INSTITUT FÜR INFORMATIK
UND PRAKTISCHE MATHEMATIK

2. Theorietag

"Automaten und Formale Sprachen"

Universität Kiel

2./3. Oktober 1992

Bericht Nr. 9220

RISTIAN-ALBRECHTS-UNIVERSITÄT
KIEL

2. Theorietag

"Automaten und Formale Sprachen"

Universität Kiel

2./3. Oktober 1992

Herausgeber:

Wolfgang Thomas
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
Olshausenstraße 40, DW-2300 Kiel 1

Inhalt

Vorwort 3

Vortragsprogramm 5

Zusammenfassungen der Vorträge 7

GI-Fachgruppe "Automaten und Formale Sprachen"

Bericht über die Gründungsversammlung 61

Fachgruppenordnung 63

Adressen der Teilnehmer 67

Vorwort

Mit dem 2. Theorietag "Automaten und Formale Sprachen" hat sich die GI-Fachgruppe gleichen Namens offiziell konstituiert. Dieses Heft enthält neben den Zusammenfassungen der gehaltenen Vorträge auch einen Bericht über die Gründungsversammlung der Fachgruppe sowie die verabschiedete Fachgruppenordnung.

Die rege Teilnahme am Theorietag (ziemlich flächendeckend über die deutschen Informatik-Standorte) und die Zahl der Vorträge belegen das große Interesse an der Thematik "Automaten und Formale Sprachen". Anregungen zur zukünftigen Arbeit der Fachgruppe sind am Schluß des Berichts über die Gründungsversammlung angesprochen.

An dieser Stelle sei allen Mitgliedern der Kieler Arbeitsgruppe "Theoretische Informatik" für ihren Einsatz vor und während der Tagung herzlich gedankt. Ohne ihre organisatorische Hilfe, Herrichtung von Kaffee und Imbiß, Fahrdienste, Unterstützung am Rechner und weiteres mehr wäre das Treffen nicht möglich gewesen. Auch der Universität Kiel möchte ich für die gewährte finanzielle Unterstützung danken.

Wolfgang Thomas

2. Theorietag "Automaten und Formale Sprachen"

Christian-Albrechts-Universität zu Kiel, 2./3.10.1992

Vortragsfolge

Freitag, 2.10.1992

J. Hromkovic: Synchronisierte Alternierung (Hauptvortrag) 7

M. Middendorf: Supersequenzen und das Wortproblem für CD Grammatik-Systeme . . . 8
S. Skalla: Zur Komplexität von Grammatiksystemen 12
C. Hümpel: Darstellungssätze für Teilklassen der RI-Sprachen 16
V. Diekert: Deterministische asynchrone Automaten für unendliche Spuren 20
A. Muscholl: Logische Definierbarkeit über unendlichen Spuren 21
S. Gärtner: Partitions-limitierte Lindenmayersysteme 23
M. Bull: Springende Mehrkopfautomaten mit Zusatzspeichern in Labyrinthen 28
G. Buntrock: Abschlußeigenschaften von wachsend kontextsensitiven Sprachen 32

Gründungsversammlung der Fachgruppe

Samstag, 3.10.1992

F. Otto: Über einige Entscheidungsprobleme für Untergruppen von kontextfreien Gruppen (Hauptvortrag) 33

D. Caucal: Bisimulation of pushdown automata 36
R. Freund: Graphgrammatiken mit Mengenproduktionen 37
H. Jürgensen: Automatentheoretische Überlegungen zum Testen von Schaltungen 41
L. Staiger: Fraktale Dimension konstruktiv erzeugter Bilder 45
H. Fernau: MRFS-Fraktale aus dem Blickwinkel der regulären ω-Sprachen 46
H. Seidl: Wann ist eine funktionale Baumübersetzung deterministisch? 51
M. Holzer: Die Komplexität linearer LL(1) und LR(1) Sprachen 52
A. Brüggemann-Klein: Unambiguity of SGML content models 53
T. Schwentick: Formale Sprachen von Output Strings: Eine allgemeine Methode zur Definition von Komplexitätsklassen 55
M. Korff: Neuere Entwicklungen aus dem Bereich algebraischer Graphgrammatiken . 56

Synchronisierte Alternierung

J. Hromkovic

Universität-Gesamthochschule Paderborn
FB 17: Mathematik-Informatik
Postfach 1621, W-4790 Paderborn

