \lambda, \emptyset) \vdash^* (z, \lambda, \alpha, M) \right\}.$$

Im weiteren verstehen wir unter einer *Mengensprache* eine von einem Mengenautomaten akzeptierte Sprache.

2.2 Formalsprachliche Eigenschaften

Aus der automatenbasierten Darstellung der Mengensprachen ergeben sich unmittelbar mehrere Abschlußeigenschaften, die hier ohne Beweis mitgeteilt werden. Es sei darauf hingewiesen, daß diese Abschlüsse konstruktiv sind.

Satz 3 *Die Mengensprachen bilden ein TRIO, sind also abgeschlossen gegen (möglicherweise löschende) Homomorphismen, inverse Homomorphismen und Schnitten mit regulären Mengen.*

Ebensolches gilt für die Operationen Vereinigung und Konkatenation. Die Beweisidee dabei ist es, disjunkte Anfragealpabete zu verwenden. Dieses Verfahren reicht jedoch nicht für die Operation der Kleeneschen Hüllenbildung aus. Es ist gegenwärtig offen, ob die Mengensprachen gegen Sternbildung abgeschlossen sind.

Satz 4 *Die Sprache*

$$L_{+,-} = \left\{ y_1 \delta_1 c y_2 \delta_2 \cdots c y_m \delta_m \;\middle|\; \begin{array}{c} m \geq 1, y_i \in \{a, b\}^*, \delta_i \in \{+, -\}, \\ \bigwedge_{1 \leq i \leq m} \delta_i = + \Leftrightarrow \left(\bigvee_{j < i} y_j = y_i \right) \end{array} \right\}$$

ist ein Trio-Erzeuger der Mengensprachen, d.h. zu jeder Mengensprache L gibt es zwei Homomorphismen f und g sowie eine reguläre Sprache R derart, daß gilt $L = f(g^{-1}(L_{+,-}) \cap R)$.

Aus Platzgründen verzichten wir auf den einfachen Beweis dieser Behauptung. Es ist in diesem Zusammenhang interessant zu beobachten, daß die Ausgangsspache L_e der Teilmenge derjenigen Wörter von $L_{+,-}$ entspricht, in denen kein Symbol $+$ auftaucht, in denen also keine Anfrage positiv beantwortet wird. Bei Umdrehen der negativen und positiven Antworten kann also L_e sozusagen als konjunktive Einschränkung von $L_{+,-}$ angesehn werden, wenn man Mengenautomaten als Orakelmaschinen mit einem Mengenorakel auffaßt.

2.3 Entscheidbarkeit und Komplexität

Satz 5 *Das Leerheitsproblem für Mengensprachen ist entscheidbar.*

Beweisidee: Es sei $A = (Z, \Sigma, \Gamma, \delta, Q, E, S, z_0)$ ein Mengenautomat. Im weiteren sei m die Mächtigkeit von $S \times Q$, also $m = |S| \cdot |Q|$. Offenbar ist A ohne die δ_q-Transitionen ein endlicher Automat mit Eingabe und Ausgabe, der eine rationale Transduktion $\tau_A \subset Z \times \Sigma^* \times \Gamma^* \times Z$ realisiert ([1]). τ_A besteht also genau aus jenen Tupeln (z, w, α, z'), für die ein Pfad von z nach z' aus δ_r– und δ_b–Transitionen existiert, auf dem w gelesen und α ausgegeben wird. Daher ist zu jedem Antwortzustand $s \in S$, und jedem Fragezustand $q \in Q$ die Menge

$$R_{s,q} = \left\{ \alpha \in \Gamma^* \;\middle|\; \bigvee_{w \in \Sigma^*} (s, w, \alpha, z') \in \tau_M \right\}$$

regulär. Für jede Teilmenge $N \subseteq S \times Q$ ist daher auch folgende Menge regulär:

$$R_N = \left\{ \alpha \in \Gamma^* \;\middle|\; \bigwedge_{(s,q) \in S \times Q} \alpha \in R_{s,q} \Leftrightarrow (s,q) \in N \right\}.$$

Die 2^m (möglicherweise leeren) Mengen $R_N, N \subseteq S \times Q$, bilden eine Partition von Γ^*.

Wir setzen nun $F = \{N \subseteq S \times Q \,|R_N$ ist endlich$\}$ und $I := (S \times Q) \setminus F$. Wir betrachten jetzt die Menge der *Globalzustände*:

$$K = S \times \{0, \infty\}^{|I|} \times \prod_{N \in F} \{0, 1, \cdots, |R_N|\}.$$

K ist endlich. Der *Globalstartzustand* ist $(z_0, \overbrace{0, 0, \cdots, 0}^{m-mal})$. Ein *Globalendzustand* ist ein Globalzustand $(s, i_1, \cdots, i_m)$, für den $s \in E$ gilt. Wenn sich der Mengenautomat M nach Stellung einer Frage in einem Antwortzustand s befindet, so enthält ein Globalzustand nicht nur den Zustand s, sondern auch für jedes $N \subseteq S \times Q$ die Anzahl der gefragten Wörter, die genau in R_N liegen. Im Falle eines unendlichen R_N wird dabei nur gespeichert, ob bisher kein ($i_N = 0$) oder mindestens ein ($i_N = \infty$) Wort aus R_N gefragt wurde.

Wir betrachten jetzt die Elemente aus K als Knoten eines gerichteten Graphen, der folgende Kanten aufweist:

$$(s, i_1, i_2, \cdots, i_m) \longrightarrow (s', i'_1, i'_2, \cdots, i'_m)$$

genau dann, wenn es ein $(q, s_1, s_2) \in \delta_q$ gibt mit:

Fall 1: Anfrage mit positiver Antwort

$$s' = s_2 \wedge \bigwedge_{N \subseteq S \times Q} (i'_N = i_N \wedge ((s, q) \in N \Rightarrow i_N \neq 0))$$

oder

Fall 2: Anfrage mit negativer Antwort

$$s' = s_1 \wedge \bigwedge_{N \subseteq S \times Q} ((s, q) \notin N \Rightarrow i'_N = i_N) \wedge$$

$$\bigwedge_{N \subseteq S \times Q} (((s, q) \in N \wedge N \in I) \Rightarrow i'_N = i_N = \infty) \wedge$$

$$\bigwedge_{N \subseteq S \times Q} (((s,q) \in N \wedge N \in F) \Rightarrow (i_N < |R_N| \wedge i'_N = i_N + 1))\,.$$

Fall 1 symbolisiert die Situation, in der eine Anfrage α zum wiederholten Male gestellt wurde. Es erfolgt also in A ein Übergang von q nach s_2

Offenbar gibt es genau dann eine akzeptierende Berechnung von A, wenn es in dem Graphen der Globalzustände einen Weg vom Globalstartzustand in einen Globalendzustand gibt. Die behauptete Entscheidbarkeit folgt aus der Endlichkeit der Menge K. □

Korollar 6 *Das Wortproblem für Mengensprachen ist entscheidbar.*

Beweis: Da $w \in L(A)$ genau dann gilt, wenn $L(A) \cap \{w\}$ nichtleer ist, folgt die Behauptung aus der effektiven Abgeschlossenheit der Mengensprachen gegenüber Schnitten mit regulären Mengen. □

Satz 7 *Es gibt Mengenautomaten, deren Wortproblem NP-hart ist.*

Beweisidee: Aus einer Kodierung F einer aussagenlogischen Formel in konjunktiver Normalform läßt sich (per Log-Reduktion) das Wort $V\$F$ herstellen, in dem V eine Auflistung aller in F vorkommenden Variablen darstellt. Ein Mengenautomat A liest nun zunächst das Teilwort V und rät zu jeder Variable x_i eine Belegung, indem er entweder die Anfrage x_iT oder x_iF stellt, die natürlich allesamt negativ beantwortet werden. Danach liest A die Formel F und rät zu jeder Klausel das erfüllende Literal. A prüft die Korrektheit indem er Anfragen der Form x_iT oder x_iF stellt, die nun allesamt positiv beantwortet werden müssen. In diesem Fall wird akzeptiert, andernfalls lehnt A ab. □

3 Diskussion und offene Fragen

Bei der Definition der Mengenautomaten wurde darauf geachtet, einen Automatentyp zu erhalten, der ein entscheidbares Leerheitsproblem besitzt. Schon kleinste Aufweichungen des Konzepts der Art, daß das Frageband nicht gelöscht wird oder dergl., sollten zur Unentscheidbarkeit der Leerheit führen. Die möglichst starke Einschränkung dieses Modells brachte eine gewisse Unhandlichkeit im Gebrauch mit sich. So ist es noch offen, ob Mengensprachen abgeschlossen sind gegen Kleene'sche Hüllenbildung. Andererseits ist damit keine Verminderung der Komplexität des Wortproblems verbunden, da dieses *NP*-hart ist. Unbekannt ist aber, ob *NP* auch eine obere Schranke für das Wortproblem ist. Dieses berührt auch die Frage, ob das L_e erzeugte Trio ein Wortproblem mit kleinerer Komplexität hat als das von $L_{+,-}$ erzeugte, also das der Mengensprachen. Der Beweis der *NP*-Härte des Wortproblems der Mengensprachen machte extensiven Gebrauch von der Möglichkeit, ebenso positive wie negative Antworten zu verwenden, wenn auch nicht in unbeschränkter Reihenfolge. Es wurden zuerst nur negative und dann nur noch positive Antworten verwendet. Diese Möglichkeit bietet L_e nicht. Es besteht also die Möglichkeit, das Wortproblem komplexitätsmäßig in die Klasse P zu schieben.

Literatur

[1] J. Berstel. *Transductions and Context-Free Languages.* Teubner Verlag, Stuttgart, 1979.

[2] M.J. Fischer. Grammars with macro-like production. Ph.d. thesis, Harvard Univ., 1968.

[3] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Language, and Computation.* Addison-Wesley, Reading Mass., 1979.

[4] Arto Salomaa. *Formale Sprachen.* Springer Verlag, 1974.

Turing Maschinen mit sublogarithmischen Platzschranken

—

The Complexity World between Constant and Logarithmic Space

Maciej Liśkiewicz and Rüdiger Reischuk

Med. Universität zu Lübeck
Institut für Theoretische Informatik
Wallstraße 40, D–23560 Lübeck

{liskiewi,reischuk}@informatik.mu-luebeck.de

Abstract

We investigate space complexity classes defined by Turing machines that use less than logarithmic space. Because of the limited counting ability of such machines most of the standard simulation techniques do not work for sublogarithmic space classes. However, machines with such little space may still be quite powerful. Therefore, it was not obvious how to obtain analogs for inclusion and separation results known for classes above logspace.

We review known facts about the sublogarithmic space world and present several new results, which show that these classes really behave differently. For example, certain closure properties do not hold. The restricted power of these machines makes it possible to prove explicit separations – even for alternating complexity classes – by combinatorial arguments and to obtain a hierarchy of non-relativized complexity classes without any unproven assumption. We will also discuss upward and downward translation issues.

These complexity classes are related to other classes within $\mathcal{P}$, in particular to context-free languages. Finally, we consider extensions to machines that, in addition, can make use of randomness, in other words to interactive proof systems with small space bounds.

Key words: space complexity, sublogarithmic complexity bounds, alternating Turing machines, halting computations, complementation of languages, complexity hierarchies, closure properties, context-free languages, bounded languages, interactive proof systems, Arthur-Merlin-Games.

1 Introduction

In the following, basic knowledge in computational complexity is assumed. More details concerning the machine models and notation can be found in [WaWe86, BDG90, Re90].

The logarithm seems to be the most crucial bound for space complexity. Since with less space one cannot implement a counter for a polynomial size range neither a position within the input nor the number of steps of a computation can be recorded. Therefore, most of the standard techniques used for machine simulations do not work in this case, and for many standard properties it was not clear whether they also hold for sublogarithmic space bounds.

For example, a novel technique was necessary to prove for small bounds S that $DSpace(S)$ is closed under complement [Si80], where $DSpace(S)$ denotes the set of all languages that can be recognized by S-space bounded deterministic Turing machines (DTM). Savitch's technique to simulate nondeterministic machines by deterministic machines requires space $(\max\{\log, S\})^2$ because of the recursion depth and the necessity to specify machine configurations. This gives a quadratic increase of the space for logarithmic or larger bounds. By some work the space can be reduced to $S \cdot \max\{\log, S\}$ [MoSu82], but still one gets an exponential blowup for very small space bounds.

Another important problem of this type is whether the closure under complement for NTM

$$NSpace(S) \;=\; co\text{-}NSpace(S)$$

shown by Immerman and Szelépcsenyi [Im88, Sz88] remains valid for space bounds $S < \log$. If this equality were not valid then obviously $DSpace(S)$ is properly contained in $NSpace(S)$.

Do relations between sublogarithmic classes have impact for larger complexity classes? In general, the principles of upward collapse and downward separation which can easily be proved by standard translation techniques for at least logarithmic space bounds do not seem to hold in the sublogarithmic space world. However, by a more complicated argument Szepietowski could at least show the implication:

Theorem 1 [Sz90] *If* $DSpace(\log\log) \;=\; NSpace(\log\log)$ *then* $DSpace(\log) \;=\; NSpace(\log)$.

Hence equality between deterministic and nondeterministic space at a very low level still gives the identity for large space classes.

Although because of the missing counters sublogarithmic space complexity classes are technically quite difficult to handle, at first glance they may look very weak. For example, it has been known for quite a while that the deterministic context-free language

$$\texttt{COUNT} \;:=\; \{1^n 0 1^m \mid n = m\}$$

cannot be recognized with sublogarithmic space [LSH65], even by nondeterministic machines. This has been generalized in [AGM92] to the result that $NSpace(o(\log))$ does not contain *any* deterministic context-free nonregular language. These results, however, do not extend to probabilistic classes. Freivalds has shown the surprising result [Fr81] that `COUNT` can be accepted by a probabilistic TM in constant space with an arbitrarily small constant for the error probability.

Thus, why should one be interested in sublogarithmic space complexity classes? Let us use the shorthand **llog** for the twice iterated logarithmic function $n \mapsto \log_2 \log_2 n$. A deterministic or nondeterministic Turing machine (TM) that uses less than llog space can recognize regular languages only [SHL65, HoUl69]. Since there exist nonregular languages in $DSpace(\text{llog})$ the function llog has turned out to be the smallest nontrivial space bound, at least for deterministic and nondeterministic TMs. Later, this lower space bound has been extended to alternating Turing machines (ATM), thus

Theorem 2 [Su80, Iw93]
$DSpace(o(\text{llog})) \;=\; NSpace(o(\text{llog})) \;=\; ASpace(o(\text{llog})) \;=\; \mathcal{REG}$.

The machine model we will refer to is the standard one for space bounds, which is equipped with a separate 2-way read-only input tape and one (or more) worktape(s). Restricted machines that can read their input only 1-way, also called *on line machines*, will not be considered here. For this model space complexity investigations are considerably easier. For example, the recognition of nonregular languages with 1-way input access requires at least logarithmic workspace, and simple tasks like the recognition of palindromes needs linear space [SHL65]. However, Sudborough has shown the surprising result that the closure of the class $ASpace(\text{llog})$ with 1-way restriction under logspace reducibility is quite strong [Su80]. It equals

$$\mathcal{SC} \;:=\; DTimeSpace(\text{POL}, \text{PLOG}) \;,$$

where POL denotes the set of all polynomial and PLOG the set of all polylogarithmic complexity bounds, i.e. $O(\log^k)$ for arbitrary k . Furthermore, $DTimeSpace(T, S)$ is the class of all languages that can be recognized with simultaneous time bound T' and space bound S' for some $T' \in T$ and $S' \in S$.

Since in [CKS81] it has been shown that $ASpace(\log) = \mathcal{P}$ sublogarithmic space classes even defined by alternating machines are contained in $\mathcal{P}$. However, without the 1-way restriction the best upper bound known for alternating sublogarithmic space classes seems to be:

Theorem 3 [Su80] *For* $S \in \texttt{SUBLOG}$ *holds*

$$ASpace(S) \;\subseteq\; DSpace(\mathrm{ExL}(S)) \ .$$

Here, $\mathrm{ExL}(S)$ denotes the set of all complexity bounds that are bounded exponentially with respect to S, i.e. by a function of the form $\exp(k \cdot S)$ for some $k \in \mathbb{N}$, and S itself has to fulfill certain weak constructibility conditions. In particular, for the smallest nontrivial space bound we get

$$ASpace(\mathrm{llog}) \;\subseteq\; DSpace(\mathrm{PLOG}) \ .$$

However, this simulation of ATM within polylogarithmic space takes more than polynomial time, and it is an open problem whether $ASpace(\mathrm{llog})$ is contained in $\mathcal{SC}$. Thus, sublogarithmic alternating space classes may actually be quite strong. Even, if we bound the number of alternations by some function A and let $AAlterSpace(A, S)$ denote the corresponding complexity class the best known upper bound drops down to:

Theorem 4 [MoSu82, To81, Su80]

$$AAlterSpace(A, S) \;\subseteq\; DSpace(S \cdot (S + A + \log)) \ .$$

This means for a constant number of alternations – denoted by the set CON of all bounded functions –

$$AAlterSpace(\mathrm{CON}, \mathrm{llog}) \;\subseteq\; DSpace(\log \cdot \mathrm{llog}) \ ,$$

and also this class may not be included in $\mathcal{SC}$. Therefore, sublogarithmic space bounded machines with the ability of alternations may be quite powerful devices. In the following we will investigate such machines in more detail and present combinatorial techniques to prove lower bounds. For this purpose, let us denote the set of all nontrivial sublogarithmic space bounds by

$$\texttt{SUBLOG} \;:=\; \Omega(\mathrm{llog}) \cap o(\log) \ .$$

2 The Alternating Sublogarithmic Space World

Complexity classes are typically described by nicely growing functions the values of which can be computed easily. This includes powers and roots of the logarithmic function or iterated logarithms. However, logarithmic space seems to be necessary for this task.

Definition 1 A function $A : \mathbb{N} \to \mathbb{N}$ is *computable in space* S if there exists a DTM that for all inputs of the form 1^n writes down the binary representation of $A(n)$ on an extra 1-way output tape using no more than $S(n)$ work space. A is *approximable from below in space* S if there exists a function A' that is computable in space S with $A'(n) \leq A(n)$ for all $n \in \mathbb{N}$ and $A'(n) = A(n)$ for infinitely many $n \in \mathbb{N}$.

If S itself is computable in space S it is called *fully space-constructible.* An equivalent requirement would be the existence of a TM that on every input of length n uses exactly $S(n)$ memory cells.

There exists fully space-constructible functions in $o(\log)$. The best known example is

$$F(n) \;:=\; \text{smallest integer that does not divide } n\,,$$

which due to the distribution of primes even belongs to $O(\text{llog})$. However, any such function has to equal some constant value infinitely often [FrLa75]. This means

Fact 1 *No fully space-constructible function belongs to* `SUBLOG`.

This property even holds for monotone increasing space bounds if one generalizes to nondeterministic constructors [Ge91], where it is required that for each input X at least one computation path uses space $S(|X|)$ and no path uses more than that.

If a complexity bound S is fully space-constructible for each input one could mark in advance the number of available memory cells to prevent that this space bound is exceeded. Thus assuming that for each string X in a language L an acceptor M for L has at least one accepting computation that uses space at most $S(|X|)$ we can easily achieve that all computation paths stay within this bound. For sublogarithmic bounds, however, the following distinction becomes important.

Definition 2 An ATM M is (*strongly*) S space-bounded if on every input X it only enters configurations that use at most $S(|X|)$ space. M is *weakly* S space-bounded if, for every input X that is accepted, it has an accepting computation tree all of which configurations use at most $S(|X|)$ space. $DSpace(S)$ denotes the class of languages accepted by S space-bounded DTMs and $weakDSpace(S)$ denotes the languages accepted by weakly S space-bounded DTMs. A corresponding notation is used for NTMs and ATMs.

We will concentrate on the more natural strong requirement for space complexity, which is also the one used in the fundamental work of Hartmanis, Lewis and Stearns. When studying closure properties and alternating hierarchies the weak measure is not appropriate. To recognize a language L in the weak sense an acceptor for L may use arbitrary space for strings in $\overline{L}$, while a machine for $\overline{L}$ has to be bounded on this set.

Another important distinction is that a machine with space bound $S \geq \log$ can make at most exponentielly in S many steps without getting into a loop. Thus, on a separate storage track one could also count the number of steps to detect infinite computation paths. This implies immediately that for such S the class $DSpace(S)$ is closed under complement. This method cannot be applied in the sublogarithmic space world. Sipser, however, found another way to check whether the starting configuration can reach an accepting configuration without the danger of being trapped in a loop. He simulates "backwards".

Theorem 5 [Si80] *For arbitrary S holds: $DSpace(S)$ is closed under complement .*

A special situation holds for bounded languages containing only strings of a certain block structure.

Definition 3 Let $Z : \mathbb{N} \to \mathbb{N}$ be a function. A language $L \subseteq \{0,1\}^*$ is Z*–bounded* if each $X \in L$ contains at most $Z(|X|)$ zeros. L is *bounded* if it is Z–bounded for some constant function Z.

The language COUNT defined above is an example for 1-boundedness. Sublogarithmic space bounded machines already face problems within a large block of identical symbols since they cannot record the exact length of such a block. The formal argument to show this property is known as the $n \mapsto n + n!$–technique developed by Stearns, Hartmanis, and Lewis in [SHL65]. For n large enough the machine will not notice any difference between the blocks 1^n and $1^{n+n!}$. In particular, if the TM obeys the space bound $s = S(n)$ in the smaller block it will use no more than s space in the large block, too.

Theorem 6 [LSH65, AlMe76, CIRB87, LiRe93a]

$$\begin{array}{rcl} \mathtt{COUNT} & \notin & DSpace(o(\log)) \;, \\ \overline{\mathtt{COUNT}} & \in & weakDSpace(\mathrm{llog}\,) \;, \\ \mathtt{COUNT} & \notin & weakNSpace(o(\log)) \;, \\ \overline{\mathtt{COUNT}} & \notin & ASpace(o(\log)) \;. \end{array}$$

These bounds also show that with respect to the weak measure there can occur an exponential difference between the space complexity of a language and its complement. Thus in contrast to the previous theorem weakly bounded deterministic classes are not closed under complement. Even an alternating machine cannot simulate a weakly bounded deterministic machine with less than an exponential increase of the space bound.

The inability to count, however, has a positive effect, at least in case of bounded languages. As exploited in the proof of the theorem above within a block of identical input symbols the machine has to behave in a periodic fashion implying that important things only happen at the right and left boundary of a block. This observation can be used to prove

Theorem 7 [HaBe76, CIRB87] *Every infinte bounded language that belongs to the class* $NSpace(\mathtt{SUBLOG})$ *contains an infinite regular subset.*

This restricted behaviour of machines with small space complexity for bounded languages also allows one to implement the inductive counting method within the same space bound and yields:

Theorem 8 [Ge93b, AGM92, Sz94] *For the class of Z –bounded languages, where Z is a constant or a small growing function, holds for sublogarithmic bounds S*

$$NSpace(S) \;=\; co\text{-}NSpace(S)\ .$$

Definition 4 *For $k \geq 1$, the class $\Sigma_k Space(S)$ and $\Pi_k Space(S)$ are defined as all languages accepted by alternating S space-bounded TMs that make at most $k-1$ alternations and start in an existential (resp. universal) state. We will also consider ATMs with a non-constant bound A for the number of alternations. In this case, the notation $\Sigma_A Space(S)$ and $\Pi_A Space(S)$ is used. $AAlterSpace(A, S)$ denotes $\Sigma_A Space(S) \cup \Pi_A Space(S)$.*

Hence, $\Sigma_1 Space(S) = NSpace(S)$. By standard techniques it follows from Immerman-Szelépcsenyi's result:

Theorem 9 *For $S \geq \log$ and for all $k \geq 1$*

$$\Sigma_1 Space(S) \;=\; \Sigma_k Space(S) \;=\; \Pi_k Space(S)\ .$$

Note that these techniques do not work for sublogarithmic space bounds. Chang et al. have shown in [CIRB87] that in the sublogarithmic space world the situation is really different.

Theorem 10 *There is a language in $\Pi_2 Space(\mathrm{llog})$ that does not belong to $\Sigma_1 Space(o(\log))$.*

Clearly, this proves that for space bounds S in `SUBLOG` the alternating S space hierarchy does not collapse to the first level and that

$$\Sigma_1 Space(S) \;\subset\; \Pi_2 Space(S)\ .$$

Does this relation also hold for higher levels? This question has received a lot of attention recently. Independently in a series of papers the answer was found [Br93, Ge93a, LiRe93a, BGR93, Ge93c, LiRe93b]. These classes indeed form a strict infinite hierarchy. A chronology of these results can be found in [Wa93].

Theorem 11 For all $k > 1$ holds

$$\begin{aligned} \Sigma_k Space(\mathrm{llog}) \setminus \Pi_k Space(o(\log)) &\neq \emptyset \qquad \text{and} \\ \Pi_k Space(\mathrm{llog}) \setminus \Sigma_k Space(o(\log)) &\neq \emptyset. \end{aligned}$$

These results give a complete and best possible separation for the sublogarithmic space world, except for the first level $k = 1$. It is open whether also $\Sigma_1 Space(S) \neq \Pi_1 Space(S)$ for $S \in \mathtt{SUBLOG}$. Although different methods have been used to separate Σ_k and Π_k none of them seems to be applicable to the first level. The theorem implies that the alternating sublogarithmic space hierarchy is an infinite one, contrary to the case for logarithmic or larger space bounds.

Corollary 1 For any $S \in \mathtt{SUBLOG}$ and all $k \geq 1$ holds

$$\begin{aligned} \Sigma_k Space(S) &\subset \Sigma_{k+1} Space(S) \ , \\ \Pi_k Space(S) &\subset \Pi_{k+1} Space(S) \ . \end{aligned}$$

Thus, the only open question is how $\Sigma_1 Space(S)$ and $\Pi_1 Space(S)$ relate. It is somewhat annoying that the new techniques do not give any help for the basic case $k = 1$. The possibility that both classes are equal is not completely unrealistic, which would give the novel result that a hierarchy is infinite, although its first level collapses. We conjecture that for $k = 1$ the classes are different, but a proof seems to be significantly more difficult than for larger k. The proofs of the separation results above use in an essential way that counting to a certain extent can be replaced by alternating quantifiers. In the base case, however we have only a nondeterministic machine, that is a single existential quantifier.

That the sublogarithmic space world can be even more complex can be observed if one restricts to bounded languages. In this case we have seen that $\Sigma_1 Space(S)$ is closed under complementation and that the Σ_1- and Π_1-classes are identical. The second and third level, however, can be separated by bounded languages [LiRe93a]. Nothing seems to be known for level 4 and higher. Thus, the sublogarithmic space hierarchy for bounded languages may be quite strange. We have made some observations leading to the conjecture that for bounded languages this hierarchy might indeed consist of only a finite number of distinct levels.

Let us come back to the full class of languages without any boundedness restrictions. The separation of the alternating levels can be generalized to an unbounded number of alternations as long as the product of alternation and space bounds remains sublogarithmic.