Die Synchronisierte Alternierung ist eine Verallgemeinerung von Alternierung. So wie Nichtdeterminismus und Alternierung als „unrealistische" Berechnungsmodelle beim Studium von deterministischen (also realistischen) Berechnungen hilfreich waren, so liefert die Synchronisierte Alternierung einige neue Aussichten für deterministische und nichtdeterministische Berechnungen. Die synchronisierten alternierenden endlichen Automaten akzeptieren zum Beispiel genau NSPACE(n) – die kontextsensitiven Sprachen, obwohl die üblichen alternierenden endlichen Automaten nur reguläre Sprachen akzeptieren. Bei einer anderen Art von Synchronisation akzeptieren die endlichen Automaten genau P. Die Frage, ob DSPACE (f(n)) echte Teilmenge von NSPACE (f(n)) ist [also nach Determinismus versus Nichtdeterminismus] kann gestellt werden als Frage über Synchronisation - ob eine Art von Synchronisation stärker als eine andere Art von Synchronisation für ein gegebenes Modell von Turing-Maschinen ist. Mehrere ähnliche Resultate, die neue Beschreibungen von fundamentalen Komplexitätsklassen ermöglichen, werden präsentiert.

Supersequenzen und das Wortproblem für CD-Grammatik-Systeme
(Kurzfassung)

Martin Middendorf
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
Universität Karlsruhe
Postfach 6980
W-7500 Karlsruhe

Zusammenfassung. Das *Minimum Run Supersequence* Problem (MinRS) ist das Problem, zu entscheiden, ob es für eine gegebene Menge von Worten über einem Alphabet Σ und eine natürliche Zahl k eine Supersequenz von L gibt, die höchstens k viele Runs besitzt. Es wird gezeigt, daß MinRS über einem Alphabet der Größe 2 in polynomieller Zeit lösbar ist und über jedem Alphabet der Größe 3 NP-vollständig ist. Darüberhinaus wird die algorithmische Komplexität einiger verwandter Probleme betrachtet. Diese betreffen Supersequenzen mit einer maximalen Anzahl von Runs und Supersequenzen, in die die gegebenen Worte disjunkt eingebettet werden können. Es wird anschließend gezeigt, daß Supersequenz Probleme ein geeignetes Mittel sind, um die Komplexität des Wortproblems für Grammatik-Systeme zu untersuchen. Dazu werden beispielhaft CD-Grammatik-Systeme mit zwei Komponenten und einem alternierenden Ableitungsmodus betrachet.

1. Supersequenzprobleme

Definition 1.1: Ein Wort S ist eine *Supersequenz eines Wortes* T, wenn man T durch Streichen von Null oder mehr Symbolen aus S erhält. Ein Wort S ist eine *Supersequenz einer Menge* L *von Worten*, wenn S Supersequenz eines jeden Wortes in L ist.

> **Problem 1: Shortest Common Supersequence (SCS)**
> Gegeben: Eine Menge L von Worten und $k \in \mathbb{N}$.
> Frage: Gibt es eine Supersequenz von L der Länge $\leq k$?

Satz 1.2 (Räihä, Ukkonen): *Shortest Common Supersequence ist NP-vollständig über jedem Alphabet der Größe* ≥ 2.

Weitere Resultate zum SCS Problem findet man in [4], [5], [6] und [9].

Definition 1.3: Es sei S ein Wort über einem Alphabet Σ. Ist S von der Form $S'TS''$, wobei S', T und S'' Worte über Σ sind, dann heißt T *Teilwort von* S. Ein maximal langes Teilwort T von S mit $T \in a^+$ für ein $a \in \Sigma$ heißt *Run von* S.

> **Problem 2: Minimum Run Supersequence (MinRS)**
> Gegeben: Eine Menge L von Worten und $k \in \mathbb{N}$.
> Frage: Gibt es eine Supersequenz von L die $\leq k$ Runs enthält?

Satz 1.4: i) *Minimum Run Supersequence ist in P über einem Alphabet der Größe 2.*
ii) *Minimum Run Supersequence ist NP-vollständig über jedem Alphabet der Größe* ≥ 3.