Theorem 12 [LiRe93b] For any pair of functions $S \in \mathtt{SUBLOG}$ and $A > 1$ with $A \cdot S \in o(\log)$, where A is approximable from below in space S, holds:

$$\begin{aligned} \Sigma_A Space(S) \setminus \Pi_A Space(S) &\neq \emptyset \,, \\ \Pi_A Space(S) \setminus \Sigma_A Space(S) &\neq \emptyset \,. \end{aligned}$$

Corollary 2 For any S and A as in the theorem above holds:

$$\begin{aligned} \Sigma_A Space(S) &\subset \Sigma_{A+1} Space(S) \,, \\ \Pi_A Space(S) &\subset \Pi_{A+1} Space(S) \,. \end{aligned}$$

Note that the class of functions that are approximable from below in space $S \in \mathtt{SUBLOG}$ is quite large. For example, it contains the bounds $\lfloor \log \rfloor$, $\lfloor \log^{1/2} \rfloor$, $\lfloor \mathrm{llog} \rfloor$ and $\lfloor \log^* \rfloor$. Thus one obtains:

1. $\Sigma_{\mathrm{llog}} Space(\mathrm{llog}) \subset \Sigma_{(\mathrm{llog})+1} Space(\mathrm{llog})$.
2. $\Sigma_S Space(S) \subset \Sigma_{S+1} Space(S)$ for $S = \log^{1/2-\epsilon}$ with $0 < \epsilon < 1/2$.
3. $\bigcup_{k \in \mathbb{N}} \Sigma_k Space(S) \subset \Sigma_{\log^*} Space(S)$ for $S \in \Omega(\mathrm{llog}) \cap o(\log / \log^*)$.
4. For $k \in \mathbb{N}$ let $\mathcal{ALSL}^k := AAlterSpace(\mathrm{llog}^k, \mathrm{llog})$. Then for any k holds: $\mathcal{ALSL}^k \subset \mathcal{ALSL}^{k+1}$.

Note that for logarithmic bounds the corresponding question is still open, i.e. for any k it is unknown whether

$$AAlterSpace(\log^k, \log) \subset AAlterSpace(\log^{k+1}, \log) \ ?$$

For any function S the class $\Sigma_1 Space(S)$ is closed under union and intersection. However, it is still an open problem whether for $S \in \texttt{SUBLOG}$ the class $\Sigma_1 Space(S)$ is closed under complementation. More general, for arbitrary k the classes $\Sigma_k Space(S)$ are closed under union, and symmetrically the $\Pi_k Space(S)$ are closed under intersection. In [LiRe93a] we have developed a technique showing that for $S \in \texttt{SUBLOG}$ and for $k = 2, 3$, $\Sigma_k Space(S)$ and $\Pi_k Space(S)$ are not closed under complementation. Furthermore, $\Sigma_k Space(S)$ is not closed under intersection, and $\Pi_k Space(S)$ not under union. Combining these ideas with the separation results above we get the same closure properties for all levels.

Theorem 13 *For any $S \in \texttt{SUBLOG}$ and all $k > 1$ holds: $\Sigma_k Space(S)$ and $\Pi_k Space(S)$ are not closed under complementation and concatenation. Moreover, $\Sigma_k Space(S)$ is not closed under intersection and $\Pi_k Space(S)$ is not closed under union.*

Note that non-closure under complementation for Σ_k and Π_k classes is not trivially equivalent to Theorem 11, which says that sublogarithmic $\Sigma_k Space$ and $\Pi_k Space$ are distinct, since sublogarithmic space-bounded machines cannot that easily detect an infinite computation path. And Sipser's method only works for deterministic machines. Thus, it is an interesting open problem whether

$$\Pi_k Space(S) \;=\; co\text{-}\Sigma_k Space(S)$$

for $k = 1, 2, \ldots$ (for more details see [LiRe93a]). But at least one can obtain the following partial solution generalizing Theorem 5 to ATMs in case of bounded languages. Such acceptors can always be guaranteed to halt. This implies

Theorem 14 *Let $S \in \texttt{SUBLOG}$ be a space bound and Z be a function computable in space S with $Z \leq \exp S$. Then for all $k \geq 1$ and for every Z-bounded language $L \subseteq \{0,1\}^*$ holds:*

$$L \in \Sigma_k Space(S) \iff \overline{L} \in \Pi_k Space(S) \; .$$

Observe that for $S \geq \log$ the function Z can grow linearly and then Z does not put any restriction on the structure of the strings in L.

Thus, this theorem gives a smooth approximation of the fact that for at least logarithmic space bounds Σ_k and Π_k are complementary for arbitrary languages. We conjecture that the computability of Z is needed in the claim above. Furthermore, there are some indications that the theorem might not be true in general for bounds Z much larger than $\exp S$.

Let us finally consider the relation between the sublogarithmic space world and context-free languages.

Definition 5 [St67] and [Gi72] *A language L is called strictly nonregular if one can find strings u, v, w, x and y such that $L \cap \{u\}\{v\}^* \{w\}\{x\}^*\{y\}$ is context-free, but nonregular.*

We have seen that the simple deterministic context-free language `COUNT` cannot even be recognized by an ATM in sublogarithmic space. This is not a coincidence because of the following general result.

Theorem 15 [AGM92, LiRe93b] *For any nonregular deterministic context-free, strictly nonregular language, or nonregular context-free bounded language L holds $L \notin AAlterSpace(\mathrm{CON}, o(\log))$. Furthermore, without bounds on the number of alternations it is not possible that L and $\overline{L}$ both belong to $ASpace(o(\log))$.*

There are more nontrivial languages besides `COUNT` that can be recognized in sublogarithmic space. An interesting example is

$$\mathtt{BIN} \; := \; \{\mathrm{bin}(1)\# \,\mathrm{bin}(2)\# \ldots \# \,\mathrm{bin}(m) \mid m \in \mathbb{N}\} \; ,$$

where $\mathrm{bin}(i)$ denotes the binary representation of the natural number i . However, the theorem above implies that such languages cannot be too simple like being context-free and deterministic or bounded.

The language `BIN` has the property that it can serve as an advice to construct the function llog , which as we have seen is not fully space-constructible. A machine can simply check the correctness of the sequence by looking at the i -th bit position of all subwords sequentially for $i = 0, 1, \ldots$. For this purpose counters of length $\log m$ suffice, where m is roughly the logarithm of the input length.

In the remaining part of this paper we will discuss the technical restrictions of sublogarithmic space bounded machines and give the general outline how some of the separations results for the alternating classes can be obtained. Complete proofs can be found in [LiRe93b]. At the end, we will discuss the question whether randomness can extend the power of machines with small space bounds.

3 Properties of Sublogarithmic Space-Bounded ATMs

For a TM M, a *memory state* is an ordered triple $\alpha = (q, u, i)$, where q is a state of M, u a string over the work tape alphabet, and i a position in u (the locaton of the work tape head). A *configuration* of M on an input X is a pair (α, j) consisting of a memory state α and a position j with $0 \leq j \leq |X| + 1$ of the input head. $j = 0$ or $j = |X| + 1$ means that this head scans the left, resp. the right end-marker. For a memory state $\alpha = (q, u, i)$ let $|\alpha|$ denote the length of the memory inscription u.

We may assume that for a successor (α', j') of a configuration (α, j) always holds $|\alpha'| \geq |\alpha|$. The state set of an ATM is partioned into subsets of existential, universal, accepting, and rejecting states. We say that a configuration $((q, u, i), j)$ is existential (resp. universal, accepting, or rejecting) if q has the corresponding mode. All accepting and rejecting configurations C are assumed to be terminating, i.e. there are no more configurations that can be reached from C. Let

$$(\alpha, i) \ \models^{\star}_{M,X} \ (\beta, j)$$

denote the property that the ATM M with X on its input tape has a computation path of the form $C_1 = (\alpha, i),\ C_2,\ \ldots,\ C_t = (\beta, j)$.

$$(\alpha, i) \ \models_{M,X} \ (\beta, j)$$

denotes the same fact, but with the following restriction: $t \geq 2$ and the mode of the configurations $C_2, \ldots, C_{t-1}$ is the same as that of C_1 (i.e. if C_1 is existential then all C_l for $l = 2, \ldots, t-1$ are existential, otherwise they are all universal). The predicate

$$\mathsf{acc}^k_M(\alpha, i, X)$$

holds if M starting in configuration (α, i) with X on its input tape accepts (i.e. has an accepting subtree), and on each computation path of that tree it makes at most $k-1$ alternations. Let

$$Space_M(\alpha, i, X)$$

denote the maximum space used in configurations M can reach on input X starting in configuration (α, i) and $Space_M(X) := Space_M(\alpha_0, 0, X)$, where $(\alpha_0, 0)$ is the initial configuration of M . Similarly let

$$Alter_M(\alpha, i, X)$$

denote the maximum number of alternations M can make on input X starting in configuration (α, i) and $Alter_M(X) := Alter_M(\alpha_0, 0, X)$.

3.1 Inputs of a Periodic Structure

In this section some properties of TM computations for binary inputs of the form $Z_1 W W \ldots W Z_2$ will be described. Let M always denote an arbitrary ATM and S a space bound in $o(\log)$. Depending on M and S , we choose a constant $\mathcal{N}_{M,S} \geq 2^8$ such that for all $n \geq \mathcal{N}_{M,S}$

$$(\mathcal{M}_n^6 + 1)^2 \ < \ n \quad \text{and} \quad S(n) \ < \ \frac{1}{2} \log n \ - 2 \ ,$$

where $\mathcal{M}_n$ denotes the number of memory states α with $|\alpha| \leq S(n)$. All claims following will hold for any integer $n \geq \mathcal{N}_{M,S}$.

From the work in [SHL65, Ge91] it is known that for sublogarithmic space bounded computations for any natural number ℓ the behavior of a DTM or NTM on input $1^{n+\ell n!}$ is exactly the same as on 1^n . We will show that a corresponding property holds for ATMs and for all inputs of the form

$$X \ = \ Z_1 \, W^n \, Z_2 \qquad \text{and} \qquad Y \ = \ Z_1 \, W^{n+\ell n!} \, Z_2 \ ,$$

where Z_1, Z_2, W are arbitrary binary strings and $\ell \in \mathbb{N}$. Since in the following we will often compare computations on such an input

X and a pumped version Y let us introduce a special notation for positions within these strings. If i is a position within X outside the pumped region W^n, that means for the example above either in Z_1 or in Z_2, then $\hat{i}$ denotes the corresponding position within Y. Thus

$$\hat{i} := \begin{cases} i & \text{if } i \leq |Z_1| \ , \\ i + |Y| - |X| & \text{if } i > |Z_1 W^n| \ . \end{cases}$$

The main technical tools for the analysis of sublogarithmic space-bounded ATMs are stated in the following lemmata. Here, X and Y denote strings as defined above and M an arbitrary ATM. Note that n now is not necessarily identical to the length of the input X. Actually, X will in general be much larger than n. But by a repeated application of the following implications we can show that any machine M still obeys a sublogarithmic bound with respect to n.

Lemma 1 (Pumping) Let α, β be memory states with $|\alpha| \leq |\beta| \leq S(n)$, then for any $i, j \in [0 \ \ldots \ |Z_1|\,] \cup [\,|Z_1\, W^n| + 1 \ \ldots \ |X| + 1\,]$ holds:

1. $(\alpha, i) \models_{M,X} (\beta, j) \iff (\alpha, \hat{i}) \models_{M,Y} (\beta, \hat{j})$,
2. $(\alpha, i) \models^{\star}_{M,X} (\beta, j) \iff (\alpha, \hat{i}) \models^{\star}_{M,Y} (\beta, \hat{j})$.

First we apply the above Pumping-Lemma to show that on the long input Y M uses the same space as on the short X.

Lemma 2 (Small Space Bound)
$\mathit{Space}_M(X) \leq S(n) \implies \mathit{Space}_M(Y) = \mathit{Space}_M(X)$.

Proof. Let $\mathit{Space}_M(X) \leq S(n)$. Assume, to the contrary, that $\mathit{Space}_M(Y) \neq \mathit{Space}_M(X)$. We will show that $\mathit{Space}_M(Y) > \mathit{Space}_M(X)$ cannot occur. A similar contradiction can be obtained for the case $\mathit{Space}_M(Y) < \mathit{Space}_M(X)$.

Assume that $\mathit{Space}_M(Y) > \mathit{Space}_M(X)$. Hence, for Y there exists a computation path $\mathcal{C}$ that starts in the initial configuration $(\alpha_0, 0)$ and ends in a configuration $(\alpha, \hat{j})$ with $|\alpha| = \mathit{Space}_M(X)$ such that from $(\alpha, \hat{j})$ M can reach a configuration $(\beta, \hat{j}')$ with $|\beta| = Space_M(X) + 1$ in one step:

$$(\alpha_0, 0) \models^{\star}_{M,Y} (\alpha, \hat{j}) \models^{\star}_{M,Y} (\beta, \hat{j}') .$$

If j fulfills the condition $j \leq |Z_1|$ or $j > |Z_1 W^n|$ of the Pumping Lemma then one can conclude immediately: $(\alpha_0, 0) \models^{\star}_{M,X} (\alpha, j) \models^{\star}_{M,X} (\beta, j')$. Otherwise, using a similar pumping argument one can show that M on input X can reach a configuration $(\alpha, \bar{j})$, in which the input head is located on W^n and reads the same symbol as in $(\alpha, \hat{j})$. Thus it can also get to memory state β in one more step. We get a contradiction since $|\beta| > Space_M(X)$. ∎

A similar property holds for the number of alternations, too. More precisely one can prove that the maximal number of alternations made by M on the input Y is the same as on X if both M uses no more space than $S(n)$ and the number of alternations is bounded by $\exp S(n)$.

3.2 Fooling ATMs

First, we will show that alternating machines can be fooled by pumping the input.

Lemma 3 (1-Alternation) Let (α, i) be an existential configuration for which M with $X = Z_1 W^n Z_2$ on the input tape scans the prefix Z_1 or the suffix Z_2 . Moreover, assume that M starting in this configuration uses never more space than $S(n)$. Then the following implication holds: if M accepts X with one alternation starting in (α, i) then M making at most one alternation also accepts the input Y when it starts in $(\alpha, \hat{i})$. The opposite implication holds for a universal (α, i) configuration.

Note that from the above property follows that if a Σ_2 TM accepts an input of the form 1^n in sublogarithmic space then it accepts $1^{n+n!}$, too. Similarly, if a Π_2 TM accepts $1^{n+n!}$ in sublogarithmic space then it has to accept the short input 1^n .

In the following two lemmata we consider the influence of shifting the input head between identical copies of a fixed string W. For this purpose let us denote the shift distance by $\Delta := |W| \cdot n!$.

Lemma 4 (Configuration Shift) Let

$$X = Z_1 \, W^{n+n!} \, W^s \, W^n \, Z_2$$

be a binary input string of M with $s \geq 1$ and let (α, i) and (β, j) be configurations with $|\alpha| \leq |\beta| \leq S(n)$ such that M in (α, i) scans prefix Z_1 or suffix Z_2 and in (β, j) scans infix W^s. Then, there exists a computation path of M on X that starts in (α, i) and ends in (β, j) if and only if M has a computation path on X that starts in the same configuration and ends in $(\beta, j - \Delta)$.

Below we prove even more. Namely we show that

$$(\alpha, i) \models_{M,X} (\beta, j) \iff (\alpha, i) \models_{M,X} (\beta, j - \Delta) ,$$

i.e. we show in addition that M along both paths either does not alternate or makes only one alternation just in the last step.

Proof. First note that the condition on j the positions j and $j - \Delta$ considered are at least n blocks W away from the boundaries Z_1 and Z_2. Define

$$\begin{aligned} X' &:= Z_1 \, W^n \, W^s \, W^n \, Z_2 \quad \text{and} \\ X'' &:= Z_1 \, W^n \, W^s \, W^{n+n!} \, Z_2 \; . \end{aligned}$$

Set $\hat{i} := i$ if $i \leq |Z_1|$, otherwise $\hat{i} := i - \Delta$. Using the Pumping Lemma twice – first for the input pair X, X' and then for X', X'' – we obtain:

$$(\alpha, i) \models_{M,X} (\beta, j) \iff (\alpha, \hat{i}) \models_{M,X'} (\beta, j - \Delta)$$

The claim of the lemma follows because $X'' = X$. ∎

In the inductive argument for the proof of Theorem 11 we have to guarantee a certain distance of the input head from the boundaries. For this purpose we define

$$m_{k,n} := k \cdot (n + n!) \; .$$

Lemma 5 (Accepting Tree Shift) Let $k \geq 2$, r, s, t be integers with $r, t \geq m_{k,n}$ and $s \geq 1$, and let $Z_1, Z_2, W \in \{0,1\}^*$ be arbitrary strings. Then for an input

$$X = Z_1 \, W^r \, W^s \, W^t \, Z_2$$

and for any configuration (α, i) with the property that M in (α, i) scans infix W^s and starting in this configuration uses never more space than $S(n)$ holds: M accepts in $k-2$ alternations when it starts in (α, i) if and only if M accepts with the same number of alternations when it starts in $(\alpha, i - \Delta)$.

3.3 Halting Computations for ATMs

Let Z be a function. Recall that a binary string X is Z*-bounded* if it contains at most $Z(|X|)$ zeros. The following theorem extends Sipser's space-bounded halting result to alternating TMs.

Theorem 16 *Let S, A, Z be bounds with $A < \infty$ and $Z \leq \exp S$ computable in space S. Then for every S-space-bounded Σ_A TM M there exists a Σ_A TM M' of space complexity S such that for all inputs X*

- *M' accepts X iff M accepts X and X is Z-bounded, and*
- *every computation path of M' on X is finite.*

The identity of Σ_k and co-Π_k for Z-bounded languages (Theorem 14) now follows easily.

4 Alternation Hierarchies

For a subset of the natural numbers $\mathcal{A}$ let $L_{\mathcal{A}}$ be the language over the single letter alphabet $\{1\}$ defined by $1^n \in L_{\mathcal{A}}$ iff $n \in \mathcal{A}$.

Definition 6 Define $L_2 := \{1\}^+$, and for $k \geq 3$ $L_k := (L_{k-1}\,\{0\})^+$.
Let $\mathcal{F}$ be an infinite subset of the natural numbers with the properties:

- $n \in \mathcal{F} \quad \Longrightarrow \quad n + n! \notin \mathcal{F}$ and
- $L_{\mathcal{F}} \in \Pi_2 Space(\mathrm{llog})$ and $\overline{L}_{\mathcal{F}} \in \Sigma_2 Space(\mathrm{llog})$.

Then we define

$L_{\Pi 2} := L_{\mathcal{F}}$ and $L_{\Sigma 2} := \{1\}^+ \cap \overline{L}_{\mathcal{F}}$,
$L_{\Sigma k} := \{w_1 0 w_2 0 \ldots 0 w_p 0 \mid w_i \in L_{k-1}$ and $\exists\, i \in [1...p]$ $w_i \in L_{\Pi k-1}\}$,
$L_{\Pi k} := \{w_1 0 w_2 0 \ldots 0 w_p 0 \mid w_i \in L_{k-1}$ and $\forall\, i \in [1...p]$ $w_i \in L_{\Sigma k-1}\}$.

Note that $L_{\Sigma 2}$ and $L_{\Pi 2}$ are just complementary. For larger k the corresponding languages are "almost" complementary, that means if restricting to strings with a syntactically correct division into subwords by the 0-blocks (more formally $L_{\Pi k} = L_k \cap \overline{L}_{\Sigma k}$). An example for a set $\mathcal{F}$ is the following:

$$\mathcal{F} \;:=\; \{n > 2 \mid \forall\, \ell \in [3\ \ldots\ n-1] \quad F(\ell) < F(n)\}\ ,$$

where $F(n)$ denotes the smallest positive integer that does not divide n . It is easy to show that $\mathcal{F}$ is infinite. The property, $n \in \mathcal{F}$ implies $n + n! \notin \mathcal{F}$, holds since $n + n! \notin \mathcal{F}$ for any integer $n > 2$.

Lemma 6 *For the specific $\mathcal{F}$ defined above with the help of the function F holds*

$$L_{\mathcal{F}} \in \Pi_2 Space(\mathrm{llog}) \quad \text{and} \quad \overline{L}_{\mathcal{F}} \in \Sigma_2 Space(\mathrm{llog})\ .$$

Proof. We describe llog space-bounded Π_2TMs M_Π and Σ_2TMs M_Σ that recognize the language $L_{\mathcal{F}}$, resp. the complement of $L_{\mathcal{F}}$. The machine M_Π verifies the condition $\forall\, \ell \in [3\ \ldots\ n-1]$ $F(\ell) < F(n)$ as follows:

- deterministically it computes $F(n)$ and writes down the binary representation of $F(n)$ on the tape;
- universally it guesses an integer $\ell \in [3 \ldots n-1]$:
  it moves its input head to the right and stops ℓ positions from the right end of the string 1^n ;
- existentially it guesses an integer $k \in [1 \ldots F(n)-1]$ and then moving the input head checks deterministically whether k divides ℓ .
  M_Π accepts if k does not divide ℓ .

The complementary machine M_Σ writes down on the work tape $F(n)$ in binary and tests whether

$$\exists\, \ell \in [3 \ \ldots\ n-1] \quad \forall\, k \in [1 \ \ldots\ F(n)-1] \qquad k \text{ divides } \ell \,.$$

Similarly as in M_Π the input head position represents the integer ℓ . The integer k is stored in binary on the work tape. It is obvious that M_Π recognizes $L_\mathcal{F}$ and that M_Σ recognizes $\overline{L}_\mathcal{F}$ in space $O(\text{llog})$.
∎

Thus, languages $L_\mathcal{F}$ as described in Definition 6 exist. For the base case of the following inductive separation we also need the property that $L_\mathcal{F} \notin \Sigma_2 Space(o(\log))$ and symmetrically that $\overline{L}_\mathcal{F} \notin \Pi_2 Space(o(\log))$. This has been shown for the example above explicitely in [LiRe93a]. Below we will give a general argument showing that this property simply follows from the condition $n \in \mathcal{F}$ and $n + n! \notin \mathcal{F}$.

4.1 ATMs with a Constant Number of Alternations

Lemma 7 For any $k \geq 2$ holds

$$\begin{aligned} L_{\Sigma k} &\in \Sigma_k Space(\text{llog}) \,, \\ L_{\Pi k} &\in \Pi_k Space(\text{llog}) \,. \end{aligned}$$

The proof of these properties is straightforward using the fact that $L_{\mathcal{F}} \in \Pi_2 Space(\mathrm{llog})$ and $\overline{L}_{\mathcal{F}} \in \Sigma_2 Space(\mathrm{llog})$. The separation now follows from the following

Theorem 17 For any $k \geq 2$ holds

$$\begin{aligned} L_{\Sigma k} &\notin \Pi_k Space(o(\log)) \ , \\ L_{\Pi k} &\notin \Sigma_k Space(o(\log)) \ . \end{aligned}$$

We will define specific inputs that belong to $L_{\Sigma k}$ and $L_{\Pi k}$ and show that any sublogarithmic space-bounded machine cannot work correctly on both inputs.

Let $L = L_{\mathcal{F}}$ be fixed. Recall that infinitely many $n \in \mathbb{N}$ exist with $n \in \mathcal{F}$, $1^n \in L$ and $1^{n+n!} \notin L$.

Definition 7 For $n \in \mathcal{F}$ define words

$$\begin{aligned} W_{\Sigma 2}^n &:= 1^{n+n!} \quad \text{and} \quad W_{\Pi 2}^n := 1^n \ , \quad \text{and for } k \geq 3 \\ W_{\Sigma k}^n &:= \left[W_{\Sigma k-1}^n \ 0\right]^{m_{k,n}} W_{\Pi k-1}^n \ 0 \left[W_{\Sigma k-1}^n \ 0\right]^{m_{k,n}} \ , \\ W_{\Pi k}^n &:= \left[W_{\Sigma k-1}^n \ 0\right]^{m_{k,n}} W_{\Sigma k-1}^n \ 0 \left[W_{\Sigma k-1}^n \ 0\right]^{m_{k,n}} \ , \end{aligned}$$

where the $m_{k,n}$ have been defined in the Accepting-Tree-Shift-Lemma.

From the definition follows easily

Lemma 8 For $k \geq 2$ and every $n \in \mathcal{F}$

$$\begin{aligned} W_{\Sigma k}^n \in L_{\Sigma k} \quad &\text{and} \quad W_{\Sigma k}^n \notin L_{\Pi k} \ , \\ W_{\Pi k}^n \in L_{\Pi k} \quad &\text{and} \quad W_{\Pi k}^n \notin L_{\Sigma k} \ . \end{aligned}$$

Let $k \geq 2$ and $S \in$ `SUBLOG` be a space bound. We will prove Theorem 17 by showing that if a Σ_k TM M accepts $L_{\Pi k}$ in space S then for sufficiently large $n \in \mathcal{F}$ M accepts $W_{\Sigma k}^n$, too. Similarly, if a Π_k TM M accepts $L_{\Sigma k}$ in space S then for large $n \in \mathcal{F}$ it accepts $W_{\Pi k}^n$ and hence makes a mistake. Recall that $\mathcal{N}_{M,S}$ denotes the constant defined for M and S above.

Proposition 1 Let $S \in o(\log)$ and M be an ATM. Then for any $k \geq 2$, for all $n \geq \mathcal{N}_{M,S}$, for all strings $U, V \in \{0,1\}^*$, and for any configuration (α, i) with

- $i \leq |U|$ or $i > |U\, W^n_{\Pi k}|$ and
- $Space_M(\alpha, i, UW^n_{\Pi k}V) \leq S(n)$ and $Space_M(\alpha, \hat{\imath}, UW^n_{\Sigma k}V) \leq S(n)$

holds:

$$\begin{array}{lll} \mathtt{acc}^k_M(\alpha, i, U\, W^n_{\Pi k}\, V) & \Longrightarrow & \mathtt{acc}^k_M(\alpha, \hat{\imath}, U\, W^n_{\Sigma k}\, V) \text{ if } \alpha \text{ is existential,} \\ \mathtt{acc}^k_M(\alpha, \hat{\imath}, U\, W^n_{\Sigma k}\, V) & \Longrightarrow & \mathtt{acc}^k_M(\alpha, i, U\, W^n_{\Pi k}\, V) \text{ if } \alpha \text{ is universal.} \end{array}$$

Proof. Remember that $\hat{\imath}$ was defined as

$$\hat{\imath} \;=\; \begin{cases} i & \text{if } i \leq |U|, \\ i + (|W^n_{\Sigma k}| - |W^n_{\Pi k}|) & \text{if } i > |U\, W^n_{\Pi k}|. \end{cases}$$

For $k = 2$ the implications above follow from the 1-Alternation Lemma. To establish the proposition for $k > 2$ we consider the first time when the machine M makes an alternation and inductively use the corresponding properties for the strings $W^n_{\Sigma k-1}$ and $W^n_{\Pi k-1}$. The argument concentrates only on the block in the middle of a $W^n_{\Sigma k}$ string, which is a $W^n_{\Pi k-1}$ word, and analogously for $W^n_{\Pi k}$ strings with a $W^n_{\Sigma k-1}$ word in the middle. The main technical difficulty for the following argument is the possibility that in an accepting computation the machine may just make its first alternation in the middle block, and therefore may notice the difference between the $W^n_{\Sigma k}$ and $W^n_{\Pi k}$ strings. But the Configuration- and Accepting-Tree-Shift-Lemmata imply that there also exist accepting computations with the first alternation outside this critical region. For details see [LiRe93b]. ∎

Next, we will show that the second requirement of the proposition above is always fulfilled.