Definition 1.5: Es sei $S = s_1 s_2 \ldots s_k$ eine Supersequenz eines Wortes $T = t_1 t_2 \ldots t_l$. Eine *Einbettung von* T *in* S ist eine streng wachsende Funktion f von $[1:l]$ nach $[1:k]$, so daß t_i und $s_{f(i)}$ der gleiche Buchstabe sind für alle $i \in [1:l]$. Eine *Einbettung einer Menge* $L = \{S_1, S_2, \ldots S_k\}$ *von Worten in ein Wort* S ist ein k-Tupel $(f_1, f_2, \ldots, f_k)$, wobei f_i eine Einbettung von S_i in S ist. Eine Einbettung von L in S heißt *disjunkt*, wenn für alle $i, j \in [1:k], i \neq j$ und $l_i \in [1:|S_i|]$, $l_j \in [1:|S_j|]$ gilt: $f_i(l_i) \neq f_j(l_j)$, d.h. keine zwei verschiedenen Buchstaben von Worten in L werden auf denselben Buchstaben von S abgebildet. Eine Supersequenz einer Menge L von Wortens heißt *d-Supersequenz*, wenn es eine disjunkte Einbettung von L in S gibt.

> **Problem 3: Disjoint Minimum Run Supersequence (DMinRS)**
> Gegeben: Eine Menge L von Worten und $k \in \mathbb{N}$.
> Frage: Gibt es eine d-Supersequenz von L die $\leq k$ Runs enthält?

Korollar 1.6: i) *Disjoint Minimum Run Supersequence ist in P über einem Alphabet der Größe 2.*
ii) *Disjoint Minimum Run Supersequence ist NP-vollständig über jedem Alphabet der Größe* ≥ 3.

Definition 1.7: Eine (d-) Supersequenz S einer Menge L von Worten heißt *minimal*, wenn kein Symbol in S gestrichen werden kann, ohne daß die Eigenschaft (d-) Supersequenz zu sein verlorengeht.

> **Problem 4: Maximum Run Supersequence (MaxRS)**
> Gegeben: Eine Menge L von Worten und $k \in \mathbb{N}$.
> Frage: Gibt es eine minimale Supersequenz von L die $\geq k$ Runs enthält?

Satz 1.8 *Maximum Run Supersequence ist NP-vollständig über jedem Alphabet der Größe* ≥ 2.

> **Problem 5: Disjoint Maximum Run Supersequence (DMaxRS)**
> Gegeben: Eine Menge L von Worten und $k \in \mathbb{N}$.
> Frage: Gibt es eine minimale d-Supersequenz von L die $\geq k$ Runs enthält?

> **Problem 6: Disjoint Runlength 1 Supersequence (DRL1S)**
> Gegeben: Eine Menge L von Worten und $k \in \mathbb{N}$.
> Frage: Gibt es eine minimale d-Supersequenz S von L, so daß jeder Run in S die Länge 1 hat?

Satz 1.9: *Disjoint Runlength 1 Supersequence ist NP-vollständig über jedem Alphabet der Größe* ≥ 2.

Korollar 1.10: *Disjoint Maximum Run Supersequence ist NP-vollständig über jedem Alphabet der Größe* ≥ 2.

2. Das Wortproblem für CD-Grammatik-Systeme

Es wird nun gezeigt, daß Supersequenzprobleme ein geeignetes Mittel darstellen, um die Komplexität des Wortproblems für Grammatik-Systeme zu untersuchen. Zum Beispiel läßt sich durch Reduktion des SCS Problems ein relativ einfacher Beweis für die bekannte Tatsache (vgl. [10]) angeben, daß das Wortproblem für ETOL-Systeme NP-vollständig ist. Etwas näher wird im folgenden das Wortproblem für CD-Grammatik-Systeme mit zwei Produktionenmengen untersucht. CD-Grammatik-Systeme sind von Csuhaj-Varju und Dassow eingeführt worden ([1]).

Definition 2.1: Ein *CD-Grammatik-System* ist ein $(2+k)$-Tupel $G = (V, T, P_1, P_2, \ldots, P_k, B)$. Dabei sind

(i) V und T disjunkte endliche nichtleere Alphabete — das *Variablenalphabet* bzw. das *Terminalenalphabet*;

(ii) P_i, $i \in [1:k]$ ist eine endliche Menge von Paaren (*Produktionen*) der Form (A, α) mit $A \in V \cup T$, $\alpha \in (V \cup T)^*$, so daß gilt

a) $A \in V$ für alle Produktionen $(A, \alpha) \in P_i$ — P_i heißt dann *kontextfreie Produktionenmenge* — oder

b) für jedes $A \in V \cup T$ existiert eine Produktion (A, α) in P_i — P_i heißt dann *EOL-Produktionenmenge*;

(A, α) wird auch mit $A \to \alpha$ bezeichnet;

(iii) $B \in V$ ist die *Startvariable*.

Definition 2.2: Unter einem Ableitungsschritt eines CD-Grammatik-Systems $G = (V, T, P_1, P_2, \ldots, P_k, B)$ bezüglich einer kontextfreien (bzw. EOL-) Produktionenmenge P_i versteht man einen Ableitungschritt bezüglich der kontextfreien Grammatik (bzw. des EOL-Systems) (V, T, P_i, B). Ist $G = (V, T, P_1, P_2, B)$ ein CD-Grammatik-System mit zwei Produktionenmengen, dann versteht man unter einer alternierenden Ableitung eine Wortes $w \in T^*$ eine Ableitung, bei der abwechselnd jeweils genau ein Ableitungsschritt bezüglich P_1 und P_2 gemacht wird.

Definition 2.3: Es sei $G = (V, T, P_1, P_2, B)$ ein CD-Grammatik-System mit zwei Produktionenmengen.

$$L_A(G) = \{w \mid w \in T^* \text{ und es gibt eine alternierende Ableitung von } w \text{ aus } B\}.$$

$L_A(G)$ heißt *die von G durch alternierende Ableitung erzeugte Sprache.*

> Es sei $G = \{V, T, P_1, P_2, B\}$ ein CD-Grammatik-System mit zwei Produktionenmengen.
>
> **Problem 7: Wortproblem für $L_A(G)$**
>
> Gegeben: Ein Wort $w \in T^*$.
>
> Frage: Ist $w \in L_A(G)$?

Die folgenden Sätze 2.4 und 2.5 zeigen, daß es bereits sehr einfache CD-Grammatik-Systeme gibt, für die das Wortproblem NP-vollständig ist.

Satz 2.4 *Es sei $G = \{V, T, P_1, P_2, B\}$ ein CD-Grammatik-System mit zwei kontextfreien Produktionenmengen wobei $V = \{B\}$, $T = \{0, 1, \$\}$ und*

$$P_1 = \{B \to B\$0B,\ B \to 0B,\ B \to 0\$B,\ B \to 0\}$$
$$P_2 = \{B \to B\$1B,\ B \to 1B,\ B \to 1\$B,\ B \to 1\}.$$

Das Wortproblem für $L_A(G)$ ist NP-vollständig.

Beweis (Idee). Reduziere DRL1S über dem Alphabet $\{0, 1\}$ auf unser Problem.

Es sei $L = \{S_1, S_2, \ldots, S_t\}$ eine Menge von Worten über dem Alphabet $\{0, 1\}$ eine Instanz von DRL1S. O.B.d.A. sei $|S_i| \geq 2$ für $i \in [1:t]$. Zeige

$S_1\$S_2\$\ldots\$S_t \in L_A(G)$ gdw.
eine minimale d-Supersequenz S von L existiert, so daß jeder Run in S die Länge 1 hat.

□

Satz 2.5: *Es sei $G = \{V, T, P_1, P_2, B\}$ ein CD-Grammatik-System mit einer EOL- und einer kontextfreien Produktionenmenge wobei $V = \{B, B'\}$, $T = \{0, 1, \$\}$ und*

$$P_1 = \{B \to B\$0B,\ B \to 0B,\ B \to 0\$B,\ B \to 0$$
$$B' \to B'\$1B',\ B' \to 1B',\ B \to 1\$B',\ B' \to 1\}$$
$$P_2 = \{B \to B',\ B' \to B,\ 0 \to 0,\ 1 \to 1,\ \$ \to \$\}.$$

Das Wortproblem für $L_A(G)$ ist NP-vollständig.

Satz 2.6: *Es sei $G = \{V, T, P_1, P_2, B\}$ ein CD-Grammatik-System mit zwei EOL-Produktionenmengen. Das Wortproblem für $L_A(G)$ ist in P.*

Literatur

[1] E. Csuhaj-Varju und J. Dassow, On cooperating/distributed grammar systems, *J. Inf Process. Cybern. EIK* **26** (1990) 49-63.

[2] M.R. Garey und D.S. Johnson, *Computers and Intractability* (W.H. Freeman, San Francisco, 1979).