Proposition 2 Let $k \geq 2$ and M be an ATM of space complexity S with $S \in o(\log)$. Then there exists a bound $S' \in o(\log)$ such that for all $n \geq \mathcal{N}_{M,S'}$

$$Space_M(W^n_{\Pi k}) \leq S'(n) \quad \text{and} \quad Space_M(W^n_{\Sigma k}) \leq S'(n)\ .$$

Proof. The idea of the proof is as follows. If in $W^n_{\Pi k}$ and $W^n_{\Sigma k}$ all substrings generated in the recursive construction which are multiplies of $n!$, are cancelled, then the remaining word has a length $p_k(n)$, which is polynomial in n. Using the Small-Space-Bound-Lemma, which shows that a sublogarithmic space-bounded machine M does not notice a difference when an arbitrary block of the input is added $n!$ times, it follows that M must obey a space bound $S(p_k(n))$ on $W^n_{\Pi k}$ and $W^n_{\Sigma k}$. If S grows sublogarithmically in n so does $S(p_k(n))$. For the technical details of this proof see [LiRe93b]. ∎

Now we are ready to prove Theorem 17. Let us assume that M is a Σ_k TM accepting $L_{\Pi k}$ in sublogarithmic space S. By Proposition 2 there exists a function $S' \in o(\log)$ such that for any $n \geq \mathcal{N}_{M,S'}$

$$Space_M(W^n_{\Pi k}) \leq S'(n) \quad \text{and} \quad Space_M(W^n_{\Sigma k}) \leq S'(n) \; .$$

Let n with $n \in \mathcal{F}$ be an integer larger than $\mathcal{N}_{M,S'}$ (such an n exists since $\mathcal{F}$ is infinite). By Lemma 8 $W^n_{\Pi k} \in L_{\Pi k}$, hence M has to accept $W^n_{\Pi k}$, which means that $\mathtt{acc}^k_M(\alpha_0, 0, W^n_{\Pi k})$ is true, where $(\alpha_0, 0)$ is the initial configuration of M. From Proposition 1 we conclude that $\mathtt{acc}^k_M(\alpha_0, 0, W^n_{\Sigma k})$ holds, too, and hence M accepts $W^n_{\Sigma k}$, which by Lemma 8 does not belong to $L_{\Pi k}$ – a contradiction.

In the same way one shows that if M is a Π_k TM that accepts $L_{\Sigma k}$ in space S then M accepts $W^n_{\Pi k}$.

4.2 Unbounded Number of Alternations

In the previous section we have proved the lower space bounds for recognizing $L_{\Sigma k}$ and $L_{\Pi k}$ on ATMs with a constant number of alternation. These results hold for all languages $L_{\Sigma k}$ and $L_{\Pi k}$ defined on the base of a subset of natural numbers $\mathcal{F}$ with the properties as in Definition 6. In this section we fix the set $\mathcal{F}$ to the example given at the beginning of this section:
$\mathcal{F} \;:=\; \{n > 2 \mid \forall\, \ell \in [3 \;\ldots\; n-1] \quad F(\ell) < F(n)\}$.

Definition 8 Let $A : \mathbb{N} \to \mathbb{N}$ be a function with $A(n) \geq 2$ for all n, and define

$$\begin{aligned} L_{\Sigma}(A) &:= \{X \mid X = W0^r \text{ with } W \in L_{\Sigma k} \text{ and } k \leq A(|X|),\} \\ L_{\Pi}(A) &:= \{X \mid X = W0^r \text{ with } W \in L_{\Pi k} \text{ and } k \leq A(|X|).\} \end{aligned}$$

The separating results for A-alternation-bounded space classes (Theorem 12) follow from the propositions below.

Lemma 9 For any $S \in \texttt{SUBLOG}$ and all functions $A \geq 2$ computable in space S holds:

$$\begin{aligned} L_{\Sigma}(A) &\in \Sigma_A Space(S) , \\ L_{\Pi}(A) &\in \Pi_A Space(S) . \end{aligned}$$

Proof. On input $X = W0^r$ the machine first computes $a := A(|X|)$ and initializes a counter with that value. It remains to check whether $W \in L_{\Sigma k}$ for some $k \leq a$. This can be done similarly as in the case for fixed k, decrementing the counter each time an alternation has been performed. ∎

For functions $A, B : \mathbb{N} \to \mathbb{N}$ let $A \leq_* B$ denote that $A(m) \leq B(m)$ for all $m \in \mathbb{N}$ with equality for infinitely many m.

Proposition 3 For any $S \in \texttt{SUBLOG}$ and for all functions A and B with $1 < A \leq_* B$ and $B \cdot S \in o(\log)$ holds:

$$\begin{aligned} L_{\Sigma}(A) &\notin \Pi_B Space(S) , \\ L_{\Pi}(A) &\notin \Sigma_B Space(S) . \end{aligned}$$

5 Closure Properties

In this section we discuss closure properties of the classes $\Sigma_k Space(S)$ and $\Pi_k Space(S)$ for sublogarithmic bounds S. First for any integer $k \geq 2$ we define the languages $A_{\Sigma k} := L_k \{0\} L_{\Sigma k}$, $B_{\Sigma k} :=$

$L_{\Sigma k}\,\{0\}\,L_k$, and symmetrically $A_{\Pi k} := L_k\,\{0\}\,L_{\Pi k}$, $B_{\Pi k} := L_{\Pi k}\{0\}L_k$. It is easy to see that

$$A_{\Sigma k}, B_{\Sigma k} \in \Sigma_k Space(\text{llog}) \quad \text{and} \quad A_{\Pi k}, B_{\Pi k} \in \Pi_k Space(\text{llog}) \ . \qquad (1)$$

Proposition 4 For all $k \geq 2$ holds:

$$\begin{aligned} A_{\Sigma k} \cap B_{\Sigma k} &\in \Pi_{k+1} Space(\text{llog}) \setminus \Sigma_{k+1} Space(o(\log)), \\ A_{\Pi k} \cup B_{\Pi k} &\in \Sigma_{k+1} Space(\text{llog}) \setminus \Pi_{k+1} Space(o(\log)). \end{aligned}$$

Proof. It is well known that for any function S the classes $\Sigma_k Space(S)$ are closed under union, and symmetrically the $\Pi_k Space(S)$ are closed under intersection. Hence, $A_{\Sigma k} \cap B_{\Sigma k} \in \Pi_{k+1} Space(\text{llog})$ and $A_{\Pi k} \cup B_{\Pi k} \in \Sigma_{k+1} Space(\text{llog})$. To prove that $A_{\Sigma k} \cap B_{\Sigma k} \notin \Sigma_{k+1} Space(o(\log))$ and $A_{\Pi k} \cup B_{\Pi k} \notin \Pi_{k+1} Space(o(\log))$ first we modify Proposition 2 in the following way:

Proposition 2′ Let $k \geq 2$ and M be an ATM of space complexity S with $S \in o(\log)$. Then there exists a bound $S'' \in o(\log)$ such that for all $n \geq \mathcal{N}_{M,S''}$ and words $W_1, W_2 \in \{W^n_{\Sigma k}\ ,\ W^n_{\Pi k}\}$

$$Space_M(W_1\,0\,W_2) \leq S''(n) \ .$$

Let us assume, to the contrary, that $A_{\Sigma k} \cap B_{\Sigma k} \in \Sigma_{k+1} Space(S)$, for some $S \in o(\log n)$. Let M be an S space-bounded Σ_{k+1} TM for $A_{\Sigma k} \cap B_{\Sigma k}$. Choose $n \in \mathcal{F}$ sufficiently large. By Lemma 8 $W^n_{\Sigma k} \in L_{\Sigma k}$ hence M has to accept $X = W^n_{\Sigma k}\,0\,W^n_{\Sigma k}$, which means that there exists an existential computation path starting in initial configuration $(\alpha_0, 0)$ and ending in a universal configuration (β, j) , with

$$(\alpha_0, 0) \models_{M,X} (\beta, j) \quad \text{and} \quad \mathtt{acc}^k_M(\beta, j, X) \ . \qquad (2)$$

(The trivial case that M accepts X without alternation could be handled similarly.) Now let $Y_1 := W^n_{\Sigma k}\,0\,W^n_{\Pi k}$ and $Y_2 := W^n_{\Pi k}\,0\,W^n_{\Sigma k}$. By Proposition 2′ there exists $S'' \in o(\log n)$ such that

$$Space_M(X),\ Space_M(Y_1),\ Space_M(Y_2) \quad \leq \quad S''(n) \ .$$

Therefore, applying Claim 1 (from the Proof of Proposition 1) and Proposition 1 to the situation described in (2) we obtain

$$(\alpha_0, 0) \models_{M,Y_1} (\beta, j) \quad \text{and} \quad \mathtt{acc}^k_M(\beta, j, Y_1)$$

if $j \leq |W^n_{\Sigma k}\, 0|$ and otherwise

$$(\alpha_0, 0) \models_{M,Y_2} (\beta, \hat{j}) \quad \text{and} \quad \mathtt{acc}^k_M(\beta, \hat{j}, Y_2) ,$$

where $\hat{j} = j + |Y_2| - |X|$. Hence M also accepts input Y_1 or Y_2. This yields a contradiction since, by Lemma 8, $Y_1, Y_2 \notin A_{\Sigma k} \cap B_{\Sigma k}$.

Similarly, one can show that if a Π_{k+1} TM accepts $A_{\Pi k} \cup B_{\Pi k}$ within space $S \in o(\log n)$, then it has to reject X, but it also rejects input Y_1 or Y_2, which both belong to $A_{\Pi k} \cup B_{\Pi k}$ – a contradiction! ∎

This result can be applied to prove Theorem 13. For all $k \geq 2$ and any $S \in \mathtt{SUBLOG}$ holds:

1. $\Sigma_k \mathit{Space}(S)$ and $\Pi_k \mathit{Space}(S)$ are not closed under complementation.
2. $\Sigma_k \mathit{Space}(S)$ is not closed under intersection,
3. $\Pi_k \mathit{Space}(S)$ is not closed under union.
4. $\Sigma_k \mathit{Space}(S)$ and $\Pi_k \mathit{Space}(S)$ are not closed under concatenation.

(1) follows immediately from Lemma 7, Theorem 17 and the following equations: $L_{\Sigma k} = L_k \cap \overline{L}_{\Pi k}$, and $L_{\Pi k} = L_k \cap \overline{L}_{\Sigma k}$, where L_k is the regular language introduced in Definition 6.

We know $A_{\Sigma k}, B_{\Sigma k} \in \Sigma_k \mathit{Space}(\mathrm{llog})$ and $A_{\Pi k}, B_{\Pi k} \in \Pi_k \mathit{Space}(\mathrm{llog})$. On the other hand, from Proposition 4 $A_{\Sigma k} \cap B_{\Sigma k} \notin \Sigma_{k+1} \mathit{Space}(o(\log))$ and $A_{\Pi k} \cup B_{\Pi k} \notin \Pi_{k+1} \mathit{Space}(o(\log))$. This proves 2. and 3.

Property 4. for Σ_k classes follows from the fact that for any $k \geq 2$ $L_{\Sigma k}\, \{0\}\, L_{\Sigma k} = A_{\Sigma k} \cap B_{\Sigma k}$ does not belong to $\Sigma_k \mathit{Space}(o(\log))$, but $L_{\Sigma k} \in \Sigma_k Space(\text{llog})$. To see that $\Pi_k Space(S)$ is not closed under concatenation define the languages $L_k^1 := L_k \cup \{\varepsilon\}$, where ε denotes the empty string and

$$L_k^2 \quad := \quad \{w_1 0 w_2 0 \ldots 0 w_p 0 \mid p \in \mathbb{N},\ w_i \in L_{k-1} \ \text{ and } \ w_1 \in L_{\Pi k-1}\} \ .$$

Obviously, both languages belong to $\Pi_k Space(\text{llog})$, but from Theorem 17 follows $L_k^1\, L_k^2 = L_{\Sigma k} \notin \Pi_k Space(o(\log))$.

6 Lower Bounds for Context-Free Languages

First, we will briefly sketch the proof for the last claim in Theorem 6 that

$$\overline{\texttt{COUNT}} \ \notin \ ASpace(o(\log)) \ .$$

Assume, to the contrary, that $\overline{\texttt{COUNT}}$ is recognized by an S space-bounded ATM A for some $S \in o(\log)$. For $S'(n) \ := \ S(2n+1)$, obviously holds $S' \in o(\log)$. Let $\hat{n} \ := \mathcal{N}_{M,S'}$. Then, the Small-Space-Bound-Lemma implies that for all $k, \ell \geq 0$

$$Space_A(1^{\hat{n}}01^{\hat{n}}) \ = \ Space_A(1^{\hat{n}+k\hat{n}!}01^{\hat{n}+\ell\hat{n}!}). \qquad (3)$$

Let $\hat{s} = Space_A(1^{\hat{n}}01^{\hat{n}})$. For this fixed $\hat{n}$ we define the following language

$$\hat{L} \ := \ \{1^{\hat{n}+k\hat{n}!}01^{\hat{n}+\ell\hat{n}!} \mid k, \ell \in \mathbb{N} \text{ and } k \neq \ell\} \ ,$$

and construct an automaton $\hat{A}$ that recognizes $\hat{L}$. $\hat{A}$ performs the following algorithm: it checks deterministically if the input X has the form $1^{\hat{n}+k\hat{n}!}01^{\hat{n}+\ell\hat{n}!}$ for some integers k and ℓ; it rejects and stops if not. Then it moves the head to the first symbol of the input and starts to simulate the machine A .

It is obvious that $\hat{A}$ accepts an input $X = 1^{\hat{n}+k\hat{n}!}01^{\hat{n}+\ell\hat{n}!}$ iff A accepts X . Hence $L(\hat{A}) = \hat{L}$. It is easy to see that step 1 can be performed within space $O(\log \hat{n}!)$, which is a constant. Moreover from (3) follows that step 2 also requires only constant space $\hat{s}$. Hence, $\hat{A}$ recognizes $\hat{L}$ within constant space, which yields a contradiction to $\hat{L}$ being non-regular. Similarly, one can show that also the language `COUNT` is not in $ASpace(o(\log))$.

The structure of a bounded language L can equivalently be represented using a finite alphabet $\{a_1, \ldots, a_r\}$. Then L is a subset of $\{a_1\}^* \ldots \{a_r\}^*$.

Definition 9 Let $V(L)$ denote the set $\{(v_1, \ldots, v_r) \in \mathbb{N}^r \mid a_1^{v_1} \ldots a_r^{v_r} \in L\}$. Sets of the form $\{\alpha + n_1\beta_1 + \ldots + n_k\beta_k \mid n_1, \ldots, n_k \in \mathbb{N}\}$ with $\alpha, \beta_1, \ldots, \beta_k \in \mathbb{N}^r$, are called *linear sets.* A finite union of linear sets is a *semilinear set.* A language L is *semilinear* if $L \subseteq \{a_1\}^* \ldots \{a_r\}^*$ and $V(L)$ is a semilinear set.

Theorem 15 can then be proved with the help of Theorem 16 and the following

Proposition 5 Let a language $L \subseteq \{a_1\}^* \ldots \{a_r\}^*$ be semilinear and assume that $L, \overline{L} \in ASpace(S)$ for some $S \in o(\log)$. Then L is regular.

Combining Theorem 15 with the other upper and lower bounds on nonregular context-free languages the situation concerning space complexity looks as follows. The general upper bound is $DSpace(\log^2)$ and the best lower bound $NSpace(\Omega(\text{llog}))$, that means llog is the minimal amount of nondeterministic space necessary and for some language this is also sufficient. Restricting to the subclass of deterministic context-free languages, however, the lower bound increases to $AAlterSpace(\text{CON}, \Omega(\log))$. Thus, even ATMs with a constant number of alternations now require at least logarithmic space.

7 Adding Randomness

A Turing machine to a certain extent can compensate little space by random bits if one is willing to tolerate a small probability of error. Let $BPSpace(S)$ denote the class of languages that can be recognized by a probabilistic TM in space S with error probability at most $1/4$.

Theorem 18 [Fr81] $BPSpace(\text{CON})$ *contains nonregular languages.*

`COUNT` is an example which can be recognized by the following experiment that has to be repeated a lot of times till a significant number of successes has occured: Separately for the two block of ones of an input X, toss a coin for each symbol in the block. Count this as a success for the block if all coins show head. Accept X if the number of successes of both blocks are about equal. Note that this strategy requires exponential (expected) time, whereas nonprobabilistic machines even with a logarithmic space bound can be assumed to be polynomial time bounded.

However, the counting abilities of probabilistic machine are limited as has been shown recently by Freivalds and Karpinski using the language `PALINDROM` of all strings that are palindromes.

Theorem 19 [FrKa94] $\texttt{PALINDROM} \notin BPSpace(o(\log))$.

What happens in the sublogarithmic space world if we combine nondeterminism and probabilism? If nondeterministic machines are extended with the ability to perform random moves one obtains the stochastic Turing machine model (STM) as introduced by Papadimitriou [Pa85]. Alternative characterizations can be given by interactive proof systems of Goldwasser, Micali and Rackoff [GMR89] and Arthur-Merlin-games of Babai [Ba85], see also [GoSi86] and [Co89]. In a probabilistic state a stochastic machine M chooses among the successor configurations with equal probability.

As for alternating machines, a computation of M on an input X can be described by a computation tree. To define acceptance of X, for each existential configuration one chooses an successor that maximizes the probability to reach an accepting leaf. The acceptance probability of X is then given by the acceptance probability of the starting configuration in this truncated tree. M accepts a language L in space S if

- for all $X \in L$, the probability that M accepts X is more than $3/4$,
- every $X \notin L$ is accepted with probability less than $1/4$, and
- M never uses more than $S(|X|)$ space.

Definition 10 Let $MA_kSpace(S)$ (resp. $AM_kSpace(S)$) denote the set of languages that can be accepted by such a machine in space S making at most $k-1$ alternations between existential and probabilistic configurations and starting in existential (resp. probabilistic) mode. For such a machine we also say that it works in space S and k rounds.

Dwork and Stockmeyer were the first who investigated interactive proof systems with small space bounded verifiers [DwSt92]. The separation results presented in [DwSt92] are stated for constant space, but they can be extended to any sublogarithmic bound (see e.g. [Co93]).

It has been shown that for sublogarithmic space bounds it makes a difference whether the random moves are known, as it is the case for a STM, resp. for Arthur-Merlin games, or whether they are hidden as in interactive proof systems. `PALINDROM` is an example that can easily be recognized by an interactive proof system with a constant space-bounded verifier and hidden random moves, but requires logarithmic space otherwise (see [DwSt92] and [Co93]).

This property does not hold for the corresponding polynomial time classes $AM_kTime(\mathrm{POL})$. Furthermore, it has been shown [Ba85] that any number of rounds can be reduced to two rounds, that is

$$MA_2Time(\mathrm{POL}) \subseteq AM_2Time(\mathrm{POL}) = AM_{\mathrm{CON}}Time(\mathrm{POL}) .$$

One might expect that for at least logarithmic space bounds similar to ATMs the alternating stochastic hierarchy collapses to some level. On the other hand we have obtained some evidence that for space bounds $S \in$ `SUBLOG` the situation is different.

Extending an impossibility result for 2-way probabilistic finite automata one can show that the language

$$\texttt{CENTER} \quad := \quad \{w0x \mid w, x \in \{0,1\}^* \quad \text{and} \quad |w| = |x|\}$$

cannot be recognized by a sublogarithmic space bounded probabilistic Turing machine with any error probability $\epsilon < \frac{1}{2}$ [FrKa94]. However, there exists a constant space interactive proof system for this language. Hence, for any S in SUBLOG languages CENTER and PALINDROME yield the following separations:

$$BPSpace(S) = AM_1Space(S) \; \subset \; AMSpace(S) \; \subset \; AMSpace(\log) \; = \; \mathcal{P} \; .$$

The last equivalence is due to Condon [Co89].

A sublogarithmic STM may require exponential expected time. It is shown in [DwSt92] that some languages cannot be recognized by such machines faster – CENTER is one of such example. On the other hand, the power of sublogarithmic space bounded STMs restricted to polynomial expected time is still an open problem. In this case we do not even know if stochastic finite automata can recognize nonregular languages. Let $\texttt{BIN}(m) \; := \; \mathrm{bin}(0)\#\,\mathrm{bin}(1)\#\,\mathrm{bin}(2)\#\ldots\#\,\mathrm{bin}(m)$, and define

$$\texttt{PATTERN} \quad := \quad \{\texttt{BIN}(2^k) * w_1 * w_2 * \ldots * w_t * u \mid w_1, \ldots, w_t \in \{0,1\}^k \text{ for some } t, k \in \mathbb{N} \text{ and } \; u \in \{w_1, \ldots, w_t\} \; \} \; .$$

Combining the proof methods described above with some of [DwSt92] for probabilistic machines we can show

Lemma 10 [LiRe95] For $S \in o(\log)$, an S space-bounded STM cannot recognize PATTERN in 2 rounds starting in probabilistic mode, that is

$$\texttt{PATTERN} \;\; \notin \; AM_2Space(o(\log)) \; .$$

On the other hand, it is not hard to see that switching the quantifiers the problem can be solved within small space.

Lemma 11 [LiRe95] For arbitrary small $\epsilon > 0$, there exists an llog space-bounded STM M for `PATTERN` with error probability ϵ that works in 2 rounds starting in nondeterministic mode, that is

$$\texttt{PATTERN} \in MA_2Space(\text{llog}) .$$

Moreover, M works in polynomial time.

Therefore, we obtain the following separation

Theorem 20 $\quad MA_2Space(\text{llog}) \not\subseteq AM_2Space(o(\log)) .$

We conjecture that the alternation hierarchy for sublogarithmic space-bounded STMs is infinite, similar as for standard ATMs, that means

$$AM_1Space(S) \subset AM_2Space(S) \subset AM_3Space(S) \subset \ldots$$

It is interesting to notice ([LiRe95]) that the $AM_kSpace(S)$ and $MA_kSpace(S)$ classes do not seem to be complementary, since

$$\overline{\texttt{PATTERN}} \notin AM_2Space(o(\log)) .$$

8 Conclusions

Right now it seems that most phenomena about the sublogarithmic space world have been understood. Still, there are some open questions. It would be nice to exactly locate the classes $ASpace(S)$ and $AAlterSpace(\text{CON}, S)$ for $S \in \texttt{SUBLOG}$. Are they contained in $\mathcal{SC}$? Is it true that they do not belong to $\mathcal{NC}^1$ and $\mathcal{L} = \mathcal{SC}^1$?

What is the relevance of the alternating sublogarithmic space hierarchy? Let us denote it by

$$\mathcal{ASL} := AAlterSpace(\text{CON}, \text{llog}) .$$

Several other alternating hierarchies have been defined in the past. The alternating logarithmic time hierarchy

$$\mathcal{AC}^0 = AAlterTime(\text{CON}, O(\log))$$

is also known to consist of an infinite number of distinct levels. It is contained in $\mathcal{NC}^1$ and hence in $\mathcal{SC}^1$. At the moment $\mathcal{ASL}$ seems to be the most powerful sequence of alternating complexity classes, for which one can prove a strict inclusion without using relativations or unproven assumptions. However, $\mathcal{AC}^0$ and $\mathcal{ASL}$ are not comparable. The languages `COUNT` and `PARITY` are somehow extremal examples showing that neither class is contained in the other.

What is the relation between the two lowest levels of $\mathcal{ASL}$? Is the class $\Sigma_1 Space(\text{llog})$ different from $DSpace(\text{llog})$ and from $\Pi_1 Space(\text{llog})$? The inability to record input head positions seems to be crucial if one wants to separate these classes. For a stronger notion of machine simulation that exploits this fact Kannan and Szepietowski have shown that such a kind of simulation of a nondeterministic TM is not possible within the same sublogarithmic space bound for a deterministic or co-nondeterministic machine [Ka83, Sz89].

Furthermore, characterize the alternating sublogarithmic space hierarchy for bounded languages completely. For the class of unbounded languages find out the exact relationship between co-$\Sigma_k Space(S)$ and $\Pi_k Space(S)$ for sublogarithmic space bounds S.

Finally, concerning small space probabilistic classes is it true that – as for probabilistic polynomial time – machines with bounded error can be simulated nonuniformly in deterministic space? It is not obvious how to achieve exponentially small error probability in this case. Furthermore, since

$$\mathcal{NL} \;=\; MA_1 Space(\log) \;\subseteq\; AM_1 Space(\log) = \mathcal{BPL}$$

one can easily show that

$$AM_2 Space(\log) \;=\; AM_1 Space(\log)\;,$$

that means the AM_2-class is quite weak in case of space bounds – contrary to time bounded classes. Is it also true that

$$AM_2 Space(\texttt{SUBLOG}) \;=\; AM_1 Space(\texttt{SUBLOG})\;?$$

References

[Al79] H. Alt, *Lower bounds on space complexity for context-free recognition,* Acta Inform. 12, 1979, 33-61.

[AGM92] H. Alt, V. Geffert, and K. Mehlhorn, *A lower bound for the nondeterministic space complexity of context-free recognition,* Inform. Process. Lett. 42, 1992, 25-27.

[AlMe76] H. Alt, and K. Mehlhorn, *Lower bounds for the space complexity of context free recognition,* Proc. 3rd ICALP, 1976, 339-354.

[Ba85] L. Babai, *Trading group theory for randomness,* in Proc. 17th Ann. ACM Symp. on Theory of Computing, 1985, 421-429.

[Br93] B. von Braunmühl, *Alternation for two-way machines with sublogarithmic space*, Proc. 10. STACS, Würzburg, 1993, 5-15.

[BDG90] J. Balcazar, J. Diaz, J. Gabarro, *Structural Complexity I and II,* Springer, 1990.

[BGR93] B. von Braunmühl, R. Gengler, and R. Rettinger *The alternation hierarchy for machines with sublogarithmic space is infinite,* Research Report, Universität Bonn, January, 1993.

[CIRB87] J. Chang, O. Ibarra, B. Ravikumar, and L. Berman, *Some observations concerning alternating Turing machines using small space,* Inform. Proc. Letters 25, 1987, 1-9.

[CKS81] A. Chandra, D. Kozen, L. Stockmeyer, *Alternation,* J. ACM 28, 1981, 114-133.

[Co89] A. Condon, *Computational model of games,* MIT Press, 1989.

[Co93] A. Condon, *The Complexity of Space Bounded Interactive Proof Systems,* in Complexity Theory: Current Research, S. Homer, U. Schöning, K. Ambos-Spies (Ed.), Cambridge Univ. Press, 1993, 147-190.

[DwSt92] S. Dwork, L. Stockmeyer, *Finite state verifiers I: the power of interaction,* J. ACM, 39 (1992), 800-828.

[FrLa75] A. Freedman, R. Ladner, *Space bounds for processing contentless inputs,* J. CSS 11, 1975, 118-128.

[Fr81] R. Freivalds, *Probabilistic 2-way machines,* Proc. 10. MFCS, 1981, 33-45.