[3] T.Gonzalez und S. Sahni, Flowshop and jobshop schedules: complexity and approximation, *Op. Res.* **26** (1978) 36-52.

[4] D. Maier, The complexity of some problems on subsequences and supersequences, *J. ACM* **25** (1978) 322-336.

[5] M. Middendorf, Zur Komplexität von Einbettungsproblemen für Wortmengen, Dissertation, Institut für Informatik, Universität Hannover, Februar 1992.

[6] M. Middendorf, More on the complexity of superstring and supersequence problems, erscheint in *Theoret. Comput. Sci.*.

[7] J. Opatrný und K. Culik II, Time complexity of recognition and parsing of EOL-languages, in: A. Lindenmayer und G. Rozenberg (eds.), *Automata, languages and development*, (North-Holland, Amsterdam, 1976).

[8] K.-J. Räihä und E. Ukkonen, The shortest common supersequence problem over binary alphabet is NP-complete, *Theoret. Comput. Sci.* **16** (1981) 187-198.

[9] V.G. Timkovskii, Complexity of common subsequence problems and related problems, *Cybernetics* **25** (1990) 565-580; übersetzt aus *Kibernetica* **25** (1989) 1-13.

[10] J. van Leeuwen, The Membership problem for ETOL-languages is polynomially complete, *Inform. Process. Lett.* **3** (1975) 138-143.

Zur Komplexität von Grammatiksystemen

Jürgen DASSOW, Stephan SKALLA
Technische Universität Magdeburg, Fakultät für Mathematik
PSF 4120, O-3010 Magdeburg

Gheorghe PĂUN[1]
Institut für Mathematik der Rumänischen Akademie der Wissenschaften
Str. Academiei 14, R-70109 Bucuresti, Rumänien

1. Einleitung und Definitionen

Tafelarchitekturen sind ein bekanntes Modell für Problemlösungsprozesse mit verteilten Ressourcen. Sie bestehen aus mehreren Wissensbasen, die nach einem Protokoll gemeinsam an der Lösung eines Problems arbeiten, wobei die Kommunikation nur über die Tafel erfolgt, auf der jeweils der aktuelle Stand der Lösung angegeben ist. In jedem Moment ist nur eine Wissensbasis aktiv, indem sie den Tafelinhalt ändert; dann wird sie inaktiv und eine weitere Wissensbasis wird aktiviert. Der Prozeß wird solange fortgesetzt, bis eine akzeptale Lösung erreicht ist. Für Einzelheiten verweisen wir auf [7].

Wir geben nun ein formales grammatikalisches Modell für Tafelarchitekturen, das in [1] eingeführt wurde.

Definition 1. Ein kooperierendes/verteiltes (cooperating/distributed, abgekürzt CD) Grammatiksystem des Grades n, $n \geq 1$, ist ein $(n+3)$-Tupel

$$\Gamma = (N, T, P_1, P_2, \ldots, P_n, S),$$

wobei N die Menge der Nichtterminale, T die Menge der Terminale und P_i für $1 \leq i \leq n$ Mengen kontext-freier Produktionen sind.

Dabei modellieren die Regelmengen die Wissensbasen, die aktuelle Satzform entspricht dem aktuellen Tafelinhalt, die Regeln ermöglichen eine Änderung des Tafelinhalts.
Wir bezeichnen die einzelnen Regelmengen als Komponenten des Systems und setzen $V_G = N \cup T$.

Definition 2. Sei Γ ein CD Grammatiksystem wie in Definition 1. Eine k-Schritt-Ableitung, $k \geq 1$, in einer Komponente $P \in \{P_1, P_2, \ldots P_n\}$, wird mit $\Longrightarrow_P^{=k}$ bezeichnet. Analog bezeichnen $\Longrightarrow_P^{\leq k}$ bzw. $\Longrightarrow_P^{\geq k}$ Ableitungen mit höchstens k bzw. mindestens k Schritten. Weiterhin bezeichnet $\Longrightarrow_P^{*}$ eine beliebige Ableitung. Falls $x \Longrightarrow_P^{*} y$ gilt und kein Wort $z \in V_\Gamma^*$ mit $z \neq y$ und $y \Longrightarrow_P^{*} z$ existiert, so schreiben wir $x \Longrightarrow_P^{t} y$.