[FrKa94] R. Freivalds, M. Karpinski, *Lower space bounds for randomized computation,* Proc. 21. ICALP, 1994, 580-592.

[Ge91] V. Geffert, *Nondeterministic computations in sublogarithmic space and space constructability,* SIAM J. Comput. 20, 1991, 484-498.

[Ge93a] V. Geffert, *Sublogarithmic Σ_2 -space is not closed under complement and other separation results,* Theoretical Informatics and Applications 27, 1993, 349-366.

[Ge93b] V. Geffert, *Tally version of the Savitch and Immerman-Szelepcsényi theorems for sublogarithmic space,* SIAM J. Comput. 22, 1993, 102-113.

[Ge93c] V. Geffert, *A hierarchy that does not collapse: alternations in low level space,* manuscript.

[Gi72] S. Ginsburg, *The mathematical theory of context-free languages,* McGraw-Hill, 1972.

[GMR89] S. Goldwasser, S. Micali, C. Rackoff, *The Knowledge Complexity of Interactive Proof Systems,* SIAM J. Comp. 18, 1989, 186-208.

[GoSi86] S. Goldwasser, M. Sipser, *Private coins versus public coins in interactive prove systems,* Proc. 18. STOC, 1986, 59-68.

[HaBe76] J. Hartmanis, L. Berman, *On tape bounds for single-letter alphabet processing,* TCS 3, 1976, 213-224.

[HoUl69] J. Hopcroft, J. Ullman, *Some results on tape-bounded Turing machines,* J. ACM 16, 1969, 168-177.

[Im88] N. Immerman, *Nondeterministic space is closed under complementation,* SIAM J. Comput. 17, 1988, 935-938.

[Iw93] K. Iwama, *ASpace(o(loglog n)) Is Regular,* SIAM J. Comput. 22, 1993, 136-146.

[Ka83] R. Kannan, *Alternation and the Power of Nondeterminism,* Proc. 15. SToC, 1983, 344-346.

[LiRe93a] M. Liśkiewicz, and R. Reischuk, *Separating the lower levels of the sublogarithmic space hierarchy,* Proc. 10. STACS, Würzburg, 1993, 16-27.

[LiRe93b] M. Liśkiewicz, and R. Reischuk, *The sublogarithmic space world,* Technical Report 048-93 ICSI Berkeley, to appear in SIAM J. Computing.

[LiRe95] M. Liśkiewicz, and R. Reischuk, *Space bounds for interactive proof systems with public coins and bounded number of rounds*, Technical Report, Med. Universität zu Lübeck, 1995.

[LSH65] P. Lewis, R. Stearns, J Hartmanis, *Memory bounds for the recognition of context free and context sensitive languages,* Proc. IEEE Conf. Switching Circuit Theory and Logical Design, 1965,191-202.

[MoSu82] B. Monien, H. Sudborough, *On elmiminating nondeterminism from Turing machines which use less than logarithm worktape space,* TCS 21, 1982, 237-253.

[Pa85] C. Papadimitriou, *Games against nature,* J. Comput. System Sci, 31 (1985) 288-301.

[Si80] M. Sipser, *Halting space-bounded computations,* Theoret. Comput. Sci. 10, 1980, 335-338.

[Re90] R. Reischuk, *Einführung in die Komplexitätstheorie,* Teubner, 1990.

[SHL65] R. Stearns, J. Hartmanis, R. Lewis, *Hierarchies of memory limited computations,* Proc. IEEE Conf. Switching Circuit Theory and Logical Design, 1965,179-190.

[St67] R. E. Stearns, *A regularity test for pushdown-machines,* Information and Control 11, 1967, 323-340.

[Su80] I. Sudborough, *Efficient algorithms for path system problems and applications to alternating and time-space complexity classes,* Proc. 21. FoCS, 1980, 62-73.

[Sz88] R. Szelépcsenyi, *The method of forced enumeration for nondeterministic automata,* Acta Informatica 26, 1988, 279-284.

[Sz89] A. Szepietowski, *Some remarks on the alternating hierarchy and closure under complement for sublogarithmic space,* IPL 33, 1989, 73-78.

[Sz90] A. Szepietowski, *If deterministic and nondeterministic space complexities are equal for loglog n then they are also equal for log n,* TCS 74, 1990, 73-78.

[Sz94] A. Szepietowski, *Turing machines with sublogarithmic space,* Springer Lec. Notes in Computer Science 843, 1994.

[To81] M. Tompa, *An extension of Savitch's Theorem to small space bounds,* IPL 12, 1981, 106-108.

[Wa93] K. Wagner, Editorial note: *The alternation hierarchy for sublogarithmic space: an exciting race to STACS'93,* Proc. 10. STACS, Würzburg, 1993, 2-4.

[WaWe86] K. Wagner, and G. Wechsung, *Computational complexity,* Reidel, 1986.

Bildkompression mit endlichen Automaten

Andreas Schrader

Universität-Gesamthochschule Siegen
Fachbereich 12 - Elektrotechnik und Informatik
D–57068 Siegen

schrader@informatik.uni-siegen.de

Zusammenfassung

Das 'multimediale Zeitalter' hat zu einer ständig wachsenden Integration von Bilddaten bei der Übertragung und Speicherung von Informationen geführt. Die Begrenzungen der Kapazitäten in Übertragungskanälen und Speichermedien erforderte dabei schon früh die Entwicklung geeigneter Kompressionsalgorithmen.

In den klassischen Verfahren der ersten Generation werden Kenntnisse oder Vermutungen über die statistische Verteilung der Daten zur Konstruktion eines redundanzreduzierenden Codes verwendet. Der zweidimensionale Charakter der Bilddaten bleibt dabei im wesentlichen unberücksichtigt.

In diesem Artikel wird die Codierung von endlichen diskreten Bilddaten mit Hilfe von endlichen Automaten beschrieben. Der Bildinhalt wird dabei als endliche Sprache interpretiert, die sich aufgrund von zweidimensionalen rekursiven Selbstähnlichkeiten ergibt und ein deterministischer endlicher Automat als Akzeptor dieser Sprache konstruiert. Dabei werden einige heuristische Ansätze zur effizienten

Speicherung der Automatentabelle mit Methoden der ersten Generation vorgestellt und die Ergebnisse dieses Ansatzes anhand von Bildbeispielen demonstriert.

1 Einführung

Die technische Behandlung von Bildern in Rechnersystemen erfordert eine Beschreibung des Bildinhaltes durch eine geeignete digitale Repräsentation. Dies kann zum Beispiel die serielle Angabe der diskreten Intensitäten der Helligkeitswerte aller Pixel sein (Bitmap); eine andere Möglichkeit besteht in der Angabe der grafischen Primitive, aus denen das Bild zusammengesetzt ist (bspw. in Grafikprogrammen). Für die unterschiedlichen Aufgaben der digitalen Bildverarbeitung gibt es jeweilige Qualitätskriterien, die eine bestimmte Codierung gegenüber anderen Alternativen vorteilhaft erscheinen lassen. So ist z.B. die Quadtree-Codierung von quadratischen Bildern eine geeignete Darstellung für die Anwendung grundlegender Operationen der *Bildverarbeitung*, wie z.B. das Skalieren. Für die *Bilderkennung* ist die Beschreibung der Bildstruktur mit Hilfe von grammatikalischen Methoden wichtig.

Hier wollen wir die Möglichkeiten der Beschreibung eines Bildes mit Methoden der formalen Sprachen für die *Datenkompression* nutzen. Ausgangspunkt unserer Überlegungen ist dabei eine zweidimensional in einem kartesischen Raster mit äquidistanten Rasterpunkten angeordnete Menge P von $n \times n$ Bildpunkten (Pixel), denen als Intensitätswert ein Element aus der Menge $\{0, 1\}$ zugeordnet wird (Binärbild, oder Bitmap mit 2 Graustufen):

$$P : [0 \ldots n-1] \times [0 \ldots n-1] \to \{0, 1\}.$$

Die semantische Interpretation ist durch eine bijektive Abbildung

$$Sem : \{0, 1\} \to \{schwarz, weiss\}$$

gegeben. Dabei ist es zunächst zweitrangig, ob die Bitmap durch Digitalisierung und Quantisierung des Ausgangssignals eines bildgebenden Sensors oder in einem Grafikprogramm entstanden ist. Die Behandlung von Bildern mit mehreren Graustufen oder von Farbbildern läßt sich durch eine entsprechende Zerlegung in mehrere Binärbilder (bspw. durch Extraktion von Bitebenen) erreichen.

Für die Übertragung oder Speicherung von digitalen Bildern müssen diese als Nachricht interpretiert und durch ein geeignetes Verfahren codiert werden. Als kanonische Form bei der Codierung von Bitmaps bietet sich die serielle Angabe der Intensitätswerte durch Codes einheitlicher Länge an. Für Binärbilder reicht die Verwendung von einem Bit pro Pixel aus. Die zweidimensionale Anordnung der Pixel kann durch ein zeilenweises Abtasten in eine serielle Folge umgewandelt werden. Dieses Verfahren findet in vielen Grafikformaten Anwendung, wobei allerdings weitere Parameter (wie z.B. Kantenlänge, Anzahl der Graustufen, usw.) mit abgelegt werden. Die eindeutige Decodierbarkeit ergibt sich durch die Eindeutigkeit von Position und Codelänge jedes einzelnen Parameters im Datenstrom.

Hauptnachteil dieser einfachen Codierung ist ihr hoher Speicherplatzbedarf. Bei heute üblichen True-Color-Bildern der Auflösung 1024×1024 Pixel ergibt sich auf diese Weise eine Datenmenge von 3 MByte. Speicherplatz und Datenübertragungskapazität sind knappe Ressourcen und so sind seit mehreren Jahrzehnten Verfahren entwickelt worden, die eine Reduzierung der Datenrate erreichen. Dabei unterscheidet man grundsätzlich zwischen verlustlosen und verlustbehafteten Techniken.

Zu den wichtigen Anwendungsgebieten der *verlustbehafteten* Verfahren gehören bspw. die Codierung von Bildsequenzen bei der Bildtelefon-Übertragung oder auch die Erzeugung progressiver Bildqualitätsstufen bei Zugriffen auf Bilddatenbanken. Hier finden Verfahren der sogenannten zweiten Generation Anwendung, die neben den statistischen Eigenschaften der Quelle auch Aspekte des menschlichen visuellen Wahrnehmungsprozesses berücksichtigen. So wird bspw. im JPEG-Verfahren [10] die Tatsache ausgenutzt, daß das menschliche Auge unempfindlicher auf Störungen in den hochfrequenten Bildteilen reagiert als auf Störungen in den niederfrequenten Bereichen. In jüngster Zeit haben zwei verwandte Verfahren die Aufmerksamkeit erregt, die die rekursive Selbstähnlichkeit von bildlichen Darstellungen natürlicher (und damit oft fraktaler) Objekte mit den Möglichkeiten der zweiten Generation verbinden und zur Datenkompression ausnutzen: Die IFS-Codierung von Barnsley [7] und die WFA-Methode von Culik [12]. Wenn die verlustbehafteten Verfahren auch hohe Kompressionsraten erreichen, so geht dies doch oft mit einer starken Verfälschung des eigentlichen Bildinhaltes einher (hier sei nur der Kacheleffekt bei der JPEG-Codierung genannt [13]).

Manche Anwendungen fordern von den Kompressionsalgorithmen die absolute Erhaltung der Bildqualität und zwar nicht nur der subjektiv empfundenen (die auch bei den obengenannten Verfahren bei mittleren Kompressionsfaktoren noch in vertretbaren Größenordnungen liegt), sondern gerade bei Anwendungen mit dem Ziel der automatischen Bildauswertung auch die Erhaltung der objektiven Bildqualität (z.B. die Speicherung und Übertragung von Röntgenbildern in medizinischen Bildverarbeitungs- und Archivierungsanlagen. Hier kann ein einzelner verfälschter Bildpunkt u.U. die automatische oder manuelle Diagnose verändern.) In solchen Anwendungen gilt es also, *verlustlos* zu komprimieren.

In diesem Artikel werden nun die Möglichkeiten untersucht, rekursive Selbstähnlichkeiten der zweidimensionalen Daten zur *verlustlosen* Kompression auszunutzen. Dabei wird von der kanonischen Form der Bildcodierung abgewichen und die Bildinformation in eine endliche Menge von Wörtern über einem endlichen Alphabet abgebildet, wobei die notwendigen Bedingungen für eine eindeutige Decodierbarkeit berücksichtigt werden. Aufgrund der Endlichkeit der Menge der Pixel im Bild P kann durch eine solche Codierung auch nur eine endliche Sprache entstehen. Endliche Sprachen gehören zu der regulären Typ-3-Sprachklasse der Chomsky-Hierarchie, für deren Beschreibung sowohl generative Verfahren (reguläre Grammatiken) als auch analysierende Verfahren (endliche Automaten) existieren. Somit läßt sich zu jedem Bild leicht ein endlicher Automat konstruieren, der genau die sich aus der Codierung ergebende endliche Sprache akzeptiert. Datenkompression kann dann erreicht werden, wenn die zur Abspeicherung dieses (möglichst minimal gewählten) Automaten notwendige Datenmenge kleiner ist, als die zur kanonischen Darstellung der Bitmap.

Der Artikel ist folgendermaßen gegliedert: Kapitel 2 beschreibt den klassischen Ansatz der Verfahren der ersten Generation, die mit informationstheoretischen Methoden optimale Codes erzeugen und zeigt die Schwierigkeiten dieses Ansatzes bei Unkenntnis des stochastischen Bilderzeugungsprozesses. Kapitel 3 erläutert den Ansatz, das Bild mit Hilfe der Quadtree-Zerlegung als endliche Sprache zu betrachten und demonstriert die Verwendung eines endlichen Automaten als zugehörigen Akzeptor dieser Sprache. Das Kapitel 4 beschreibt einige heuristische Ansätze zur effizienten Codierung der Automatentabelle. Die Ergebnisse dieser Ansätze werden für die in Kapitel 5 vorgestellten Beispielbilder mit herkömmlichen Verfahren und theoretischen Zahlen verglichen.

2 Der klassische Ansatz – Optimalcodierung der Shannonschen Quelle mit Präfixcodierern (Huffman)

Die klassische *Informationstheorie* nach Shannon, die streng genommen eigentlich eher eine *Kommunikationstheorie* ist, stellt ein Maß für die Information zur Verfügung, die zwischen zwei Systemen übermittelt wird. Der Sender wird dabei als diskreter stochastischer Zufallsprozeß (Nachrichtenquelle) interpretiert, der über einen endlichen Vorrat an Nachrichten verfügt. Die Nachrichen selber setzen sich aus einer Folge von Zeichen a_i aus einem endlichen Alphabet $A = \{a_1, a_2, \ldots, a_n\}$ zusammen. Diese Zeichen sind die Elementarereignisse des Zufallsprozesses und treten mit bestimmten Wahrscheinlichkeiten $p(a_i)$ auf. Dabei gilt die Vollständigkeit: $\sum_{i=1}^{n} p(a_i) = 1$. Die Information wird in diesem Modell gleichgesetzt mit der Vorhersagbarkeit eines Ereignisses. Aus einer Reihe von Forderungen an die Eigenschaften einer solchen Informationsfunktion I ergibt sich eine logarithmische Funktion [2]: $I_i = -\text{ ld } \ p(a_i)$.

Für die Basis 2 im Logarithmus (ld = logarithmus dualis) wird die Maßeinheit der Information [1 Bit] genannt. Wir können somit den mittleren Informationsgehalt einer Quelle durch Berücksichtigung der Auftrittswahrscheinlichkeiten der einzelnen Zeichen bestimmen:

$$H = -\sum_{i=1}^{n} p(a_i) \cdot \mathrm{ld}\; p(a_i) \text{ [Bit]}.$$

Aufgrund der formalen Ähnlichkeit zu der entsprechenden Formel in der stochastischen Thermodynamik wird diese Größe auch *Entropie* genannt.

Die obengenannten Überlegungen gelten dabei zunächst für Quellen, die kein Gedächtnis besitzen, d.h. bei denen $p(a_i)$ für ein Zeichen unabhängig von allen zuvor ausgesendeten Zeichen a_j ist.

So gesehen besteht ein endliches, binäres Bild aus einer Nachricht mit einer endlichen Anzahl von Zeichen, die von einer stochastischen Nachrichtenquelle (dem Bilderzeugungsprozeß) produziert wurden, wobei die Art und Anzahl der Zeichen weiter unten diskutiert wird.

2.1 Quellencodierung

Zur Speicherung des Bildes in einem digitalen Rechnersystem wird eine (binäre) Codierung der Nachricht notwendig. Die injektive Abbildung der (Quellen-)Codierung

$$C : A \to \{0,1\}^* \backslash \{\epsilon\}$$

ordnet jedem Zeichen des Alphabets einen eindeutigen Code zu. Wir gehen dabei im folgenden stets von einem störungsfreien, idealisierten Übertragungskanal (oder Speichermedium) aus und können daher Aspekte der Kanalcodierung unberücksichtigt lassen. Da wir bei der Speicherung oder Übertragung einer Zeichenfolge auch eine Codefolge erzeugen, muß deren eindeutige Decodierbarkeit garantiert sein. Dies läßt sich prinzipiell auf drei verschiedene Arten erreichen:

- Alle Codes besitzen die gleiche Codelänge.
- Man verwendet spezielle Trennzeichen zwischen den Codewörtern.
- Man verwendet einen Präfixcode, bei dem garantiert ist, daß kein Codewort der Anfang (Präfix) eines anderen Codewortes ist.

Da gezeigt werden kann, daß jeder eindeutig decodierbare Code durch einen Präfixcode mit gleicher Codewortlänge ersetzt werden kann ([9], S. 35), kann man sich auf die Behandlung ebensolcher beschränken, wenn die Codelänge einziges Kriterium ist.

Für die (Bilddaten-)Kompression besteht die Aufgabe nun darin, einen Code zu finden, der eine Codierung der (Bild-)Daten mit möglichst geringer Länge ermöglicht. Es kann gezeigt werden, daß die mittlere Codelänge

$$n(C) = \sum_{i=1}^{n} p(a_i) \cdot |C(a_i)|$$

die Entropie H der Quelle nicht unterschreiten kann. Für die Effizienz eines Codes

$$\mathrm{Eff}(C) = \frac{H}{n(C)}$$

gilt daher stets ([9], S. 131):

$$\mathrm{Eff}(C) \leq 1.$$

Ziel der Datenkompression ist folglich die Minimierung der Redundanz

$$R = n(C) - H.$$

Die Redundanz ist damit der objektiv unwichtige Teil der Codierung, d.h. man kann einen anderen Code wählen, dessen mittlere Codewortlänge kürzer ist, der somit also eine geringere Redundanz besitzt und trotzdem die gleiche Information übermittelt. Die *objektive* Redundanz ist dabei nicht mit der *subjektiven* Irrelevanz einer Information zu verwechseln ([11], S. 83f.).

2.2 Optimale Präfixcodes

Wenn nun für die verlustlose Kompression die untere Grenze einer Codierung durch die Entropie der Quelle gegeben ist, so ist damit noch nichts darüber ausgesagt, wie ein entsprechender optimaler Code zu konstruieren ist. Es wurden allerdings schon sehr früh drei Algorithmen entwickelt, die relativ nahe an der optimalen Codierung liegen (Shannon-, Shannon/Fano- und Huffman-Codierung, [11], S. 112f). Während die beiden erstgenannten gute Laufzeiten aufweisen, kann der Huffman-Algorithmus für viele Wahrscheinlichkeitsverteilungen (WV) bessere Werte liefern. Der Huffman-Algorithmus bildet einen binären Codierungsbaum, indem rekursiv die beiden Zeichen mit geringster Wahrscheinlichkeit zusammengefaßt werden [1]. Auf diese Weise erzeugt der Huffman-Algorithmus einen optimalen Präfixcode, d.h. es gibt keinen anderen Präfixcode mit geringerer Codelänge für die gegebene Quelle, wenn jedem Quellenzeichen genau ein Code zugeordnet wird. In ([11], S. 115–118) wird anschaulich diskutiert, bei welchen Quellen sich die Optimalität gegenüber der einfacheren Shannon- oder Shannon/Fano-Codierung bemerkbar macht. Für eine Quelle mit nur einem Zeichen ist keine Codierung sinnvoll, bei $|A| = 2$ kann jedem Zeichen 0 oder 1 zugeordnet werden, wobei sich immer eine mittlere Codelänge von 1 ergibt. Für $|A| = 3$ und $|A| = 4$ liefern die Algorithmen gleiche Codelängen und erst für Nachrichtenquellen mit mehr als 4 Zeichen zeigt sich die Überlegenheit des Huffman-Codes, wenn auch nur für bestimmte WV. Der Aufbau des Huffman-Codierungsbaumes benötigt dabei für eine Quelle mit $|A| = n$ Zeichen eine Laufzeit aus der Klasse $O(n \cdot \mathrm{ld}\ n)$.

Die Optimalität eines Präfix-Codes darf nicht darüber hinwegtäuschen, daß die Entropie der Quelle nur selten erreicht wird. Der Grund hierfür liegt in der Tatsache, daß jedem einzelnen Zeichen ein binäres Codewort zugeordnet wird. Es lassen sich somit keine rationalen Anteile, sondern nur ganzzahlige Bit-Werte auf die Zeichen verteilen. Dieser Nachteil kann auf zwei verschiedene Weisen aufgehoben werden:

1. Durch die Verwendung eines Verfahrens, das alternativ zu Präfix-Codes auch nichtganzzahlige Bit-Werte zuordnen kann. Hier ist die arithmetische Codierung zu nennen, die ein Zahlenintervall der reellen Achse in Abhängigkeit der WV der Quelle in immer genauere Teile zerlegt und die Trenngrenze als Code verwendet. Nachteil dieses Verfahrens ist der erhöhte Implementierungsaufwand. Testergebnisse haben gezeigt, daß die arithmetische Codierung in den meisten Fällen die Huffman-Codierung um weniger als 10% übertrifft ([6], S. 538).

2. Durch die Zusammenfassung von jeweils k Zeichen der Quelle zu n^k Pseudozeichen (Blöcken). Die Blockcodierung nähert sich für $k \to \infty$ der Entropie der Quelle immer näher an.

Der erste Ansatz wird in diesem Artikel nicht weiter verfolgt und kann durch den 10%-Effekt implizit berücksichtigt werden. Der zweite Ansatz trifft in der praktischen Realisierung auf zwei kritische Probleme:

- Für die Decodierung der Nachricht muß der Codebaum im Decodierer bekannt sein. Will man nicht immer die gleiche WV verwenden (was wenig sinnvoll wäre), ist man gezwungen, den Codebaum zusätzlich zur eigentlichen Information mit zu übertragen. Dadurch wird sowohl bei kurzen Nachrichten als auch bei langer Blockbildung (wegen des exponentiellen Wachstums des Codebaums) die Kompressionsrate gesenkt. Einen Kompromiß kann man mit den in diesem Artikel ebenfalls unberücksichtigten adaptiven Verfahren erreichen, bei denen der Codebaum im Laufe der Codierung sowohl im Codierer als auch im Decodierer ständig dem aktuellen Verlauf der Nachrichtenstatistik angepaßt wird.
- Die Blöcke können nicht beliebig lang werden, da sie stets klei-

ner als die endliche Nachricht (das Bild) sein müssen. Außerdem ist die WV der Quelle i.a. gar nicht bekannt und kann nur durch die relative Häufigkeit der Zeichen bestimmt werden. Je länger die Blockbildung ist, umso ungenauer wird dadurch aber auch die Abschätzung der Wahrscheinlichkeiten. Im Extremfall besteht die gesamte Nachricht aus einem einzigen Block, dessen rel. Häufigkeit = 1 ist, womit eine sinnvolle Codierung nicht mehr möglich ist.

2.3 Der stochastische Bilderzeugungsprozeß

Kehren wir zurück zu den Binärbildern, so stellt sich die Frage, wieviele Elemente enthält das Alphabet A der Nachrichtenquelle und aus wie vielen Zeichen setzt sich das Bild zusammen? Betrachten wir zunächst zwei Extreme:

1. $A = \{0, 1\}$. Dabei bedeutet das Zeichen $a_i = 0$, daß der Bildpunkt schwarz ist und das Zeichen $a_i = 1$, daß der Bildpunkt weiß ist. Für jeden einzelnen Bildpunkt wird folglich ein Zeichen erzeugt und das Bild enthält somit n^2 einzelne Zeichen, die alle codiert werden müssen. Der optimale Huffman-Code ergibt sich dabei direkt aus der Tatsache, daß es nur zwei unterschiedliche Zeichen gibt und die kleinste Informationseinheit das Bit ist: $C(0) = 0, C(1) = 1$. D.h. jedes Pixel wird mit genau einem Bit codiert und wir erhalten eine Codelänge von n^2 Bit. Diese kanonische Form wird auch als *Bitmap* bezeichnet und gilt im folgenden als Bezugsgröße für die Bestimmung des erreichten Kompressionsfaktors. Dabei ist die gewählte Codierung unabhängig von der gegebenen Wahrscheinlichkeitsverteilung optimal (siehe obige Ausführungen zur Huffman-Codierung).

2. $A = \{0, 1, \ldots, 2^{n^2} - 1\}$. D.h. wir fassen das Bild als ein einziges Zeichen auf, indem wir die Intensitätswerte der Pixel als Ziffer einer Dualzahl mit n^2 Stellen interpretieren. Dadurch gibt es insgesamt 2^{n^2} verschiedene Bilder (und somit Zahlen) deren Zahlenwert wir als Nachricht interpretieren. Setzen wir voraus, daß alle möglichen Bilder gleichwahrscheinlich vorkommen und eine andere Aussage läßt sich bei der Codierung eines einzigen Bildes über den Bilderzeugungs-

prozeß schließlich nicht machen, so ergibt sich für die Entropie der Quelle:

$$H = -\sum_{i=1}^{2^{n^2}} p(a_i) \cdot \mathrm{ld}\ p(a_i)\ [\mathrm{Bit}].$$

Mit $p(a_i) = \frac{1}{2^{n^2}}$ ergibt sich damit $H = -\ \mathrm{ld}\ \frac{1}{2^{n^2}}[\mathrm{Bit}] = n^2[\mathrm{Bit}]$. D.h. auch in diesem Fall kann die Datenmenge nicht unter die Bitmap-Größe gesenkt werden. Man kann dieses Vorgehen auch als die Blockbildung mit $k = n^2$ interpretieren.

Während also im 2. Fall aufgrund der Tatsache, daß die Stichprobe nur ein einziges Zeichen enthält, mit Hilfe der relativen Häufigkeiten keine Rückschlüsse auf die Wahrscheinlichkeitsverteilung der Quelle gezogen werden können, hat im 1. Fall die Abschätzung der WV keinen Einfluß auf die erzeugte Codelänge. Das Optimum liegt nun irgendwo zwischen diesen Extremen und hängt wesentlich von der Größe des erzeugten Codebaums ab.