Diese Ableitungsarten entsprechen den folgenden Strategien bei Tafelarchitekturen: $\Longrightarrow^{\leq k}$ modelliert eine Zeitbeschränkung für die Arbeit an der Tafel; $\Longrightarrow^{\geq k}$ stellt eine Forderung nach einer Mindestkompetenz dar, da mindestens ein vorgeschriebener Beitrag zur Lösung geleistet werden muß; $\Longrightarrow^{*}$ erlaubt jeder Wissensbasis eine Arbeit eine Arbeit an der Tafel,

[1] Durch die Alexander-von-Humboldt-Stiftung geförderte Forschung

solange es von ihr gewünscht wird; $\Longrightarrow^{t}$ entspricht der Forderung, daß eine Komponente solange arbeitet, wie sie zur Lösung beitragen kann.
Wir setzen

$$M = \{*, t\} \cup \bigcup_{k \geq 1} \{\leq k, k, \geq k\}.$$

Definition 3. Die vom CD Grammatiksystem Γ in der Ableitungsart $f \in M$ erzeugte Sprache $L_f(\Gamma)$ ist durch

$$\begin{aligned} L_f(\Gamma) \;=\; & \{x \in T^* \mid S = x_0 \Longrightarrow_{P_{i_1}}^{f} x_1 \Longrightarrow_{P_{i_2}}^{f} x_2 \Longrightarrow \ldots \Longrightarrow_{P_{i_m}}^{f} x_m = x, \\ & m \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq m\} \end{aligned}$$

definiert.

Aufgrund der Motivation sind die beiden folgenden Parameter eines CD Grammatiksystems von besonderem Interesse: die Anzahl der Komponenten und die Größe der Komponenten, die wir durch die Anzahl der darin enthaltenen Regeln messen, da die Anzahl der Regeln als ein Maß für die Fähigkeit der Komponente zu Änderungen des Tafelinhalts betrachtet werden kann. Formal führt dies zu der folgenden Definition.

Definition 4. Für ein CD Grammatiksystem Γ wie in Definition 1 eine Sprache L und eine Ableitungsart $f \in M$ setzen wir

$$\begin{aligned} Deg(\Gamma) &= n, \\ Deg_f(L) &= min\{Deg(\Gamma) : L = L_f(\Gamma)\}, \\ Prod(\Gamma) &= max\{card(P_i) \mid 1 \leq i \leq n\}, \\ Prod_f(L) &= min\{Prod(\Gamma) : L = L_f(\Gamma)\}. \end{aligned}$$

Weiterhin definieren wir für $n \geq 1, m \geq 1$ (einschließlich $n = \infty$, $m = \infty$) $CD_{n,m}CF$ als die Menge aller CD Grammatiksysteme Γ with $Deg(\Gamma) \leq n$ und $Prod(\Gamma) \leq m$. Für eine Menge X von CD Grammatiksystemen und eine Ableitungsart $f \in M$ bezeichnen wir mit

$$\mathcal{L}_f(X) = \{L_f(\Gamma) \mid \Gamma \in X\}$$

die zugehörige Familie von Sprachen.

Offenbar gelten für natürliche Zahlen $n \geq 1$ und $m \geq 1$ und $f \in M$ die beiden folgenden Relationen

$$\mathcal{L}_f(CD_{n,m}CF) \subseteq \mathcal{L}_f(CD_{n+1,m}CF) \text{ und } \mathcal{L}_f(CD_{n,m}CF) \subseteq \mathcal{L}_f(CD_{n,m+1}CF).$$

Ziel dieser Arbeit sind die hieraus resultierenden Hierarchien, d.h. die Untersuchung der Frage, ob die Inklusionen echt sind.

Mit $\mathcal{L}(CF)$ bzw. $\mathcal{L}(CF_{fin})$ bzw. $\mathcal{L}(ET0L)$ bezeichnen wir die Familien der kontext-freien Sprachen bzw. der kontext-freien Sprachen mit endlichem Index bzw. der ETOL Sprachen (siehe [5], [9], [8]).

2. Resultate

Satz 1. *Seien $n \geq 1$ und $m \geq 1$ natürliche Zahlen und*

$$f \in \{*, = 1, \geq 1\} \cup \{\leq k : k \geq 1\}.$$

Dann gelten:
a) $\mathcal{L}_f(CD_{n,\infty}CF) = \mathcal{L}_f(CD_{\infty,\infty}CF) = \mathcal{L}(CF)$,
b) $\mathcal{L}_f(CD_{\infty,m}CF) = \mathcal{L}_f(CD_{\infty,\infty}CF) = \mathcal{L}(CF)$,
c) $\mathcal{L}_f(CD_{n,m}CF) \subset \mathcal{L}_f(CD_{n+1,m}CF)$ *und* $\mathcal{L}_f(CD_{n,m}CF) \subset \mathcal{L}_f(CD_{n,m+1}CF)$.