Tabelle 1 zeigt die Entropie der Quelle bei Blockbildung für die Beispielbilder. Die Zeichen $a_1, \ldots, a_n$ der Quelle werden zu neuen Pseudozeichen (z.B. $a'_1 = a_1a_1, a'_2 = a_1a_2, a'_3 = a_2a_1, a'_4 = a_2a_2$ für eine Blocklänge von $k = 2$ und $n = 2$ Zeichen). Für diese Menge $A' = \{a'_1, \ldots, a'_m\}$ berechnet sich die Block-Entropie durch

$$H' = -\sum_{i=1}^{m} p(a'_i) \cdot \mathrm{ld}\ p(a'_i)\ Bit.$$

Dabei ist zu beachten, daß es aufgrund der Pseudokonstruktion dazu komm en kann, daß $m \neq n^k$, da manche Kombinationen in der Bildmenge eventuell überhaupt nicht vorkommen, d.h. $p(a'_i) = 0$ ist. Diese Zeichen haben daher keinen Code und folglich gilt die sonst gültige Ungleichung $H' \leq k \cdot H$ hier nicht. H' berechnet außerdem die Entropie bezogen auf die Pseudozeichen. Da diese k Zeichen enthalten, muß die Pseudoentropie durch die Blocklänge dividiert werden, um zur eigentlichen Entropie bzgl. der Zeichen zu kommen: $H = \frac{H'}{k}$.

Bild	Blockgröße	Anzahl Zeichen	Wahrscheinlichkeitsverteilung		Entropie H [Bit]
			p(0)	p(1)	
Cranach	1	2	0.18	0.82	0.67
	2	4	—	—	0.51
	4	16	—	—	0.34
	8	256	—	—	0.38
Duden	1	2	0.22	0.78	0.75
	2	4	—	—	0.61
	4	16	—	—	0.42
	8	256	—	—	0.46
Röntgen	1	2	0.97	0.03	0.19
	2	4	—	—	0.15
	4	16	—	—	0.10
	8	256	—	—	0.11
Schlüssel	1	2	0.09	0.91	0.44
	2	4	—	—	0.30
	4	16	—	—	0.18
	8	256	—	—	0.16

Tabelle 1: Entropie der Beispielbilder

Die letzte Spalte in Tabelle 1 zeigt die auf diese Weise korrigierte Blockentropie $H = \frac{H'}{k}$. Man erkennt deutlich die fallenden Entropiewerte bei steigender Blocklänge. Dabei muß aber beachtet werden, daß zusätzlich noch der Codierungsbaum mit abgelegt werden muß. Bei $k = 8$ beginnt sich der oben erwähnte exponentielle Größeneffekt des Codierungsbaumes bemerkbar zu machen, denn er beinhaltet hier schon n^k Codes und für unsere Anwendung mit $n = 2$ bedeutet dies, 256 Codewörter als Huffman-Codebaum mit übertragen zu müssen. Bei $k = 16$ macht die Übertragung mit den schon 65.536 Codewörtern keinen Sinn mehr, auch wenn man nur die tatsächlich vorkommenden Codewörter berücksichtigen würde (dies wären bei 512×512 Pixeln Bildgröße maximal $\frac{262.144}{16} = 16.384$ verschiedene Codes). Wir wollen daher in Kap. 4 die Block-Entropiewerte für $k = 8$ mit den Ergebnissen des unten vorgestellten Verfahrens vergleichen.

3 Interpretation des Bildes als endliche Sprache

Alternativ zur Interpretation des Bildes als serielle Folge von Intensitätswerten (u.U. mit Blockbildung) wollen wir nun die Beschreibung des Bildes P mit Hilfe einer endlichen Sprache L über einem festgelegten Alphabet Σ vornehmen:

$$L \subset \Sigma^*, L \text{ endlich.}$$

Endliche Sprachen lassen sich durch reguläre Ausdrücke beschreiben und sind somit Teilklasse der Chomsky-Typ3-Sprachklasse, deren Beschreibung auch durch die Definition eines entsprechenden endlichen Automaten (FA) erfolgen kann.

Zur Erzeugung einer solchen Bildsprache L muß nun eine bijektive Abbildung ϕ:

$$\phi : [0 \ldots n-1] \times [0 \ldots n-1] \to L$$

gefunden werden, die jedem Pixel des Bildes ein eindeutiges Codewort $\omega \in L$ zuordnet. Zur Vereinfachung des Zerlegungsprozesses wollen wir im folgenden von der Quadtree-Zerlegung ausgehen, die mit Hilfe des Alphabets $\Sigma = \{0, 1, 2, 3\}$ für jedes Pixel ein Codewort $\omega \in \Sigma^k$ der Länge $|\omega| = k$ erzeugt. Der Parameter k ergibt sich bei dieser Methode direkt aus der Größe des (quadratischen) Bildes: $|\Sigma|^k = n^2$ (Bei $n = 512 \Rightarrow k = 9$).

Diese Zuordnung wird erreicht, indem das Bild rekursiv in vier gleichgroße Subquadranten mit einer Zuordnung nach Abb. 1 zerlegt wird:

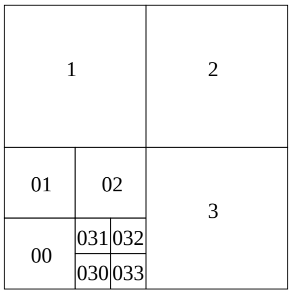


Abbildung 1: Einige Quadtree-Zuordnungen bis $k = 3$

Die Codeworte entstehen durch Konkatenation dieser Präfixe. Für $n = 512$ besitzt ein Teilquadrat bei $k = 9$ die Größe eines Pixels und der Zerlegungsprozeß endet bei dieser Auflösung.

Die Sprache L_P enthält nun alle Codewörter $\omega \in L$, deren Pixel in der Bitmap den Wert 1 besitzen:

$$L_P := \{\omega \in L | P(\phi^{-1}(\omega)) = 1\}$$

Es kann gezeigt werden, daß die Wahl der Werte mit $P(\phi^{-1}(\omega)) = 0$ einen identischen Kompressionsfaktor ergibt [14]. Ergebnis dieses Codierungsprozesses ist somit eine endliche Menge $L_P \subseteq \Sigma^k$ mit

$|L_P| \leq n^2$. Diese Menge charakterisiert in eindeutiger Weise das Bild P.

3.1 Codierung mit endlichen Automaten

Endliche Sprachen lassen sich, als Teilklasse der regulären Sprachen, mit Hilfe von endlichen Automaten beschreiben. Wir wollen im folgenden nur deterministische und vollständige endliche Automaten $A = (Q, \Sigma, \delta, q_0, F)$ verwenden, mit Q: endliche Menge von Zuständen, Σ: Eingabealphabet, $\delta : Q \times \Sigma \rightarrow Q$: (totale) Übergangsfunktion, $q_0 \in Q$: ausgezeichneter Startzustand und $F \subseteq Q$: Menge der akzeptierenden Endzustände.

Die Überführungsfunktion ist eine totale Abbildung, d.h. $\forall q \in Q, a \in \Sigma \ \ \exists q^{'} \in Q : (q, a, q^{'}) \in \delta$. Sie wird rekursiv erweitert auf die Mehrschrittableitung $\delta^* : Q \times \Sigma^* \rightarrow Q$, wobei $\delta^*(q, \epsilon) := q$ und $\delta^*(q, a\omega) := \delta^*(\delta(q, a), \omega)$, mit $q \in Q, \omega \in \Sigma^*, a \in \Sigma$. Sie beschreibt den Wechsel der Zustände beim Lesen der Eingabe $\omega \in \Sigma^*$. Die von A akzeptierte (Bild-)Sprache ist dann:

$$L(A) := \{\omega \in \Sigma^* | \delta^*(q_0, \omega) \in F\}.$$

Für eine gegebene endliche (Bild-)Sprache $L_P \subseteq \Sigma^*$ läßt sich ein DFA mit einem schlingenfreien Übergangsgraph und einem akzeptierenden (F) und einem nichtakzeptierenden ($\overline{F}$) Endzustand leicht konstruieren, für den gilt: $L(A) = L_P$. Das Bild P läßt sich aus diesem Automaten rekonstruieren, indem der Automat für alle $\omega \in \Sigma^k \cap L_P$, die einen schwarzen Bildpunkt repräsentieren in den Zustand F und für alle anderen $\omega \in L \setminus L_P$ in den Zustand $\overline{F}$ übergeht. Wir haben auf diese Weise eine Transformation der Bildinformation von der Intensitätsdarstellung in die Automaten-Darstellung gewonnen.

Ein DFA kann durch die Zusammenfassung von äquivalenten Zuständen reduziert werden. Zwei Zustände q_1, q_2 sind äquivalent ($q_1 \equiv q_2$), wenn $\forall \omega \in \Sigma^*$ gilt: $\delta^*(q_1, \omega) \in F \Longleftrightarrow \delta^*(q_2, \omega) \in F$. Ein Automat ist dann reduziert, wenn $\forall q_1, q_2 \in Q : q_1 \equiv q_2 \Longleftrightarrow q_1 = q_2$ und alle

Zustände vom Startzustand q_0 aus erreichbar sind: $\forall q \in Q\ \exists \omega \in \Sigma^*$ mit $\delta^*(q_0, \omega) = q$.

Betrachtet man die Teile des Bildes, die durch einen Zustand repräsentiert werden, so läßt sich die Äquivalenz zweier Zustände als die Ähnlichkeit zweier Bildteile deuten. Jeder Zustand steht im unreduzierten Automaten für einen bestimmten Subquadranten des Bildes. Der akzeptierende Endzustand F steht für ein schwarzes Quadrat (beliebiger Größe, allerdings mit einer Kantenlänge, die einer 2-er-Potenz entspricht) und der nichtakzeptierende Endzustand $\overline{F}$ steht für ein weißes Quadrat. Interpretiert man den Übergangsgraph als Baum der Ordnung $|\Sigma|$, so sind jeweils die Zustände äquivalent, deren Teilbäume äquivalent sind. Dies wird in Abbildung 2 verdeutlicht.

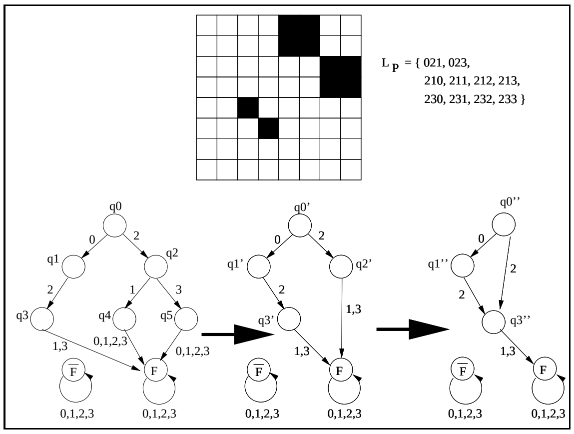


Abbildung 2: Beispielbild mit bis auf Skalierung identischen Bildteilen.

Das Bild P wird durch die Sprache L_P repräsentiert, für die der entsprechende DFA skizziert ist (dabei sind die Übergänge nach $\overline{F}$ zur besseren Lesbarkeit weggelassen.) Man erkennt die Äquivalenz der Zustände q_4 und q_5, die ein identisches Übergangsverhalten in die nachfolgenden Endzustände besitzen. Sie repräsentieren die beiden großen schwarzen Quadrate im rechten oberen Teilquadranten des Bildes und sind somit äquivalent zum Endzustand F. Nach dem ersten Reduktionsschritt erkennt man die Äquivalenz der Zustände $q_2^{'}$ und $q_3^{'}$, die ein identisches Übergangsverhalten in die nachfolgenden Endzustände besitzen. Beide akzeptieren die Worte, die durch den regulären Ausdruck $(1|3)(0|1|2|3)^*$ gebildet werden und können somit zu einem neuen Zustand $q_3^{''}$ zusammengefaßt werden. Die rechte Seite von Abbildung 2 zeigt den auf diese Weise reduzierten Automaten, der die gleiche Sprache L_P akzeptiert, wenn vorausgesetzt wird, daß $L_P \subset \Sigma^k$, d.h. die Wortlänge auf $|\omega| \leq k$ beschränkt ist. Während der Zustand $q_3^{'}$ die kleine Diagonale im linken unteren Teilquadranten des Bildes repräsentierte, wurde mit $q_2^{'}$ die große Diagonale im rechten, oberen Teilquadranten dargestellt. Die Ähnlichkeit dieser beiden Bildteile, die sich nur durch einen Skalierungsfaktor voneinander unterscheiden, wird für die Bildung des Zustands $q_3^{''}$ ausgenutzt, der das Prinzip 'Diagonale' repräsentiert. Die Art der Übergänge, die in diesen neuen Zustand $q_3^{''}$ hineinführen, bestimmen durch ihre Zeichenfolge ('02' oder '2') und die Länge dieser Folge die Position und die Größe des Musters. Auf diese Weise wirken sich Ähnlichkeiten im Bild günstig auf die Anzahl der Zustände des Automaten aus. Im folgenden wollen wir stets davon ausgehen, daß ein vollständig reduzierter Automat erzeugt wurde.

4 Informationsgehalt von Zustandsübergangstabellen

Nachdem im ersten Schritt des Codierungsmechanismus ein DFA mit einer minimalen Zahl von Zuständen erzeugt wurde, stellt sich die Frage der effizienten Codierung des Automaten. Den größten informationstragenden Teil des Automaten stellt die Überführungsrelation dar. Die linke Spalte von Tabelle 2 gibt einen Einblick in die Größenordnungen der Automaten, die aus den Beispielbildern (siehe Kap. 5) erstellt wurden.

Da wir für die Sprache ein vierwertiges Alphabet Σ ($|\Sigma| = 4$) verwen-

Bild	Anzahl Zustände $\lvert Q\rvert$	Information I_A [Bit]	c.f.
Cranach	4.113	128.520	0.49
Duden	5.108	165.268	0.63
Röntgen	1.656	41.437	0.16
Schlüssel	1.796	47.497	0.18

Tabelle 2: Automatengröße der Beispielbilder

det haben, gibt es für jeden Automatenzustand vier Übergänge zu Folgezuständen. Die Übergänge des akzeptierenden (F) und nichtakzeptierenden ($\overline{F}$) Endzustands müssen nicht explizit codiert werden und so handelt es sich um eine Tabelle mit $(|Q|-2)\cdot|\Sigma|$ Einträgen.

4.1 Automatentabelle als Shannonsche Nachrichtenquelle

Die naheliegendste Möglichkeit der effizienten Codierung der Tabelle besteht in der Interpretation der Tabelle als Nachricht einer stochastischen Quelle mit den Tabelleneinträgen als Zeichen. Das Alphabet der Quelle enthält dabei genau die möglichen $|A| = |Q|$ Folgezustände. Für eine Optimal-Codierung dieser Quelle können wir die Häufigkeiten $h(a_i)$ der einzelnen Alphabetzeichen der Tabelle als Abschätzung für die WV der Quelle verwenden und mit ihrer Hilfe die Entropie der Tabelle bestimmen:

$$H_A = \sum_{i=1}^{|Q|} \frac{h(a_i)}{|\Sigma|\cdot(|Q|-2)} \cdot \mathrm{ld}\,\left(\frac{h(a_i)}{|\Sigma|\cdot(|Q|-2)}\right)\ [\mathrm{Bit}].$$

Der Informationsgehalt der Tabelle ergibt sich somit durch

$$I_A = |\Sigma|\cdot(|Q|-2)\cdot H_A.$$

Die rechten Spalten von Tabelle 2 zeigen die auf diese Weise gebildeten Werte und den Kompressionsfaktor (c.f.), der sich durch Division mit der Bitmap-Größe ergibt. Im Vergleich mit Tabelle 1 zeigt sich, daß die Werte größer sind als die Entropie-Werte der Bitmap bei Blockbildung mit $k = 8$ (der Vergleich ist deshalb möglich, weil die Entropiewerte in Tabelle 1 den Kompressionsfaktoren entsprechen). Im folgenden soll nun gezeigt werden, wie mit Hilfe von einigen heuristischen Ansätzen die offensichtlich in der Tabelle enthaltene Redundanz reduziert werden kann.

4.2 Heuristische Optimierungs-Ansätze

Im folgenden werden zwei heuristische Ansätze vorgestellt, mit denen eine kompakte Speicherung der Automatentabelle erreicht werden kann. Die Ergebnisse werden an den Beispielbildern quantitativ abgeschätzt und mit bestehenden Kompressionsverfahren verglichen.

Wir können die Zustände des Automaten in Äquivalenzklassen nach folgender Relation aufteilen:

$$[Q]_i := \{q \in Q \setminus \{F, \overline{F}\} | \exists \omega \in \Sigma^* : \delta^*(q_0, \omega) = q \wedge |\omega| = i$$
$$\wedge |\omega| \geq |\omega^{'}| \forall \omega^{'} \in \Sigma^* : \delta^*(q_0, \omega^{'}) = q\}$$

D.h. wir bilden Schichten im Übergangsgraphen und alle Zustände mit gleicher maximaler Pfadlänge vom Startzustand gehören zur selben Schicht. Für den linken Automaten in Abb. 2 gilt dann z.B. : $[Q]_0 = \{q_0\}$, $[Q]_1 = \{q_1, q_2\}$, $[Q]_2 = \{q_3, q_4, q_5\}$ und $[Q]_3 = \{F, \overline{F}\}$.

Offensichtlich können die Übergänge von Zuständen in $[Q]_i$ nur auf Zustände aus den Schichten $[Q]_j$ mit $j > i$ oder auf $\{F, \overline{F}\}$ zeigen. D.h. die Anzahl der möglichen Folgezustände ist nicht für alle Tabelleneinträge gleich groß. Diese Tatsache wird zu einer effizienten Codierung ausgenutzt. Dazu zeigt Tabelle 3 für die Beispielbilder die Anzahl der Zustände der einzelnen Schichten $[Q]_i$ mit $i = 1 \ldots 9$. Der 'Max'-Wert ergibt sich aus einer Abschätzung in [14] für die maximale Anzahl von Zuständen im Automaten im worst-case-Fall:

$$|Q|_{max,k} = \sum_{j=0}^{k-l-1} |\Sigma|^j + \sum_{j=0}^{|\Sigma|^l} \binom{|\Sigma|^l}{j},$$

wobei l die größte ganzzahlige Lösung der folgenden Ungleichung darstellt:

$$|\Sigma|^{k-l} \geq 2^{|\Sigma|^l} - 2^{|\Sigma|^{l-1}}$$

(Im Fall k=9 also l=1).

Die jeweils zweite Zeile 'Σ_{Follow}' bildet die Summe aller für einen Zustand $q \in [Q]_i$ in Frage kommenden Zustände $q' \in \{[Q]_{i+1} \ldots [Q]_k\}$.

Für ein *Verfahren A* bietet sich die Codierung der Folgezustände in kanonischer Weise mit jeweils $\lceil \log_2 \Sigma_{Follow} \rceil$ Bit an. So lassen sich z.B. die Zustände der Menge $[Q]_8 \cup [Q]_9$ mit jeweils 4 Bit codieren. Die Trennindizes der einzelnen Schichten müssen bei einer solchen Vorgehensweise als Zusatzinformation berücksichtigt werden. So ergibt sich ein zusätzlicher Aufwand von k Grenzwerten mit Werten $\leq |Q|_{max,9} = 21.861$, also $k \cdot \lceil \log_2 21.861 \rceil$ Bit $= k \cdot 15$ Bit. Tabelle 4 zeigt die mit diesem Vorgehen erzielten Werte.

Die Ergebnisse erreichen zwar nur knapp die Entropiewerte der Huffman-Codierung laut Tabelle 2, aber dabei muß man berücksichtigen, daß hier keine zusätzliche Information abgelegt werden muß (Im worst-case-Fall wird schließlic h die Übertragung einer Huffman-Tabelle für 21.861 Codezeichen notwendig).

Das *Verfahren B* erreicht wesentlich bessere Ergebnisse und beruht auf den folgenden Beobachtungen:

- Der akzeptierende und der nichtakzeptierende Endzustand treten häufig als Folgezustand auf.
- In den ersten Schichten des Automaten ist dieser im allgemeinen relativ vollständig, d.h. Zustände konnten nicht zusammengefaßt werden und so bilden sich in der Tabelle lange Folgen

i	0	1	2	3	4
Max	1 (q_0)	4	16	64	256
Cranach	1	4	15	56	208
Σ_{Follow}	4.113	4.112	4.108	4.093	4.037
$\lceil \log_2 \Sigma_{Follow} \rceil$	13	13	13	12	12
Duden	1	4	16	64	240
Σ_{Follow}	5.108	5.107	5.103	5.087	5.023
$\lceil \log_2 \Sigma_{Follow} \rceil$	13	13	13	13	13
Röntgen	1	4	15	41	95
Σ_{Follow}	1.656	1.655	1.651	1.636	1.595
$\lceil \log_2 \Sigma_{Follow} \rceil$	11	11	11	11	11
Schlüssel	1	3	8	20	54
Σ_{Follow}	1.796	1.795	1.792	1.784	1.764
$\lceil \log_2 \Sigma_{Follow} \rceil$	11	11	11	11	11
i	5	6	7	8	9
Max	1.024	4.096	16.384	14	2 $(F, \overline{F})$
Cranach	642	1.795	1.376	14	2
Σ_{Follow}	3.829	3.187	1.392	16	2
$\lceil \log_2 \Sigma_{Follow} \rceil$	12	12	11	4	1
Duden	894	2.442	1.431	14	2
Σ_{Follow}	4.783	3.889	1.447	16	2
$\lceil \log_2 \Sigma_{Follow} \rceil$	13	12	11	4	1
Röntgen	215	497	772	14	2
Σ_{Follow}	1.500	1.285	788	16	2
$\lceil \log_2 \Sigma_{Follow} \rceil$	11	11	10	4	1
Schlüssel	163	531	1.000	14	2
Σ_{Follow}	1.710	1.547	1.016	16	2
$\lceil \log_2 \Sigma_{Follow} \rceil$	11	11	10	4	1

Tabelle 3: Zugehörigkeit der Zustände zu den Schichten $[Q]_i$

Bild	Codieraufwand $[Bit]$	c.f.
Cranach	145.520	0.56
Duden	190.212	0.73
Röntgen	48.612	0.19
Schlüssel	48.252	0.18

Tabelle 4: Ergebnisse nach Verfahren A

Bild	Größe der Automatentafel	Anzahl			
	$(\lvert Q\rvert - 2) \cdot \lvert\Sigma\rvert$	F	$\overline{F}$	Diff '1'	Rest
Cranach	16.444	1.418	3.473	5.202	6.351
Duden	20.424	1.259	4.404	5.267	9.494
Röntgen	6.616	2.066	440	2.334	1.776
Schlüssel	7.176	1.490	778	2.825	2.083

Tabelle 5: Häufigkeiten der Kategorien in den Automatentabellen

von Folgezuständen, deren Numerierung sich aufgrund der Bildungsreihenfolge im Algorithmus nur durch eine Eins voneinander unterscheidet.

Tabelle 5 zeigt einige ausgewählte Daten der Automatentabellen für die Beispielbilder. Der akzeptierende Endzustand F und der nichtakzeptierende Endzustand $\overline{F}$ machen zusammen mindestens 25% der Tabelleneinträge aus. Etwa in derselben Größenordnung liegt auch die Anzahl der Zustände, die nach Eliminierung aller anderen Tabellenelemente eine monoton steigende Folge von Zahlen bilden, die jeweils eine Differenz von '1' zueinander aufweisen. Eine Verbesserung der Ergebnisse kann also erreicht werden, wenn man einen zweistufigen Codierungsmechanismus verwendet, bei dem zuerst entschieden wird, zu welcher der vier in der Tabelle aufgeführten Kategorien der jeweilige Eintrag gehört und dann im zweiten Schritt eine geeignete

Bild	feste Codelänge				Schritt 1
	F	$\overline{F}$	Diff '1'	Rest	[Bit]
Cranach	00	01	10	11	32.888
Duden	00	01	10	11	40.848
Röntgen	00	01	10	11	13.232
Schlüssel	00	01	10	11	14.352
Bild	Präfixcode				Schritt 1
	F	$\overline{F}$	Diff '1'	Rest	[Bit]
Cranach	111	110	10	0	31.428
Duden	111	110	10	0	37.017
Röntgen	10	110	0	111	13.114
Schlüssel	110	111	0	10	13.795

Tabelle 6: Schritt 1 von Verfahren B

Codierung für die 'Rest'-Elemente findet.

Tabelle 6 zeigt in Spalte 1 die Ergebnisse für Schritt 1 bei Verwendung einer festen Codelänge von 2 Bit pro Tabelleneintrag und folgender Zuordnung:

$$'00' \rightarrow F,\ \ '01' \rightarrow \overline{F},\ \ '10' \rightarrow \text{Diff '1'},\ \ '11' \rightarrow \text{'Rest'}.$$

Spalte 2 zeigt die Werte bei Verwendung eines Präfixcodes mit Huffman-Zuordnung entsprechend der Häufigkeit der einzelnen Kategorien.

Die Rekonstruktion der Automaten-Tabelle ist somit für die Werte der Kategorien F, $\overline{F}$ und Diff '1' eindeutig möglich, wenn der Startwert der monoton steigenden Folge mit abgelegt wird. Für die Werte der 'Rest'-Menge können wir eine kanonische Codierung vornehmen, die mit $\lceil \log_2(gr\ddot{o}\beta terWert - kleinsterWert) \rceil$ Bit pro Eintrag auskommt. Tabelle 7 zeigt diese Werte für die Beispielbilder und die Summe der Werte aus Schritt 1 und 2 mit den erreichten Kompressionsfaktoren.

Im Vergleich mit den Ergebnissen der klassischen Huffman-Codierung mit Blocklänge k=8 zeigt sich die Überlegenheit des Verfahrens im

Bild	Min	Max	$\lceil \log_2(Max - Min) \rceil$	Schritt 2 [Bit]
Cranach	437	4.110	12	76.212
Duden	1.226	5.105	12	113.928
Röntgen	159	1.653	11	19.536
Schlüssel	109	1.793	11	22.913

Bild	Σ Verfahren B [Bit]	c.f.
Cranach	107.640	0.41
Duden	150.945	0.58
Röntgen	32.650	0.12
Schlüssel	36.708	0.14

Tabelle 7: Schritt 2 und Gesamtergebnis von Verfahren B

Bild *Schlüssel*. Bei genauer Betrachtung wird deutlich, daß in diesem Bild der linke obere Quadrant vollständig weiß ist und somit mit einem einzigen Zustand im Automaten repräsentiert werden kann. Bei den anderen Bildern zeigen sich geringfügig schlechtere Werte. Dabei ist aber wiederum zu beachten, daß bei der Huffman-Codierung zusätzlich der Codebaum zu berücksichtigen ist, während beim hier vorgestellten Verfahren lediglich Angaben über Anzahl der Zustände, Wert von $|\Sigma|$ usw. abgelegt werden müssen.

Um den Vergleich mit bestehenden Kompressionsverfahren zu verdeutlichen, zeigt Tabelle 8 abschliessend die Werte für die Programme:

- COMPRESS (Version 8.2) Adaptive Lempel-Ziv-Codierung [5].
- PACK Huffman-Codierung.
- COMPACT Adaptive Huffman-Codierung.

Alle Tools wurden unter DEC ULTRIX 4.4 verwendet. Die Bitmap wurde dabei als 32 KByte-Datei gelesen, in der jeweils 8 Pixel zu einem Byte zusammengefaßt wurden. Das Programm COMPRESS

Bild	COMPRESS [Bit]	c.f.	PACK [Bit]	c.f.	COMPACT [Bit]	c.f.
Cranach	107.960	0.41	102.944	0.39	102.672	0.39
Duden	125.768	0.48	123,264	0.47	123.192	0.47
Röntgen	31.048	0.12	49.712	0.19	48.848	0.19
Schlüssel	36.616	0.14	57.856	0.22	56.944	0.22

Tabelle 8: Vergleich mit bestehenden Kompressionsverfahren.

verwendet einen Tabellen-basierten Ansatz und wurde deshalb mit in den Vergleich integriert, um neuere Verfahren für sequentielle Daten zu berücksichtigen, die höhere Kompressionsfaktoren erreichen können, als die traditionellen Huffman-Codierer.

Auch hier wird deutlich, daß sich die verlustlose Kompression mit Automaten durchaus mit herkömmlichen Methoden vergleichen läßt. Bei den Bildern *Cranach*, *Röntgen* und *Schlüssel* werden ähnliche Werte erreicht bzw. übertroffen. Nur im Bild *Duden* wird ein wesentlich schlechteres Ergebnis erzielt.

5 Verwendetes Bildmaterial

Die vorgestellten Ergebnisse wurden aus den folgenden vier Bildbeispielen berechnet, die so ausgewählt wurden, daß eine möglichst große Bandbreite von Nutzanwendungen abgedeckt wird. Alle Abbildungen sind mit Hilfe eines Scanners von natürlichen Vorgaben abgetastet und anschließend binarisiert und auf eine einheitliche Größe von 512 mal 512 Pixel gebracht worden.

- *Cranach* ist eine Karikatur auf das Papsttum von Lukas Cranach dem Älteren um 1545. Es handelt sich dabei um eine Nachzeichnung nach einem Holzschnitt [4].
- *Duden* ist ein Ausschnitt aus dem etymologischen Wörterbuch der deutschen Sprache [8] und steht als Beispiel für ein Faksi-

mile.

- *Röntgen* ist aus einem gescannten Röntgenbild des Autors entstanden und steht für medizinische Bilddatenverarbeitung.

- *Schlüssel* ist der gescannte Zimmerschlüssel des Autors, wie er als Vorlage in einer Bilderkennung zu finden wäre.

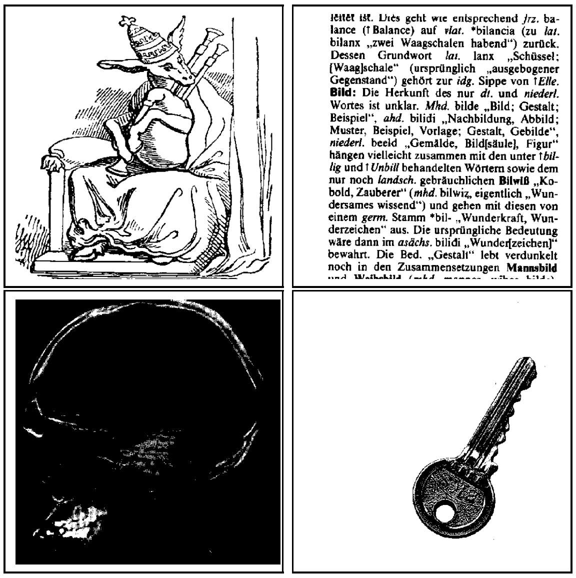


Abbildung 3: Beispielbilder *Cranach, Duden, Röntgen, Schlüssel*

6 Zusammenfassung und Ausblick

Wir haben in diesem Artikel die Codierung von digitalen Bildsignalen mit Hilfe von endlichen deterministischen Automaten beschrieben. Dazu werden den relevanten Pixeln der Bildstruktur Wörter einer endlichen Sprache zugeordnet, welche dann durch einen DFA repräsentiert wird. Es wurden weiterhin zwei heuristische Ansätze zu einer effektiven Codierung der Automatenübergangstabelle des reduzierten Automaten angegeben. Die erreichten Kompressionsfaktoren wurden mit den Ergebnissen klassischer Entropiecodierungsverfahren verglichen, die aufgrund von Kenntnissen über den stochastischen Bilderzeugungsprozeß einen 'optimalen' Präfixcode erzeugen. Dabei wurde der Nachteil dieser Verfahren deutlich, die aufgrund der Nichtstationarität des zweidimensionalen Bildsignals nur suboptimale Lösungen erzeugen können. Auch die Verwendung eines adaptiven Huffman-Verfahren (COMPACT, siehe Tabelle 8) konnte diesen Nachteil nur bedingt korrigieren. Der hier verwendete Ansatz findet sich in ähnlicher Form auch in der WFA-Methode von Culik [12], wobei der Hauptunterschied in der Intention und in der Interpretation der Automatenzustände besteht. Bei Culik steht jeder einzelne Zustand für ein vollständiges Bild und das Originalbild wird durch eine gewichtete Linearkombination aus diesen Zustandsbildern erzeugt. Das Verfahren ist auf die verlustbehaftete Codierung von Grauwertbildern ausgerichtet und verwendet dazu Abstandskriterien, die an die subjektiven Vergleichskriterien des Menschen angepaßt sind.

Der hier vorgestellte Ansatz dient der Ausnutzung zweidimensionaler Redundanz zur verlustlosen Kompression. Hier entspricht jeder Automatenzustand einem bestimmten Teilmuster des Bildes. Dabei entfällt eine explizite Bestimmung der Faktoren der Linearkombination und das Originalbild ergibt sich durch Vereinigung der Teilmuster.

Die Quadtree-Zerlegung ist der erste kanonische Ansatz zur Erzeugung der endlichen Bildsprache. Hier lassen sich allerdings noch eine Reihe bildabhängiger Alternativen denken. Z.B. die Variation von $|\Sigma|$ auf jeder Schicht $[Q]_i$ oder eine feinere und abweichende Struktur bei gleichbleibendem $|\Sigma|$. Weiterführende Arbeiten könnten die Zerlegung von transformationscodierten Bilddaten untersuchen, wobei durch die Dekorrelation der Daten eine Steigerung des Kompressionsfaktors zu erwarten ist. Eine weitere Problematik ergibt sich bei der Analyse der Mächtigkeiten alternativer Automatentypen und der Erweiterung der Bilddaten von der zweiwertigen Bitmap zu Grauwert-, Farb- und Bewegtbildern.

Literatur

[1] D.A. Huffman, *A method for the construction of minimum redundancy codes*, Proc. IRE 40 (1952), 10, pp. 1098-1101.

[2] Claude E. Shannon, Warren Weaver, *Mathematische Grundlagen der Informationstheorie*, Oldenbourg Verlag, München, 1976.

[3] J. Ziv, A. Lempel, *A Universal Algorithm for Sequential Data Compression*, IEEE Transactions on Information Theory, Vol. IT-23, No. 3, pp. 337-343, May 1977.

[4] Georg Reinhardt [Hrsg.], *A. Paul Weber, Das graphische Werk: Handzeichnungen und Lithographien 1930 - 1978*, Schirmer-Mosel, München, 1980.

[5] Terry A. Welch, *A Technique for High-Performance Data Compression*, IEEE Computer, Vol. 15, No. 6, pp. 8-19, June 1984.

[6] I.H. Witten, M.N. Radford, J.G. Cleary, *Arithmetic Coding for Data Compression*, Communications of the ACM, Vol. 30, No. 6, pp. 520-540, 1987.

[7] Michael F. Barnsley, Alan D. Sloan, *A Better Way to Compress Images*, Byte, No. 1, pp. 215-223, 1988.

[8] G. Drosdowski, *Duden Etymologie: Herkunftswörterbuch der deutschen Sprache*, Dudenverlag, Mannheim, Wien, Zürich, 1989.

[9] W. Heise, P. Quattrocchi, *Informations- und Codierungstheorie*, Springer Verlag, Berlin, Heidelberg, 1989.

[10] Gregory K. Wallace, *The JPEG Still Picture Compression Standard*, Communication of the ACM, Vol. 34, No. 4, pp. 40-44, 1991.

[11] H. Völz, *Grundlagen der Information*, Akademie Verlag, Berlin, 1991.

[12] Karel Culik II, Jarkko Kari, *Image-data compression using edge-optimizing algorithm for WFA inference*, Information Processing & Management, Vol. 30, No. 6, pp. 829-834, 1994.

[13] Frank Katritzke, *Über die redundanzvermindernde Codierung digitaler Bilddaten*, Diplomarbeit, Universität-Gesamthochschule Siegen, 1994.

[14] Michael Zettler, *Rekursive, hierarchische Zerlegungskodierung digitaler Bilddaten mit endlichen Automaten*, Diplomarbeit, Universität-Gesamthochschule Siegen, 1994.

A Modal μ–Calculus for Durational Transition Systems

Helmut Seidl

FB IV – Informatik
Universität Trier
D–54286 Trier

seidl@ti.uni-trier.de

Abstract

Durational transition systems are finite transition systems where every transition is additionally equipped with a duration. We consider the problem of interpreting μ–formulas over durational transition systems. In case the formula contains only operations minimum, maximum, addition, and sequencing, we show that the interpretation ist not only computable but (up to a linear factor) as efficiently computable as the interpretation of ordinary μ–formulas over finite transition systems.

Keywords: μ–Calculus, durational transition systems, model checking.

1 Introduction

The μ–calculus, a propositional modal logic with fixpoint operators, was introduced by Dexter Kozen as a tool for the description and analysis of the behavior of possibly infinite computations [Ko83]. The expressive power of modal μ–calculus is intimately connected to finite–state automata on infinite trees [VW86, EJ91, EJS93, BVW94]. Classically, a μ–formula denotes a predicate on states. Typical properties to be expressed and analyzed are *safety* and *lifeness* assertions. The formula $\nu x.\langle a \rangle x$, for example, denotes the set of all states allowing for an infinite sequence of a–actions. Especially, the modal operator $\langle a \rangle$ (for action a) constructs a property of the actual state from a property of a next state. Thus, it relates presence to a (possibly infinite) future. For detailed explanation of various modal (and temporal) logics consult, e.g., Colin Stirling in [St92].

The problem with which we are concerned here, however, is not only to determine properties of states but also quantitative information. We are interested in questions like: how long does it take a process to reach a state with a certain property. Technically, this means that closed formulas are no longer interpreted as predicates or, equivalently, as mappings $Q \to \{0, 1\}$ for some base set Q. Instead, we employ mappings $t : Q \to D$ for some time domain D. In this paper we consider D as a subinterval of

$$-\infty < 0 < 1 < 2 < \ldots < n < \ldots < \infty$$

i.e., the non–negative integers extended by $-\infty$ (to express unaccessibility) and ∞. The "logical" connectives "$\vee$ and "$\wedge$" are now replaced with maximum and minimum, respectively. Additionally, we consider addition "+" (appropriately extended) and the sequencing operator ";" which returns the second argument only provided the first one is different from $-\infty$. Otherwise, the result is $-\infty$. While addition is useful for modelling the time behavior of two processes being executed one after the other, the ";" may serve for a primitive form of conditional: the second argument is only counted provided the event corresponding to the first will ever occur.

A corresponding theory of processes with durational actions was developed by Gorrieri, Roccetti and Stancampiano [GRS95]. For the analysis of complexity-like properties of programs, e.g., the number of accesses to some variable, least solutions of (non–hierarchical) systems of equations over non–negative integers (extended by ∞) have been considered in [Se94]. This corresponds to the interpretation of μ–formulas without occurrences of the greatest fixpoint operator.

In this paper we are going to investigate whether or not the interpretation of arbitrary μ–formulas can be effectively computed – even when the chosen time interval is infinite. It turns out that this is in deed the case. Moreover, we find that it is not only computable but, even more, as efficiently (at least up to a linear factor) computable as the interpretation over a two point interval.

2 References

[ACD93] R. Alur, C. Courcoubetis, D.L. Dill: Model Checking in Dense Real Time. Inform. and Comput. 104, 2–34, 1993

[AD94] R. Alur, D.L. Dill: A Theory of Timed Automata. TCS 126, 183–235, 1994

[An92] H.R. Andersen: Model checking and Boolean Graphs. Proc. ESOP'92, LNCS 582, 1–19, 1992

[BVW94] O. Bernholtz, M.V. Vardi, P. Wolper: An Automata–Theoretic Approach to Branching–Time Model Checking (Extended Abstract). Proc. Conf. Computer Aided Verification, LNCS 818, 142–155, 1994

[BCM92] J.R. Burch, E.M. Clarke, K.L. McMillan: Symbolic Model Checking 10^{20} States and Beyond. Inf. and Comput. 98, 142–170, 1992

[EJ91] E.A. Emerson, C.S. Jutla: Tree Automata, Mu–Calculus and Determinacy (Extended Abstract). Proc. 32nd IEEE Symp. on FOCS, 368–377, 1991

[EJS93] E.A. Emerson, C.S. Jutla, A.P. Sistla: On Model–Checking for Fragments of μ–Calculus. Proc. Conf. on Computer Aided Verification, LNCS 697, 385–396, 1993

[GRS95] R. Gorrieri, M. Roccetti, E. Stancampiano: A Theory of Processes with Durational Actions. TCS 140, 73–94, 1995

[HNS94] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine: Symbolic Model Checking for Real–Time Systems. Inform. and Comput. 111, 193–244, 1994

[Ko83] D. Kozen: Results on the Propositional μ–Calculus. TCS 27, 333–354, 1983

[LS94] F. Laroussinie, Ph. Schoebelen: A Hierarchy of Temporal Logics with Past. Proc. 11th STACS, LNCS 775, 47–58, 1994; long version submitted to TCS

[LPS95] F. Laroussinie, S. Pinchinat, Ph. Schoebelen: Translations between Modal Logics of Reactice Systems. Special Issue on AMAST'93, TCS 140, 53–71, 1995

[LBC94] D.E. Long, A. Browne, E.M. Clarke, S. Jha, W.R. Marrero: An Improved Algorithm for the Evaluation of Fixpoint Expressions. Proc. Conf. on Computer Aided Verification, LNCS 818, 338–350, 1994

[Se94] H. Seidl: Least Solution of Equations over $\mathcal{N}$. Proc. ICALP'94, LNCS 820, 400–411, 1994

[St92] C. Stirling: Modal and Temporal Logics. In: S. Abramsky, D. Gabbay, and T. Maibaum (eds.): Handbook of Logic in Computer Science, 477–563, 1992

[Va88] M. Vardi: A Temporal Fixpoint Calculus. Proc. 15th POPL, 250–259, 1988

[VW86] M.V. Vardi, P. Wolper: Automata–Theoretic Techniques for Modal Logics of Programs. JCSS 32, 183–221, 1986

Uniformly Optimal Prediction

Ludwig Staiger
Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
Weinbergweg 17
D–06120 Halle
staiger@informatik.uni-halle.de

Abstract

The present paper links the concepts of Kolmogorov complexity (in Complexity theory) and Hausdorff dimension (in Fractal geometry) for a class of recursive (computable) ω-languages.

It is shown that the complexity of an infinite string contained in a Σ_2-definable set of strings is upper bounded by the Hausdorff dimension of this set and that this upper bound is tight. Moreover, we show that there are computable gambling strategies guaranteeing a uniform prediction quality arbitrarily close to the optimal one estimated by Hausdorff dimension and Kolmogorov complexity provided the gambler's adversary plays according to a sequence chosen from a Σ_2-definable set of strings.

We provide also examples which give evidence that our results do not extend further in the Arithmetical hierarchy.

The *Kolmogorov complexity* of a finite word w, $K(w)$, is, roughly speaking, the length of a shortest input (program) π for which a universal algorithm $\mathfrak{U}$ prints w (see e.g. [Ch87], [LV93]). For infinite words (ω-words) its (normalized) Kolmogorov complexity might be thought of as

$$\lim_{n\to\infty} \frac{K(\xi/n)}{n} \ , \tag{1}$$

where ξ/n denotes the prefix of length n of ξ.

Although this limit need not exist, Cai and Hartmanis [CH89] proved that the graph of the function $t : \xi \to [0,1]$ is a fractal regardless by which intermediate value

$$\underline{\kappa}(\xi) := \liminf_{n\to\infty} \frac{K(\xi/n)}{n} \leq t(\xi) \leq \kappa(\xi) := \limsup_{n\to\infty} \frac{K(\xi/n)}{n} \tag{2}$$

the possibly not existing limit in Eq. (1) is replaced.

Other evidence of the intimate relationship between fractals and Kolmogorov complexity was given in [Br82], [Ry86], [Ry93, 94], [St81] and [St93] where a different approach was pursued:

Given a (computable) set of infinite words (ω-language) F, bound the maximum possible Kolmogorov complexity $\underline{\kappa}(\xi)$ and $\kappa(\xi)$ for $\xi \in F$.

Here Ryabko [Ry86] showed that for arbitrary ω-languages F the Hausdorff dimension, $\dim F$, is a lower bound to $\underline{\kappa}(F) := \sup\{\underline{\kappa}(\xi) : \xi \in F\}$, but as Example 3.18 of [St93] shows already for simple computable ω-languages Hausdorff dimension is not an upper bound to $\kappa(F) := \sup\{\kappa(\xi) : \xi \in F\}$. Instead the results of [Br82], [St81] and [St93] show that the upper Kolmogorov complexity of ω-words in F is closely related to (metric) entropy, an information size measure different from Hausdorff dimension (cf. [Fa90]).

In [St93], however, we showed that for restricted classes of computable ω-languages F (that is, ω-languages satisfying additionally certain combinatorial properties) its Hausdorff dimension is also an upper bound to $\underline{\kappa}(F)$, thus giving a partial completion to Ryabko's lower bound.

After having introduced some notation and preliminaries in Section 1, we show in the the second part that Hausdorff dimension is an upper bound to lower Kolmogorov complexity for arbitrary Σ_2-definable ω-languages. Moreover, we give evidence that, unless other structural properties of ω-languages are involved, our upper bound does not extend further in the arithmetical hierarchy of ω-languages. This result tightens Ryabko's lower bound in the range of computable (recursive) ω-languages.

Then the third section considers the following prediction problem for infinite sequences (cf. [Ry93, 94]): A gambler plays a fair game against an adversary which draws its outcomes according to the symbols of an arbitrarily chosen infinite sequence $\xi \in X^\omega$. The prediction quality of the gambler's strategy is measured by the exponent of the increase of the gambler's capital. It was shown in [Ry93, 94] that this exponent $\lambda(\xi)$ is bounded from above by $1 - \underline{\kappa}(\xi)$ provided the gambler plays according to a computable strategy.

In this paper we investigate the case when the adversary is bound to draw its outcomes from a sequence $\eta \in F$ (unknown to the gambler) where F is a Σ_2-definable ω-language. We show that then there are computable strategies guaranteeing a prediction quality $\lambda(\eta)$ arbitrarily close to the value $(1-\dim F)$ uniformly on F, that is, regardless from which $\eta \in F$ the adversary draws its outcomes.

Finally, Section 4 gives some connections to previous work, in particular, upper bounds on Kolmogorov complexity via Hausdorff dimension for ω-languages satisfying additional combinatorial properties are considered.

1 Notation and Preliminary Results

By $\mathbb{N} = \{0, 1, 2, \ldots\}$ we denote the set of natural numbers. We consider the space X^ω of infinite strings (sequences, ω-words) on a finite alphabet of cardinality $r := card\, X \geq 2$. By X^* we denote the set (monoid) of finite strings (words) on X, including the *empty* word e. For $w \in X^*$ and $b \in X^* \cup X^\omega$ let $w \cdot b$ be their *concatenation*. This concatenation product extends in an obvious way to subsets $W \subseteq X^*$ and $B \subseteq X^* \cup X^\omega$. As usual we denote subsets of X^* as languages and subsets of X^ω as ω-languages.

Furthermore $|w|$ is the *length* of the word $w \in X^*$, hence $X^n = \{w : w \in X^* \wedge |w| = n\}$. As introduced above by b/n we denote the *length n prefix* of a string $b \in X^*$, $|b| \geq n$, or $b \in X^\omega$, and $\mathbf{A}(b) := \{b/n : n \in \mathbb{N} \wedge n \leq |b|\}$ and $\mathbf{A}(B) := \bigcup_{b\in B} \mathbf{A}(b)$ are the sets of all finite prefixes of $b \in X^* \cup X^\omega$ and $B \subseteq X^* \cup X^\omega$, respectively.

As usual we define Π_1*-definable* ω*-languages* $E \subseteq X^\omega$ as

$$E = \{\xi : \forall n(n \in \mathbb{N} \to \xi/n \notin W_E)\} \tag{3}$$

where $W_E \subseteq X^*$ is a recursive language, and we define Σ_2*-definable* ω*-languages* $F \subseteq X^\omega$ as

$$F = \{\xi : \exists i(i \in \mathbb{N} \wedge \forall n(n \in \mathbb{N} \to (i, \xi/n) \notin M_F))\} \tag{4}$$

where M_F is a recursive subset of $\mathbb{N} \times X^*$.

Define $L_i := \{w : (i, w) \in M_F\}$. Then Eqs. (3) and (4) may be alternatively written as

$$E = X^\omega \setminus W_E \cdot X^\omega \ , \quad \text{and} \tag{5}$$

$$F = \bigcup_{i\in\mathbb{N}} (X^\omega \setminus L_i \cdot X^\omega) = \bigcup_{k\in\mathbb{N}} (X^\omega \setminus (\bigcap_{i=0}^{k} L_i \cdot X^*) \cdot X^\omega) \ . \tag{6}$$

In particular, every Σ_2-definable ω-language is a countable union of Π_1-definable ω-languages.

From the definitions and Eqs. (5) and (6) it is clear that we may think of W_E and M_F as upwards closed, that is, $W_E = W_E \cdot X^*$ and M_F satisfies the condition

$$\begin{array}{lcl} (k, w) \in M_F & \Longrightarrow & \forall v(v \in X^* \to (k, w \cdot v) \in M_F) \ \text{ and} \\ (k, w) \in M_F & \Longrightarrow & \forall i(i \leq k \to (i, w) \in M_F) \ , \end{array} \tag{7}$$

that is, $L_i = L_i \cdot X^*$ and $L_i \supseteq L_{i+1}$.

In order to give a more detailed analysis we consider X^ω as a topological space (Cantor space), where the open subsets are defined as $W \cdot X^\omega$ $(W \subseteq X^*)$. It should be mentioned that this space is compact, hence for every closed subset $F \subseteq X^\omega$ the inclusion $F \subseteq W \cdot X^\omega$

implies that there is already a finite subset $W' \subseteq W$ for which $F \subseteq W' \cdot X^\omega$.

It is known (e.g. [St86, Theorem 6.2]) that a subset $F \subseteq X^\omega$ is Π_1-definable iff F is closed and $X^* \setminus \mathbf{A}(F)$ is recursively enumerable, consequently, a Σ_2-definable subset $E \subseteq X^\omega$ is a countable union of closed sets (a so-called $\mathbf{F}_\sigma$*-set*).

The Hausdorff dimension of an ω-language $F \subseteq X^\omega$ may be defined using the entropy of languages. To this end we introduce the *structure function* of a language $L \subseteq X^*$, $\boldsymbol{s}_L : \mathbb{N} \to \mathbb{N}$, as

$$\boldsymbol{s}_L(n) := \mathit{card}\, L \cap X^n \ ,$$

and we define its *entropy*

$$H_L := \limsup_{n \to \infty} \frac{\log_r(1 + \boldsymbol{s}_L(n))}{n} \ .$$

In other words, we have the following implications.

$$\sum\nolimits_{n\in\mathbb{N}} \boldsymbol{s}_L(n) \cdot r^{-\alpha\cdot n} = \sum\nolimits_{v\in L} r^{-\alpha\cdot|v|} < \infty \quad \to \quad \alpha \geq H_L \qquad (8)$$

$$\sum\nolimits_{n\in\mathbb{N}} \boldsymbol{s}_L(n) \cdot r^{-\alpha\cdot n} = \sum\nolimits_{v\in L} r^{-\alpha\cdot|v|} = \infty \quad \to \quad \alpha \leq H_L \qquad (9)$$

Moreover, let $V^\delta := \{\xi : \xi \in X^\omega \wedge \mathbf{A}(\xi) \cap V \text{ is infinite}\}$ be the δ*-limit* of the language $V \subseteq X^*$. Then according to Eq. (3.11) of [St93]

$$\dim F := \inf\{H_W : W \subseteq X^* \wedge F \subseteq W^\delta\} \qquad (10)$$

is the *Hausdorff dimension* of $F \subseteq X^\omega$.

In particular, following [St93, Lemma 3.10], we have for $\alpha > \dim F$

$$\forall\varepsilon\big(\varepsilon > 0 \to \exists W(W \subseteq X^* \wedge F \subseteq W \cdot X^\omega \wedge \sum_{w\in W} r^{-\alpha\cdot|w|} < \varepsilon)\big) \ . \qquad (11)$$

We mention still a relation between lower Kolmogorov complexity $\underline{\kappa}$ and the entropy of languages (cf. [St93, Proposition 2.14]).

Lemma 1 *If $L \subseteq X^*$ and L or $X^* \setminus L$ are recursively enumerable languages then*

$$\underline{\kappa}(\xi) \leq H_L \quad \textit{for all} \quad \xi \in L^\delta \ .$$

2 An Upper Bound for Kolmogorov Complexity

In the first section we derive the upper bound for $\underline{\kappa}(\xi)$ by the Hausdorff dimension of $F \subseteq X^\omega$, $\dim F$, provided $\xi \in F$ and F is Σ_2-definable.

To this end we derive a recursive analogue to the definition of the Hausdorff dimension (cf. Eq. (10)).

Theorem 2 *If $F \subseteq X^\omega$ is a Σ_2-definable ω-language then*

$$\dim F = \inf \{H_V : F \subseteq V^\delta \wedge V \subseteq X^* \text{ is recursive}\} .$$

Remark: It should be mentioned that an ω-language $F \subseteq X^\omega$ is Σ_2-definable if and only if there is a recursive language $W \subseteq X^*$ such that $F = X^\omega \setminus W^\delta$ (cf. [St86, Theorem 7.4]). Accordingly, an ω-language $E \subseteq X^\omega$ is Π_2-*definable* iff there is a recursive language $V \subseteq X^*$ such that $E = V^\delta$, but as we shall see below in Lemma 4 our recursive analogue to the definition of the Hausdorff dimension is not valid for Π_2-definable ω-languages.

Proof. We describe an algorithm which constructs for every $\alpha > \dim F$ a recursive language V_α such that $F \subseteq V_\alpha^\delta$. To this end let $(U_j)_{j\in\mathbb{N}}$ be an effective enumeration of all finite prefix codes in X^* such that $\sup\{|v| : v \in U_j\} \le \sup\{|v| : v \in U_{j+1}\}$, and let the languages $L_k \subseteq X^*$ defining F be given according to Eq. (6) as $L_k := (\bigcap_{i=0}^k L_i \cdot X^*)$.
Define

$$\mathbf{test}_\alpha(k,j,n) :\Leftrightarrow \Big(\big(U_j \cup (L_k \cap X^n)\big) \cdot X^\omega = X^\omega \wedge \sum_{v\in U_j} r^{-\alpha\cdot|v|} < r^{-k}\Big) .$$

Observe that $\mathbf{test}_\alpha(k,j,n)$ is effectively computable provided $r^{-\alpha}$ is rational and that if $\mathbf{test}_\alpha(k,j,n)$ is true then $F \subseteq U_j \cdot X^\omega$, because $L_k \cdot X^\omega \cap F = \emptyset$.

Now the following algorithm, when given M_F, computes a finite prefix code C_k satisfying $F \subseteq C_k \cdot X^\omega$ and $\sum_{w\in C_k} r^{-\alpha\cdot|w|} < r^{-k}$:

Algorithm C_k

input $k,\ r^\alpha$
$n = 0$
repeat $j = -1$
repeat $j = j + 1$
until $\mathbf{test}_\alpha(k, j, n) \vee \big(\sup\{|v| : v \in U_j\} > n\big)$
$n = n + 1$
until $\mathbf{test}_\alpha(k, j, n)$
output $C_k := U_j$

Informally, our algorithm successively for every $n \geq 0$ searches for a U_j satisfying the condition $\mathbf{test}_\alpha(k, j, n)$, more precisely, it searches until such a U_j is found or else all U_j having $\sup\{|v| : v \in U_j\} \leq n$ fail to satisfy $\mathbf{test}_\alpha(k, j, n)$.
In the latter case the value of n is increased (thus resulting in allowing for a larger codeword length and in enlarging the complementary ω-language $(L_k \cap X^n) \cdot X^\omega$) and the search starts anew. Consequently, the algorithm terminates iff there is a finite prefix code U such that $\sum_{v\in U} r^{-\alpha\cdot|v|} < r^{-k}$ and $U \cdot X^\omega \cup (L_k \cap X^n) \cdot X^\omega = X^\omega$ for some $n \in \mathbb{N}$.

We have still to verify that our algorithm always terminates provided $\alpha > \dim F$. To this end we observe that according to Eq. (11) for every $\varepsilon > 0$ there is a $W \subseteq X^*$ such that $F \subseteq W \cdot X^\omega$ and $\sum_{w\in W} r^{-\alpha\cdot|w|} < \varepsilon$.

Since $X^\omega \setminus L_k \cdot X^\omega$ is a closed subset of F, for $\varepsilon \leq r^{-k}$ we find a finite subset $W_k \subseteq W$ such that $X^\omega \setminus L_k \cdot X^\omega \subseteq W_k \cdot X^\omega$. Consequently, there is also a finite prefix code U_j satisfying $(U_j \cup L_k) \cdot X^\omega = X^\omega$ and thus $(U_j \cup (L_k \cap X^n)) \cdot X^\omega = X^\omega$ for n large enough.

Now let $V_\alpha := \bigcup_{k\in\mathbb{N}} C_k$. Obviously, V_α is recursively enumerable if $r^{-\alpha}$ is rational. Note that V_α is even recursive: Since $\alpha > 0$ and $w \in C_k$ imply $r^{-\alpha\cdot|w|} < r^{-k}$, we have $w \in V_\alpha$ iff $\exists k(k < |w| \cdot \alpha \wedge w \in C_k)$.

The predicate $k < |w| \cdot \alpha$ is recursive and bounds the quantifier $\exists k$ from above.

Next, $\sum_{w \in V_\alpha} r^{-\alpha \cdot |w|} \leq \sum_{k \in \mathbb{N}} \sum_{w \in C_k} r^{-\alpha \cdot |w|} \leq \sum_{k \in \mathbb{N}} r^{-k} \leq 2 < \infty$, whence $H_{V_\alpha} \leq \alpha$.

It remains to show that $F \subseteq V_\alpha^\delta$. If $\xi \in F$ there is a k such that $\xi \in X^\omega \setminus L_i \cdot X^\omega$ for all $i \geq k$. Hence, ξ has a prefix $w_i \in C_i$ for $i \geq k$. As it was observed above, $|w_i| > i/\alpha$. Consequently, ξ has infinitely many prefixes in $V_\alpha = \bigcup_{i \in \mathbb{N}} C_i$. □

Utilizing Lemma 1 we obtain the announced upper bound on $\underline{\kappa}(F)$ by $\dim F$.

Theorem 3 *If $F \subseteq X^\omega$ is a Σ_2-definable ω-language then $\underline{\kappa}(\xi) \leq \dim F$ for all $\xi \in F$.*

Observe that, since Hausdorff dimension is countably stable, that is,

$$\dim \bigcup_{i \in \mathbb{N}} F_i = \sup \dim F_i \ , \tag{12}$$

this result immediately extends to arbitrary countable unions of Σ_2-definable (or, equivalently, to countable unions of Π_1-definable) ω-languages $\bigcup_{j \in \mathbb{N}} F_j$.

The bound of Theorem 3 cannot be tightened to ensure that there is indeed a $\xi \in F$ such that $\underline{\kappa}(\xi) = \dim F$, as the following example shows.

Example 1 *Let $E := \{0\}^\omega \cup \bigcup_{n \in \mathbb{N}} 0^n \cdot (1 \cdot X^n)^\omega$. Then $\mathbf{A}(E)$ is a recursive language, hence E is Π_1-definable. In the same way one shows that the ω-languages $\{0\}^\omega$ and $0^n \cdot (1 \cdot X^n)^\omega$ are also Π_1-definable. Now, similar to [St93, Example 4.11] one computes*

$$\dim \ 0^n \cdot (1 \cdot X^n)^\omega = \frac{n}{n+1} \ .$$

Consequently, $\dim E = 1 > \frac{n}{n+1} \geq \underline{\kappa}(\xi)$ *for* $\xi \in 0^n \cdot (1 \cdot X^n)^\omega$.

Next, we show that the upper bound of Theorem 3 cannot be extended further to higher classes of the Arithmetical hierarchy of ω-languages.

To this end we consider the class

$$\mathfrak{S} := \{E : E \subseteq X^\omega \wedge E \text{ is closed } \wedge \mathbf{A}(E) \text{ is recursively enumerable}\} ,$$

which in some sense seems to be dual to the class of Π_1-definable ω-languages. Since $E \in \mathfrak{S}$ implies $E = \mathbf{A}(E)^\delta$, every $E \in \mathfrak{S}$ is Π_2-definable, but as the remark following Corollary 7.2 in [St86] shows, not every Π_2-definable and closed ω-language belongs to $\mathfrak{S}$.

In [St86] it was asked for a "nice" characterization of the class $\mathfrak{S}$. Already in Example 1.15 of [St93] it was explained that except for the above mentioned Π_2-definability no other relationship to the classes of the arithmetical hierarchy of ω-languages can be established. Here we give further evidence of this latter fact resulting in a proof that an analogue of Theorem 3 cannot hold for the class $\mathfrak{S}$. More precisely, we present a countable ω-language $E \in \mathfrak{S}$ which except for a single limit point $\xi \in E$ is Σ_2-definable but does not satisfy $\underline{\kappa}(E) \leq \dim E$.

Lemma 4 *There is a countable ω-language $E \in \mathfrak{S} \setminus \Sigma_2$ such that E is the closure (in Cantor space) of a Σ_2-definable ω-language E' with $\underline{\kappa}(\xi) = 1$ for $\xi \in E \setminus E'$.*

Proof. P. Martin-Löf (cf. [LV93] or [Ca94]) proved that a random sequence $\zeta \in X^\omega$ has $\underline{\kappa}(\zeta) = 1$ and that the set of all random sequences $\mathfrak{R}_{\mathrm{rand}}$ is Σ_2-definable. Hence, $\mathfrak{R}_{\mathrm{rand}}$ contains a nonempty Π_1-definable subset $F = X^\omega \setminus W_F \cdot X^\omega$, where $W_F \subseteq X^*$ is a recursive language.

Fix the natural order on $X = \{0, 1, \ldots, r-1\}$ and extend it to a quasilexicographical order on X^*. Since F is closed in the Cantor topology of X^ω, it contains a leftmost sequence ζ_{left}.[1]

[1] This corresponds to the obvious interpretation of ω-words $\zeta \in X^\omega$ as r-ary expansions of real numbers $\iota(\zeta) := 0.\zeta \in [0,1]$. The mapping $\iota : X^\omega \to [0,1]$ is continuous and is one-to-one on $X^\omega \setminus X^* \cdot \{0^\omega, (r-1)^\omega\} \supseteq \mathfrak{R}_{\mathrm{rand}}$.

We show that there is a countable ω-language $E \in \mathfrak{S}$ such that E contains the leftmost sequence $\zeta_{left} \in F$.

Define for every $n \in \mathbb{N}$ the word v_n as the lexicographically first word in $X^n \setminus W_F \cdot X^*$. Obviously, the language $V := \{v_n : n \in \mathbb{N}\}$ is recursive, and $\zeta_n := v_n \cdot 0^\omega$ is the leftmost ω-word in $(X^n \setminus W_F \cdot X^*) \cdot X^\omega \supseteq F$.

Since $(X^n \setminus W_F \cdot X^*) \cdot X \supseteq X^{n+1} \setminus W_F \cdot X^*$, the family $\big((X^n \setminus W_F \cdot X^*) \cdot X^\omega\big)_{n \in \mathbb{N}}$ of closed subsets converges to $F = \bigcap_{n \in \mathbb{N}} (X^n \setminus W_F \cdot X^*) \cdot X^\omega$, and since ζ_n is the leftmost point in $(X^n \setminus W_F \cdot X^*) \cdot X^\omega \supseteq F$, we have also $\lim_{n \to \infty} \zeta_n = \zeta_{left}$ whence $\bigcup_{n \in \mathbb{N}} \mathbf{A}(\zeta_n) \supseteq \mathbf{A}(\zeta_{left})$.

Defining $E' := \{\zeta_n : n \in \mathbb{N}\} = V \cdot 0^\omega$ and $E := E' \cup \{\zeta_{left}\}$ yields that E is closed, $\mathbf{A}(E) = \mathbf{A}(V) \cup V \cdot 0^*$ and E' is Σ_2-definable. $\square$

Since E is countable, $\dim E = 0$, and consequently, neither the analogue of Theorem 3 nor the recursive version of the definition of the Hausdorff dimension (Theorem 2) can hold for the class $\mathfrak{S}$. One can only show the following upper bound on $\underline{\kappa}(E)$ related to the structure function $\boldsymbol{s}_{\mathbf{A}(E)}$ (see [St93, Proposition 2.11]).

Theorem 5 *If $E \subseteq X^\omega$ is an ω-language such that $\mathbf{A}(E)$ or the complement $X^* \setminus \mathbf{A}(E)$ is recursively enumerable then*

$$\underline{\kappa}(\xi) \leq \liminf_{n \to \infty} \frac{\log_r \boldsymbol{s}_{\mathbf{A}(E)}(n)}{n} \quad \textit{for all} \quad \xi \in E \ .$$

As a further consequence of the proof of Lemma 4 one obtains the following result on ω-languages in the arithmetical hierarchy.

Corollary 6 *There is an ω-language $E \in \mathfrak{S}$ which is not representable as a countable union of Σ_2-definable ω-languages.*

3 Computable Martingales and Prediction Quality

Quite recently Ryabko [Ry93, 94] proved another kind of relationship between Hausdorff dimension and Kolmogorov complexity. He considered a problem which is related to the prediction problem of infinite sequences. This problem, addressed to in the introduction, can be easily described using Martingales, Ville's formalism of the concept of gambling strategy.

Here we consider Martingales as functions $\mathcal{V} : X^* \to [0, \infty)$ which satisfy

$$\mathcal{V}(e) > 0 \ \text{ and } \ \mathcal{V}(w) = \frac{\sum_{x \in X} \mathcal{V}(wx)}{card\, X} ,$$

and we denote by $\Lambda_{\mathcal{V}}(w) := \log_{card\, X} \mathcal{V}(w)$ the *exponent of the increase (pay-off)* of $\mathcal{V}$.

Let $\mathcal{V}$ be a computable Martingale (or as in [Ry93, 94]: a Turing prediction strategy). Then in [Ry86, 93, 94] it is proved that for all $\xi \in X^\omega$

$$\begin{aligned} \lambda_{\mathcal{V}}(\xi) \quad &:= \quad \limsup_{w \to \xi} \frac{\Lambda_{\mathcal{V}}(w)}{|w|} \leq 1 - \underline{\kappa}(\xi) \quad \text{, and} && (13) \\ \dim F \quad &\leq \quad \sup\{\underline{\kappa}(\xi) : \xi \in F\} \ . && (14) \end{aligned}$$

But it is also mentioned that there is no optimal prediction method, that is, there is no computable Martingale $\mathcal{V}_{\mathrm{opt}}$ such that $\lambda_{\mathcal{V}_{\mathrm{opt}}}(\xi) \geq \lambda_{\mathcal{V}}(\xi)$ for all $\xi \in X^\omega$ and all computable Martingales $\mathcal{V}$.

Our aim is to prove that for every Σ_2-definable ω-language F and every $\alpha > \dim F$ there is a computable Martingale $\mathcal{V}$ such that $\lambda_{\mathcal{V}}(\xi) \geq 1 - \alpha$ for all $\xi \in F$, thus being nearly uniformly optimal for the ω-language F.

In order to achieve our goal we introduce families of covering codes as follows.

For a prefix code $C \subseteq X^*$ we define its *minimal complementary code* as

$$\widehat{C} := (X \cup \mathbf{A}(C) \cdot X) \setminus \mathbf{A}(C) \ . \tag{15}$$

If $C = \emptyset$ we have $\widehat{C} = X$, and if $C \neq \emptyset$ the set $\widehat{C}$ consists of all words $w \cdot x \notin \mathbf{A}(C)$ where $w \in \mathbf{A}(C)$ and $x \in X$. It is readily seen that $C \cup \widehat{C}$ is a maximal prefix code, $C \cap \widehat{C} = \emptyset$, and $\mathbf{A}(C \cup \widehat{C}) = \{e\} \cup \mathbf{A}(C) \cup \widehat{C}$.

We call $\mathfrak{C} := (C_w)_{w \in X^*}$ a *family of covering codes* provided each C_w is a finite prefix code. Since then the set $C_w \cup \widehat{C}_w$ is a finite maximal prefix code, every word $u \in X^*$ has a uniquely specified $\mathfrak{C}$-factorization $u = u_1 \cdots u_n \cdot u'$ where $u_{i+1} \in C_{u_1 \cdots u_i} \cup \widehat{C}_{u_1 \cdots u_i}$ for $i = 0, \ldots, n-1$ ($u_1 \cdots u_i = e$, if $i = 0$) and $u' \in \mathbf{A}(C_{u_1 \cdots u_n} \cup \widehat{C}_{u_1 \cdots u_n})$. Analogously, every $\xi \in X^\omega$ has a uniquely specified $\mathfrak{C}$-factorization $\xi = u_1 \cdots u_i \cdots$ where $u_{i+1} \in C_{u_1 \cdots u_i} \cup \widehat{C}_{u_1 \cdots u_i}$ for $i = 1, \ldots$.

In the sequel we use Martingales derived from prefix codes in the following manner.

Lemma 7 *Let $0 \leq \alpha \leq 1$ and $\emptyset \neq C \subseteq X^*$ be a prefix code satisfying $\sum_{v \in C} r^{-\alpha|v|} < \infty$. Then there is a Martingale $\mathcal{V}_C^{(\alpha)} : X^* \to \mathbb{R}_+$ such that*

$$\mathcal{V}_C^{(\alpha)}(w) = \begin{cases} \dfrac{r^{(1-\alpha)|w|}}{\sum_{v \in C} r^{-\alpha|v|} + \sum_{u \in \widehat{C}} r^{-|u|}} & , \textit{for } w \in C \ , \textit{ and} \\[2ex] \dfrac{1}{\sum_{v \in C} r^{-\alpha|v|} + \sum_{u \in \widehat{C}} r^{-|u|}} & , \textit{for } w \in \widehat{C} \ . \end{cases} \tag{16}$$

Proof. Set $\Gamma := \sum_{v \in C} r^{-\alpha|v|} + \sum_{u \in \widehat{C}} r^{-|u|}$, and define for $u \in \mathbf{A}(C \cup \widehat{C})$ and $w \in C \cup \widehat{C}$, $v \in X^*$

$$\begin{aligned} \mathcal{V}_C^{(\alpha)}(u) &:= \frac{r^{|u|}}{\Gamma} \cdot \left(\sum_{u \cdot w \in C} r^{-\alpha|u \cdot w|} + \sum_{u \cdot w \in \widehat{C}} r^{-|u \cdot w|} \right) \\ \mathcal{V}_C^{(\alpha)}(w \cdot v) &:= \mathcal{V}_C^{(\alpha)}(w) \ . \end{aligned}$$

Then $\mathcal{V}_C^{(\alpha)}$ fulfills Eq. (16). We have still to show that $\mathcal{V}_C^{(\alpha)}(u) = \frac{1}{r} \sum_{x \in X} \mathcal{V}_C^{(\alpha)}(ux)$. This identity is obvious if $u \notin \mathbf{A}(C \cup \widehat{C})$.

Now, let $u \in \mathbf{A}(C \cup \widehat{C})$. Then

$$\begin{aligned}
\sum_{x\in X} \frac{\mathcal{V}_C^{(\alpha)}(ux)}{r} &= \sum_{x\in X} \frac{r^{|ux|}}{r\cdot \Gamma} \cdot \left(\sum_{uxw\in C} r^{-\alpha|uxw|} + \sum_{uxw\in\widehat{C}} r^{-|uxw|} \right) \\
&= \frac{r^{|u|}}{\Gamma} \cdot \sum_{x\in X} \left(\sum_{uxw\in C} r^{-\alpha|uxw|} + \sum_{uxw\in\widehat{C}} r^{-|uxw|} \right) ,
\end{aligned}$$

because for $u \in \mathbf{A}(C \cup \widehat{C}) \setminus (C \cup \widehat{C})$ the set $\{w : w \in C \cup \widehat{C} \wedge u \sqsubseteq w\}$ partitions into the sets $\{w : w \in C \cup \widehat{C} \wedge ux \sqsubseteq w\}$ $(x \in X)$, and the required equation follows. □

Remark. If C is a finite prefix code and $r^{-\alpha}$ is a rational number then $\mathcal{V}_C^{(\alpha)}$ is a computable (recursive) Martingale.

Now let be given a family of covering codes $\mathfrak{C} := (C_w)_{w\in X^*}$ such that $\sum_{v\in C_w} r^{-\alpha|v|} < \infty$, and let $\mathcal{V}_{C_w}^{(\alpha)}$ be as defined above. Then we define our Martingale $\mathcal{V}$ as follows:

For $u \in X^*$ consider the $\mathfrak{C}$-factorization $u_1 \cdots u_n \cdot u'$, and put

$$\mathcal{V}_{\mathfrak{C}}^{(\alpha)}(u) := \left(\prod_{i=0}^{n-1} \mathcal{V}_{C_{u_1\cdots u_i}}^{(\alpha)}(u_{i+1}) \right) \cdot \mathcal{V}_{C_{u_1\cdots u_n}}^{(\alpha)}(u') \ , \tag{17}$$

that is, $\mathcal{V}_{\mathfrak{C}}^{(\alpha)}$ is in some sense the concatenation of the Martingales $\mathcal{V}_{C_w}^{(\alpha)}$. Observe that $\mathcal{V}_{\mathfrak{C}}^{(\alpha)}$ is computable if only $r^{-\alpha}$ is rational and the function which assigns to every w the corresponding code C_w is computable.

We have the following.

Lemma 8 *Let $\mathfrak{C} = (C_w)_{w\in X^*}$ be a family of covering codes such that $\sum_{v\in C_w} r^{-\alpha|v|} < r^{-|w|}$ for some $\alpha \in (0,1)$, and let $\xi \in X^\omega$ have a $\mathfrak{C}$-factorization $\xi = u_1 \cdots u_i \cdots$ such that all factors u_{i+1} belong to $C_{u_1\cdots u_i}$ whenever $i \geq n_\xi$. Then there is a constant $c_\xi > 0$ not depending on i for which*

$$\mathcal{V}_{\mathfrak{C}}(u_1 \cdots u_i) \geq c_\xi \cdot r^{(1-\alpha)\cdot|u_1\cdots u_i|} \ .$$

Proof. Since $\widehat{C}_w$ is a code, we have $\sum_{v\in\widehat{C}_w} r^{-|v|} \leq 1$, and from the assumption we obtain

$$\sum_{v\in C_w} r^{-\alpha|v|} + \sum_{v\in\widehat{C}_w} r^{-|v|} \leq r^{-|w|} + 1 \ .$$

Now $|u_i| \geq 1$ and the above Eq. (16) yield

$$\mathcal{V}^{(\alpha)}_{C_{u_1\cdots u_i}}(u_{i+1}) \geq \begin{cases} \dfrac{1}{r^{-i}+1} & , \text{ if } i \leq n_\xi \ , \text{ and} \\[2ex] \dfrac{r^{(1-s)\cdot|u_{i+1}|}}{r^{-i}+1} & , \text{ if } i > n_\xi \ . \end{cases} \tag{18}$$

Put

$$c_\xi := \prod_{i=0}^{\infty} \frac{1}{r^{-i}+1} \cdot \prod_{i=0}^{n_\xi} r^{-(1-s)\cdot|u_{i+1}|} \ .$$

Clearly, $c_\xi > 0$, and using Eq. (18) by induction on i the assertion is easily verified. □

Now we prove the announced result.

Theorem 9 *For every Σ_2-definable ω-language $F \subseteq X^\omega$ and every $\alpha > \dim F$ there is a recursive Martingale $\mathcal{V}$ such that*

$$\lambda_{\mathcal{V}}(\xi) \geq 1 - \alpha \quad \textit{for all } \xi \in F \ .$$

Proof. If $\dim F = 1$ the assertion is obvious. Let $\dim F < 1$. We have to construct for every α such that $1 > \alpha > \dim F$ and r^α is rational a computable Martingale $\mathcal{V}$ satisfying the assertion.

In virtue of Lemma 8 it suffices to construct a computable family of covering codes $\mathfrak{C} = (C_w)_{w\in X^*}$ such that the function which assigns to every w the corresponding finite prefix code C_w is computable.

To this end we modify the predicate **test** introduced in the proof of Theorem 3 as follows.

$$\begin{aligned} \mathbf{test}'_\alpha(w,j,n) \quad :\Leftrightarrow \quad & \Big(\big(w \cdot U_j \cup (L_{|w|} \cap X^{|w|+n}) \big) \cdot X^\omega \supseteq w \cdot X^\omega \\ & \wedge \sum_{v\in U_j} r^{-\alpha\cdot|v|} < r^{-|w|} \Big) \ . \end{aligned}$$

In the same way we modify the algorithm presented there.

Algorithm C_w

input $w,\ r^\alpha$
$n = 0$
repeat $j = -1$
repeat $j = j + 1$
until $\mathbf{test}'_\alpha(w, j, n) \vee \big(\sup\{|v| : v \in U_j\} > n\big)$
$n = n + 1$
until $\mathbf{test}'_\alpha(w, j, n)$
output $C_w := U_j$

This algorithm computes a prefix code C_w with $\sum_{v \in C_w} r^{-\alpha \cdot |v|} < r^{-|w|}$ and $w \cdot C_w \cdot X^\omega \supseteq w \cdot X^\omega \setminus L_{|w|} \cdot X^\omega$, and it terminates for $\alpha > \dim F$, because $\dim(F \cap w \cdot X^\omega) \leq \dim F < \alpha$.

It remains to show that every $\xi \in F$ has a $\mathfrak{C}$-factorization $\xi = u_1 \cdots u_i \cdots$ such that almost all factors u_{i+1} belong to the corresponding $C_{u_1 \cdots u_i}$.

Let $\xi \in F$. Then there is a $k \in \mathbb{N}$ such that $\xi \in X^\omega \setminus L_i \cdot X^\omega$ for all $i \geq k$. Consequently, $w \in \mathbf{A}(\xi)$ implies $w \notin L_k$, and according to the definition of $\mathfrak{C}$ there is a $u \in C_w$ such that $w \cdot u \in \mathbf{A}(\xi)$ whenever $|w| \geq k$. $\square$

We cannot improve Theorem 9 much further, because on the one hand Theorem 3 and Eq. (13) show that it is not possible to construct a computable Martingale $\mathcal{V}$ such that $\forall \xi (\xi \in F \to \lambda_{\mathcal{V}}(\xi) \geq 1 - \alpha)$ when $\alpha < \dim F$, and on the other hand again the ω-language E defined in the proof of Lemma 4 shows that the result of Theorem 9 cannot be extended to further classes of the Arithmetical hierarchy.

4 Relation to previous results

In the introduction we mentioned that under certain additional structural constraints the result of Theorem 3 extends beyond the range of Σ_2-definable ω-languages. In this section we recall some of the results of [St93] giving evidence of this fact, but showing also that the constraints involved are not recursion-theoretic ones.

The first two classes of ω-languages satisfying properties analogous to Theorems 2 and 3 are the classes of recursive ω-power languages and of δ-limits of recursive submonoids.

An *ω-power language* is an ω-language of the form $W^\omega := \{\xi : \xi \in X^\omega \wedge \exists (w_i)_{i\in\mathbb{N}}(w_i \in W \wedge w_i \neq e \wedge \xi = w_0 \cdots w_i \cdots)\}$, where $W \subseteq X^*$, and the *submonoid* of X^* generated by W, W^*, is the language $W^* := \{w_1 \cdots w_n : n \in \mathbb{N} \wedge w_i \in W\}$.

Now Eq. 6.2 of [St93] is an analogue to Theorem 2 for submonoids $W^* \subseteq X^*$.

$$\dim W^\omega = \dim (W^*)^\delta = H_{W^*} \tag{19}$$

Then Ryabko's lower bound $\dim W^\omega \leq \underline{\kappa}(W^\omega)$ in connection with Lemma 1 shows the following.

Proposition 10 *If $W \subseteq X^*$ or its complement $X^* \setminus W$ are recursively enumerable languages then $\underline{\kappa}(W^\omega) = \underline{\kappa}\left((W^*)^\delta\right) = \dim W^\omega$.*

According to the discussion in Section 6 of [St93], the class $\{W^\omega : W \subseteq X^* \wedge W \text{ is recursive}\}$ is incomparable to the class of all Σ_2-definable subsets of X^ω.

A second class which exhibits a similar property is the class of so-called balanced ω-languages introduced in [St89]. We call a subset F of X^ω *balanced* if and only if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that $f(n) = o(2^{c\cdot n})$ for arbitrary $c > 0$ and for which the inequality

$$\boldsymbol{s}_F(w, |w| + n) \leq f(|w|) \cdot \frac{\boldsymbol{s}_{\mathbf{A}(F)}(|w| + n)}{\boldsymbol{s}_{\mathbf{A}(F)}(|w|)} \tag{20}$$

holds for arbitrary $w \in X^*$ and $n \in \mathbb{N}$. Here $\boldsymbol{s}_F(w,\cdot)$ is a shorthand for the structure function of the language $U(F,w) := \mathbf{A}(F) \cap w \cdot X^*$, $\boldsymbol{s}_{U(F,w)}$.
Roughly speaking, our Eq. (20) means that for arbitrary $w \in X^*$ the function $\boldsymbol{s}_F(w,\cdot)$ does not grow much faster than the average taken over all functions $\boldsymbol{s}_F(v,\cdot)$ such that $|v| = |w|$ and $v \in \mathbf{A}(F)$.

Then Theorem 4 of [St89] proves that for closed and balanced $F \subseteq X^\omega$ the following identity holds true.

$$\dim F = \liminf_{n\to\infty} \frac{\log_r \boldsymbol{s}_{\mathbf{A}(F)}(n)}{n} \tag{21}$$

Consequently, Theorem 5 implies an upper bound on $\underline{\kappa}(F)$ for balanced ω-languages in $\mathfrak{S}$.

Lemma 11 *If $F \in \mathfrak{S}$ is a balanced ω-language then $\underline{\kappa}(F) \leq \dim F$.*

We conclude this section with mentioning Corollary 3.17 of [St93] which proves that for closed and balanced ω-languages F there is always a $\xi \in F$ such that $\underline{\kappa}(\xi) \geq \dim F$, thus yielding, in contrast to Example 1 above, that for balanced closed[2] Σ_2-definable ω-languages F as well as for balanced ω-languages $F \in \mathfrak{S}$ there is indeed a $\xi \in F$ such that $\underline{\kappa}(\xi) = \underline{\kappa}(F) = \dim F$.

Concluding Remark

Our Theorems 2, 3 and 9 in connection with previous results of Ryabko [Ry86,93,94] and this author [St93] gave evidence that there is a strong coincidence between the concepts of Kolmogorov complexity (in Complexity theory) and Hausdorff dimension (in Fractal geometry) for a class of recursive (computable) ω-languages. The results of this paper show a borderline in the Arithmetical hierarchy up to which this coincidence holds true, and our examples give evidence that it does not extend much further.

[2]This property, however, does not hold in general for non-closed balanced Σ_2-definable ω-languages, e.g. it does not hold for $E' := \bigcup_{n\in\mathbb{N}}(1 \cdot X^n)^\omega$.

References

[Br82] A. A. Brudno, Entropy and complexity of trajectories of dynamic systems. *Trudy Mosk. Mat. Obshchestva* **44** (1982), 124 – 149 (in Russian; English translation: *Trans. Mosc. Math. Soc.* **44** (1983) 127 – 151).

[CH89] J.-Y. Cai and J. Hartmanis, The Kolmogorov complexity of the real line is a fractal, in *"Proc. 4th IEEE Structure in Complexity Conference,"*, 138 – 146, IEEE Computer Science Press, Washington DC, 1989.

[Ca94] C. Calude, *Information and Randomness. An Algorithmic Perspective.* Springer-Verlag, Berlin, 1994.

[Ch87] G. J. Chaitin, *Information, Randomness, & Incompleteness. Papers on Algorithmic Information Theory.* World Scientific, Singapore, 1987.

[Fa90] K.J. Falconer, *Fractal Geometry.* Wiley, Chichester, 1990.

[LV93] M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications.* Springer–Verlag, New York, 1993.

[Ry86] B. Ya. Ryabko, Noiseless coding of combinatorial sources, Hausdorff dimension, and Kolmogorov complexity. *Problemy Peredachi Informatsii* **22** (1986), No. 3, 16–26 (in Russian; English translation: *Problems of Information Transmission* **22** (1986), No. 3, 170–179).

[Ry93] B. Ya. Ryabko, An algorithmic approach to prediction problems. *Problemy Peredachi Informatsii* **29** (1993), No. 2, 96–103 (in Russian).

[Ry94] B. Ya. Ryabko, The complexity and effectiveness of prediction algorithms. *J. of Complexity* **10** (1994), 281–295.

[St81] L. Staiger, Complexity and entropy, in J. Gruska and M. Chytil, (Eds.) *Mathematical Foundations of Computer*

Science, Lecture Notes in Comput. Sci. **118**, 508 – 514, Springer–Verlag, Berlin, 1981.

[St86] L. Staiger, Hierarchies of recursive ω-languages. *J. Inform. Process. Cybernet. EIK*, **22** (1986), No. 5/6, 219 – 241.

[St89] L. Staiger, Combinatorial properties of the Hausdorff dimension. *J. Statist. Plann. Inference* **23** (1989), No. 1, 95 – 100.

[St93] L. Staiger, Kolmogorov complexity and Hausdorff dimension, *Inform. and Comput.* **102** (1993), No. 2, 159 – 194.

Automaten mit ein bißchen Parallelität

Thomas Worsch

Lehrst. Informatik f. Ingenieure u. Naturwissenschaftler
Universität Karlsruhe
Am Fasanengarten 5, D–76128 Karlsruhe

worsch@ira.uka.de

Van Emde Boas (1985) hat die Begriffe der ersten und zweiten Maschinenklasse eingeführt. Sie liefern eine grobe „Unterteilung" in sequentielle und parallele Maschinenmodelle. Diese Formulierung ist mit einer gewissen Vorsicht zu genießen, denn ein Nachweis, daß die beiden Maschinenklassen tatsächlich disjunkt sind, implizierte auch, daß $\mathrm{P} \neq \mathrm{PSPACE}$ ist (siehe z.B. Wiedermann (1995)).

Nichtsdestotrotz gibt es Überlegungen, deren Ziel Maschinenmodelle sind, die weder in der ersten noch in der zweiten Maschinenklasse liegen, sondern „dazwischen". D.h. man ist auf der Suche nach Modellen, für die man aufgrund gewisser bewiesener Zusammenhänge mit sequentiellen und parallelen Modellen geneigt ist, anzunehmen, daß sie in keiner der beiden Maschinenklassen liegen.

Ein Vorschlag stammt von Wiedermann (1992) und betrifft die von ihm eingeführten *parallelen Turingmaschinen* in der Variante mit mehreren Köpfen pro Steuereinheit. Daß diese nicht zur ersten Maschinenklasse gehören, wird dadurch gezeigt, daß sie von einer sequentiellen Turingmaschine nicht mit linearem Overhead in der Raumkomplexität simuliert werden können. Dies liegt unseres Erachtens daran, daß die Definition der Raumkomplexität paralleler Turingmaschinen „ungeeignet“ ist. Sie berücksichtigt nämlich nicht die Anzahl der in einer Konfiguration vorhandenen Steuereinheiten, sondern nur den auf den Arbeitsbändern benötigten Platz. Zählt man dagegen die Steuereinheiten bei der Raumkomplexität mit, so sind parallele Turingmaschinen (auch mit mehreren Köpfen) in der ersten Maschinenklasse (siehe Worsch (1996)).

Ein weiterer Kandidat waren die *iterativen Baumautomaten* von Čulik und Yu (1984). Die Autoren konnten zeigen, daß dieses Modell in der Lage ist, die Np-vollständige Sprache Sat in Linearzeit zu erkennen. Dies konnte man einerseits als Hinweis darauf deuten, daß das Modell „nicht mehr“ in der ersten Maschinenklasse ist, andererseits ist doch noch eine „gewisse Lücke“ zu Pspace bzw. der zweiten Maschinenklasse vorhanden.

Für ein anderes baumartiges Modell bewiesen Mycielski und Niwiński (1991), daß es in der Lage ist, die Pspace-vollständige Sprache Qbf in Polynomialzeit zu erkennen. Es ist nun eine recht leichte Aufgabe, die Techniken aus beiden Arbeiten zu kombinieren, um einzusehen, daß iterative Baumautomaten Qbf sogar in Linearzeit erkennen können. Insbesondere ist das Modell der iterativen Baumautomaten also in der zweiten Maschinenklasse. Dies wird auch bestätigt durch ein Ergebnis von Yu (1995), das er mir wenige Tage nach dem Theorietag mitteilte. Demzufolge sind iterative Baumautomaten und alternierende Turingmaschinen linearzeit-äquivalent.

Wir haben daher begonnen, eine Verallgemeinerung von Zellularautomaten zu untersuchen, bei der die Nachbarschaft jeder Zelle räumlich oder/und zeitlich variieren darf. Solange der maximale Durchmesser durch eine Konstante beschränkt ist, liegt das Modell — bei vernünftigen Konstruierbarkeitsforderungen an die Nachbarschaften — in der ersten Maschinenklasse. Dürfen die Nachbarschaften mit der Eingabewortlänge wachsen, ist dies nicht mehr klar. Erste Ergebnisse lassen es nicht ausgeschlossen erscheinen, daß dies ein erstes Modell ist, von dem man in gewissem Sinne annehmen kann, daß es zwischen den beiden Maschinenklassen liegt (siehe auch Ruf (1996)).

Literatur

ČULIK, K. und YU, S. (1984): *Iterative Tree Automata*, Theoretical Computer Science **32**, 227–247.

MYCIELSKI, J. und NIWIŃSKI, D. (1991): *Cellular Automata On Trees, A Model For Parallel Computation*, Fundamenta Informaticae **XV**, 139–144.

RUF, J. (1996): *Untersuchungen von erweiterten Zellularautomaten mit variabler Nachbarschaft*, Diplomarbeit, Universität Karlsruhe.

VAN EMDE BOAS, P. (1985): The second machine class: models of parallelism, in: Lenstra, J.K., von Leeuwen, J.(eds.): *Parallel Computers and Computations*, 133–161.

WIEDERMANN, J. (1992): *Weak Parallel Machines: a New Class of Physically Feasible Parallel Machine Models*, Mathematical Foundations of Computer Science, Proc. of the 17th Symposium, LNCS 629, 95–111.

WIEDERMANN, J. (1995): *Quo Vadetis, Parallel Machine Models?*, in: van Leeuwen, J. (ed.): Computer Science Today, LNCS 1000, 101–114.

WORSCH, TH. (1996): *On Parallel Turing Machines with several heads*, in Vorbereitung.

YU, S. (1995): persönliche Mitteilung.

Mitgliederversammlung der GI-Fachgruppe 0.1.5 „Automaten und Formale Sprachen“

Am 28. September 1995 fand während des 5. Theorietages „Automaten und Formale Sprachen“ in Schloß Rauischholzhausen eine Mitgliederversammlung der GI-Fachgruppe 0.1.5 statt. Anwesend waren die folgenden Fachgruppen-Mitglieder:

Oliver Boldt (Magdeburg), Henning Bordihn (Magdeburg), Thomas Buchholz (Gießen), Gerhard Buntrock (Würzburg), David Damanik (Frankfurt/Main), Kai Differt (Offenbach), Manfred Droste (Dresden), Henning Fernau (Tübingen), Rudolf Freund (Wien), H. Peter Gumm (Marburg), Norbert Gutleben (Cottbus), Markus Holzer (Tübingen), Martin Hühne (Dortmund), Klaus P. Jantke (Leipzig), Matthias Jantzen (Hamburg), Daniel Kirsten (Leipzig), Henner Kröger (Gießen), Armin Kühnemann (Dresden), Martin Kutrib (Gießen), Klaus-Jörn Lange (Tübingen), Friedrich Otto (Kassel), Bernd Reichel (Magdeburg), Klaus Reinhardt (Tübingen), Rüdiger Reischuk (Lübeck), Jörg Richstein (Gießen), Arnd Rußmann (Frankfurt/Main), Andreas Schrader (Siegen), Helmut Seidl (Trier), Ludwig Staiger (Halle/Saale), Ralf Stiebe (Magdeburg), Wolfgang Thomas (Kiel), Klaus Wich (Maintal-Bischhofsheim), Renate Winter (Halle/Saale), Thomas Worsch (Karlsruhe).

Um 16:50 Uhr eröffnet Herr Thomas als stellvertretender Fachgruppensprecher die Versammlung. (Der Fachgruppensprecher, Herr Dassow, ist wegen Krankheit verhindert.)

Es wird folgende Tagesordnung vorgeschlagen:

1. Genehmigung der Tagesordnung
2a. Bericht über die Arbeit der Fachgruppenleitung
2b. Bericht aus dem GI-Fachausschuß „Theoretische Informatik"
3. Neuwahl/Bestätigung der Fachgruppenleitung
4. Theorietage 1996, 1997
5. Verschiedenes

TOP 1: Gegen die vorgeschlagene Tagesordnung bestehen keine Einwände. Sie wird somit angenommen.

TOP 2a: Herr Thomas berichtet stellvertretend für Herrn Dassow über Aktivitäten seit Oktober 1994:

i. Teilnahme des Sprechers an der Sitzung des Fachausschusses am 2. März 1995. Auf dieser Sitzung wurde das Mitglied der Fachgruppe, Prof. Dr. Matthias Jantzen, zum Sprecher des Fachausschusses gewählt.
ii. Der von der Fachgruppenleitung erarbeitete Vorschlag für ein Mitglied der Fachgruppe im Programmkomitee von STACS 1996 wurde realisiert. Prof. Dr. Klaus-Jörn Lange ist Mitglied des Programmkomitees.
Auf einen Vorschlag für einen einzuladenden Vortragenden wurde verzichtet, da 1995 durch Prof. Dr. Wolfgang Thomas die Fachgruppe einen Hauptvortrag hatte.
iii. Einrichtung einer Informationsseite für die Fachgruppe im Internet. Sie ist zu finden unter http://www.informatik.uni-kiel.de/inf/Thomas/GI-015/index.html. Auf diese Information wird auch durch die Gesellschaft für Informatik (http://www.gi-ev.de) verwiesen.
iv. Zuarbeit über Ziele, Aufgaben, Veranstaltungen und Leitungsgremien der Fachgruppe für eine zentrale Datei der Gesellschaft für Informatik.

Die Versammlung nimmt zustimmend Kenntnis. Zur Führung der WWW-Seite der Fachgruppe wird angeregt, Verweise auf vorhandene Homepages von Mitgliedern einzufügen. Herr Thomas sagt dies zu.

TOP 2b: Der Sprecher des Fachausschusses, Herr Jantzen, berichtet aus dem Fachausschuß „Theoretische Informatik". Unter anderem über WWW-Seiten des Fachausschusses, die zentral bei der GI gehalten werden, zur GI-Jahrestagung und zur Weiterentwicklung von STACS (der nächste deutsche Veranstaltungsort nach Lübeck 1997 ist noch offen).

TOP 3: Herr Thomas informiert die Anwesenden über die derzeitige Zusammensetzung der Fachgruppenleitung: Jürgen Dassow (Sprecher), Manfred Droste, Annegret Habel, Friedrich Otto, Wolfgang Thomas (Stellvertretender Sprecher).

Er stellt die Alternativen Neuwahl oder Bestätigung der Fachgruppenleitung für ein weiteres Jahr (aber nicht länger) zur Diskussion. Von den Mitgliedern der Fachgruppenleitung liegt bis auf Annegret Habel, die noch nicht befragt werden konnte, das Einverständnis, ihr Amt für ein weiteres Jahr zu bekleiden, vor. Nach mehreren Wortbeiträgen wird über folgenden Antrag abgestimmt:

Bestätigung der Fachgruppenleitung. Falls Frau Habel für das Amt nicht mehr zur Verfügung stehen sollte, rückt Ludwig Staiger aufgrund seiner Stimmenzahl bei der letzten Wahl der Fachgruppenleitung in dieses Gremium nach. Der Antrag wird mit 26 Ja-Stimmen und 2 Enthaltungen ohne Nein-Stimmen angenommen.

TOP 4: Herr Droste, der sich bereits auf dem 4. Theorietag mit der Durchführung des Theorietages 1996 einverstanden erklärt hat, schlägt als Tagungsort ein Hotel in Cunnersdorf bei Königsstein in der Sächsischen Schweiz vor. Als mögliches Datum wird vorbehaltlich der Kollision mit anderen Tagungen der 19./20. September 1996 genannt. Herr Droste beabsichtigt, den Tag vor dem eigentlichen Theorietag für eine Reihe eingeladener Vorträge zu nutzen (die von Dresden aus finanziert würden). Die Versammlung dankt Herrn Droste und nimmt den Vorschlag per Akklamation an. Die Ausrichtung des darauffolgenden Theorietags ist noch offen; Angebote sind willkommen.

TOP 5: Herr Lange schlägt vor, die Vortragslängen bei künftigen Theorietagen flexibler zu gestalten. Nach Diskussion wird einvernehmlich festgehalten, zukünftig den Zeitbedarf angemeldeter Vorträge zu erfragen und im Rahmen der Möglichkeiten zu berücksichtigen.

Herr Thomas dankt im Namen aller Anwesenden Herrn Kutrib und Herrn Worsch für die perfekte Ausrichtung des 5. Theorietages und schließt die Versammlung um 17:30 Uhr.

Zusatzinformation (Stand 12. Januar 1996):

zu TOP 3: Frau Annegret Habel ist mit ihrer Bestätigung als Mitglied der Fachgruppenleitung für ein weiteres Jahr einverstanden.

zu TOP 4: Der Termin für den 6. Theorietag wurde auf den 19./20. September festgesetzt. Angebote für die Ausrichtung des 7. Theorietags (1997) liegen noch nicht vor; Hinweise dazu sollten an Herrn Dassow gerichtet werden.

Teilnehmerverzeichnis

- **Oliver Boldt**
  Fakultät für Informatik
  Otto-von-Guericke-Universität Magdeburg
  Postfach 41 20
  39016 Magdeburg

  Tel.: 0391/671-2851
  Email: boldt@fma2.math.uni-magdeburg.de

- **Henning Bordihn**
  Fakultät für Informatik
  Otto-von-Guericke-Universität Magdeburg
  Postfach 41 20
  39016 Magdeburg

  Tel.: 0391/671-2851
  Fax: 0391/67-12810
  Email: bordihn@cs.uni-magdeburg.de
  WWW: http://irb.cs.uni-magdeburg.de/theo/bordihn.html

- **Thomas Buchholz**
  AG Informatik
  Universität Gießen
  Arndtstr. 2
  35392 Gießen

  Tel.: 06003/1289
  Fax: 0641/702-2547
  Email: buchholz@informatik.uni-giessen.de
  WWW: http://www.informatik.uni-giessen.de/staff/buchholz/

- **Gerhard Buntrock**
  Lehrstuhl für Theoretische Informatik
  Institut für Informatik
  Universität Würzburg
  Am Exerzierplatz 3
  97072 Würzburg

  Tel.: 0931/79621.12
  Fax: 0931/79621.20
  Email: bunt@informatik.uni-wuerzburg.de
  WWW: http://haegar.informatik.uni-wuerzburg.de/person/mitarbeiter/bunt

- **David Damanik**
  FB Mathematik, AG 8.1
  Johann Wolfgang Goethe Universität
  Robert-Mayer-Str. 10
  60054 Frankfurt/Main

  Tel.: 069/79823752
  Email: damanik@math.uni-frankfurt.de

- **Kai Differt**
  Marienstr. 124
  63069 Offenbach

  Tel.: 069/835438
  Email: differt@psc.informatik.uni-frankfurt.de

- **Manfred Droste**
  Institut für Algebra
  TU Dresden
  01062 Dresden

  Tel.: 0351/463-3908
  Fax: 0351/463-4235
  Email: droste@math.tu-dresden.de

- **Henning Fernau**
  Universtität Tübingen
  Wilhelm-Schickard-Institut für Informatik
  Sand 13
  72076 Tübingen

  Tel.: 07071/29-7569
  Fax: 07071/68142
  Email: fernau@informatik.uni-tuebingen.de
  WWW: http://wsiserv.informatik.uni-tuebingen.de/~holzer/mitarbeiter/fernau.html

- **Rudolf Freund**
  Technische Universität Wien
  Resselg. 3
  A-1040 Wien

  Tel.: 00431588014084
  Fax: 004315041589
  Email: freund@csdec1.tuwien.ac.at
  WWW: http://logic.tuwien.ac.at/staff/rudi/home.html

- **H. Peter Gumm**
  Universität Marburg
  Fachbereich Mathematik
  Hans-Meerwein Str.
  35032 Marburg

  Tel.: 06421/285447
  Fax: 06421/285419
  Email: gumm@mathematik.uni-marburg.de
  WWW: http://www.mathematik.uni-marburg.de/~gumm

- **Norbert Gutleben**
  Theoretische Informatik
  BTU Cottbus
  Postfach 101 344
  03013 Cottbus

  Tel.: 0355/692187
  Fax: 0355/692236
  Email: gutleben@informatik.tu-cottbus.de

- **Markus Holzer**
  Universtität Tübingen
  Wilhelm-Schickard-Institut für Informatik
  Sand 13
  72076 Tübingen

  Tel.: 07071/29 7568
  Fax: 07071/ 68142
  Email: holzer@informatik.uni-tuebingen.de
  WWW: http://wsiserv.informatik.uni-tuebingen.de/~holzer/mitarbeiter/holzer.html

- **Martin Hühne**
  Lehrstuhl II
  Fachbereich Informatik
  Universität Dortmund
  44221 Dortmund

  Tel.: 0231/755-4808
  Fax: 0231/755-2047
  Email: huehne@ls2.informatik.uni-dortmund.de
  WWW: http://ls2-www.informatik.uni-dortmund.de/~huehne

- **Klaus P. Jantke**
  Hochschule für Technik, Wirtschaft
  und Kultur Leipzig
  Fachbereich IMN
  Postfach 30066
  04251 Leipzig

  Tel.: 0341/350-6426
  Fax: 0341/350-6426
  Email: jantke@informatik.th-leipzig.de
  WWW: http://www.informatik.th-leipzig.de

- **Matthias Jantzen**
  FB-Informatik
  Universität Hamburg
  Vogt-Kölln-Str. 30
  22527 Hamburg

  Tel.: 040/54715-409
  Fax: 040/54715-246
  Email: jantzen@informatik.uni-hamburg.de

- **Daniel Kirsten**
  Hochschule für Technik, Wirtschaft
  und Kultur Leipzig
  Fachbereich IMN
  Postfach 30066
  04251 Leipzig

  Email: kirsten@imn.th-leipzig.de

- **Henner Kröger**
  AG Informatik
  Universität Gießen
  Arndtstr. 2
  35392 Gießen

  Tel.: 0641/702-2547
  Fax: 0641/702-2547
  Email: kroeger@informatik.uni-giessen.de
  WWW: http://www.informatik.uni-giessen.de/staff/kroeger.html

- **Armin Kühnemann**
  Institut für Softwaretechnik
  Fakultät Informatik
  TU Dresden
  01062 Dresden

  Tel.: 0351/4575374
  Fax: 0351/4575504
  Email: kuehne@orchid.inf.tu-dresden.de

- **Martin Kutrib**
  AG Informatik
  Universität Gießen
  Arndtstr. 2
  35392 Gießen

  Tel.: 0641/702-2540
  Fax: 0641/702-2547
  Email: kutrib@informatik.uni-giessen.de
  WWW: http://www.informatik.uni-giessen.de/staff/kutrib.html

- **Klaus-Jörn Lange**
  Universtität Tübingen
  Wilhelm-Schickard-Institut für Informatik
  Sand 13
  72076 Tübingen

  Tel.: 07071/29-7567
  Fax: 07071/68142
  Email: lange@informatik.uni-tuebingen.de
  WWW: http://wsiserv.informatik.uni-tuebingen.de/~lange

- **Friedrich Otto**
  Fachbereich Mathematik/Informatik
  Universität-GH Kassel
  34109 Kassel

  Tel.: 0561/804-4573
  Fax: 0561/804-4008
  Email: otto@theory.informatik.uni-kassel.de

- **Bernd Reichel**
  Fakultät für Informatik
  Otto-von-Guericke-Universität Magdeburg
  Postfach 41 20
  39016 Magdeburg

  Tel.: 0391/67-12851
  Fax: 0391/67-12810
  Email: reichel@cs.uni-magdeburg.de
  WWW: http://irb.cs.uni-magdeburg.de/theo/reichel.html

- **Klaus Reinhardt**
  Universtität Tübingen
  Wilhelm-Schickard-Institut für Informatik
  Sand 13
  72076 Tübingen

  Tel.: 07071/29-7569
  Fax: 07071/68142
  Email: reinhard@informatik.uni-tuebingen.de
  WWW: http://wsiserv.informatik.uni-tuebingen.de/~holzer/mitarbeiter/reinhardt.html

- **Rüdiger Reischuk**
  Institut für Theoretische Informatik
  Universität Lübeck
  Wallstraße 40
  23560 Lübeck

  Tel.: 0451/7030-416
  Fax: 0451/7030-438
  Email: reischuk@informatik.mu-luebeck.de
  WWW: http://www.informatik.mu-luebeck.de

- **Jörg Richstein**
  AG Informatik
  Universität Gießen
  Arndtstr. 2
  35392 Gießen

  Tel.: 0641/702-2540
  Fax: 0641/702-2547
  Email: richstein@informatik.uni-giessen.de

- **Arnd Rußmann**
  FB 20 (Informatik)
  Johann Wolfgang Goethe Universität
  Postfach 11 19 32
  60054 Frankfurt/Main

  Tel.: 069/798-28419
  Email: russmann@psc.informatik.uni-frankfurt.de
  WWW: http://jaguar.psc.informatik.uni-frankfurt.de:8000/~russmann

- **Andreas Schrader**
  Universität-GH Siegen
  FB 12 - Elektrotechnik und Informatik
  Fachgruppe Programmiersprachen
  Hölderlinstr. 3
  57068 Siegen

  Tel.: 0271/7402314
  Fax: 0271/7402532
  Email: schrader@informatik.uni-siegen.de
  WWW: http://www.informatik.uni-siegen.de/~schrader/index.html

- **Helmut Seidl**
  FB IV - Abteilung Informatik
  Universität Trier
  54286 Trier

  Tel.: 0651/201-2835
  Fax: 0651/201-3822
  Email: seidl@ti.uni-trier.de
  WWW: http://www.informatik.uni-trier.de/~seidl/

- **Ludwig Staiger**
  Martin-Luther-Universität Halle-Wittenberg
  Institut für Informatik
  Weinbergweg 17
  06120 Halle (Saale)

  Tel.: 0345/5524714
  Fax: 0345/5527009
  Email: staiger@informatik.uni-halle.de
  WWW: http://www.informatik.uni-halle.de/∼staiger/

- **Ralf Stiebe**
  Fakultät für Informatik
  Otto-von-Guericke-Universität Magdeburg
  Postfach 41 20
  39016 Magdeburg

  Tel.: 0391/671-2851
  Fax: 0391/671-2810
  Email: stiebe@cs.uni-magdeburg.de
  WWW: http://irb.cs.uni-magdeburg.de/theo/stiebe.html

- **Wolfgang Thomas**
  Institut für Informatik
  und Praktische Mathematik
  Universität Kiel
  Olshausenstr. 40
  24098 Kiel

  Tel.: 0431/880-4480
  Fax: 0431/880-4054
  Email: wt@jupiter.informatik.uni-kiel.d400.de
  WWW: http://www.informatik.uni-kiel.de/∼wt/

- **Klaus Wich**
  Spessartstr. 3
  63477 Maintal-Bischhofsheim

  Tel.: 06109/68608
  Email: wich@informatik.uni-frankfurt.de

- **Renate Winter**
  Martin-Luther-Universität Halle
  Institut für Informatik
  Weinbergweg 17
  06120 Halle/Saale

  Tel.: 0345/5524 719
  Email: winter@informatik.uni-halle.de
  WWW: http://www.informatik.uni-halle.de/~winter/

- **Thomas Worsch**
  Lehrstuhl Informatik für Ingenieure
  und Naturwissenschaftler
  Universität Karlsruhe
  Am Fasanengarten 5
  76128 Karlsruhe

  Tel.: 0721/608-4311
  Fax: 0721/698675
  Email: worsch@ira.uka.de
  WWW: http://www.ira.uka.de/I3V_HTML/MITARBEITER/75