Satz 1 c) ist eine Verallgemeinerung des Resultates von Gruska, daß es zu jeder natürlichen Zahl $n \geq 1$ eine kontext-freie Sprache existiert, die zu ihrer Erzeugung mindestens n Nichtterminale erfordert, da kontext-freie Grammatiken mit Grammatiksystemen mit einer Komponente übereinstimmen (siehe [3], [4]).
In [6] wurde eine Verallgemeinerung von Satz 1 a) auf den Fall gegeben, daß die verschiedenen Komponenten auch mit verschiedenen Ableitungsarten arbeiten können.

Satz 2. *Seien $n \geq 3, n' \geq 1, m \geq 5$ und $m' \geq 1$ natürliche Zahlen. Dann gelten:*
a) $\mathcal{L}(CF) = \mathcal{L}_t(CD_{1,\infty}CF) \subset \mathcal{L}_t(CD_{2,\infty}CF) \subseteq \mathcal{L}_t(CD_{3,\infty}CF) = \mathcal{L}_t(CD_{\infty,\infty}CF) = \mathcal{L}(ETOL)$,
b) $\mathcal{L}(CF) = \mathcal{L}_t(CD_{\infty,1}CF) \subset \mathcal{L}_t(CD_{\infty,2}CF) \subseteq \mathcal{L}_t(CD_{\infty,3}CF) \subseteq \mathcal{L}_t(CD_{\infty,4}CF) \subseteq \mathcal{L}_t(CD_{\infty,m}CF) = \mathcal{L}_t(CD_{\infty,\infty}CF) = \mathcal{L}(ETOL)$,
c) $\mathcal{L}_f(CD_{n,m}CF) \subset \mathcal{L}_f(CD_{n+1,m}CF)$ *und* $\mathcal{L}_f(CD_{n,m}CF) \subset \mathcal{L}_f(CD_{n,m+1}CF)$.

Es ist ein offenes Problem, ob die in Satz 2 a) bzw. b) gegebenen Schranken 3 bzw. 5 minimal sind.

Satz 3. *Seien $k \geq 1, n \geq 1, m \geq k$ und $m' \geq 1$ natürliche Zahlen. Dann gelten:*
a) $\mathcal{L}_{=k}(CD_{\infty,k}CF) = \mathcal{L}_{=k}(CD_{\infty,m}CF) = \mathcal{L}_{=k}(CD_{\infty,\infty}CF)$,
b) $\mathcal{L}_{=k}(CD_{n,2}CF) \subset \mathcal{L}_{=k}(CD_{n+1,2}CF)$ *und* $\mathcal{L}_{=k}(CD_{2,m'}CF) \subset \mathcal{L}_{=k}(CD_{2,m'+1}CF)$.

Satz 4. *Seien $k \geq 1, n \geq 1, m \geq 2k-1$ und $m' \geq 1$ natürliche Zahlen. Dann gelten:*
a) $\mathcal{L}_{\geq k}(CD_{\infty,k}CF) = \mathcal{L}_{\geq k}(CD_{\infty,m}CF) = \mathcal{L}_{\geq k}(CD_{\infty,\infty}CF)$,
b) $\mathcal{L}_{\geq k}(CD_{n,2}CF) \subset \mathcal{L}_{\geq k}(CD_{n+1,2}CF)$ *und* $\mathcal{L}_{\geq k}(CD_{2,m'}CF) \subset \mathcal{L}_{\geq k}(CD_{2,m'+1}CF)$.

Es ist uns bisher nicht gelungen, für die Ableitungsarten $= k$ und $\geq k$ ein zu den Aussagen a) in den Sätzen 1 und 2 analoges Resultat zu gewinnen, und die Unendlichkeit der Hierarchien für den Fall, daß beide Parameter beschränkt sind, wurde bisher nur für den Fall gezeigt, daß einer der Parameter 2 ist.

Abschließend geben wir ein Unentscheidbarkeitsresultat.

Satz 5. *Die folgenden Probleme sind für eine gegebene natürliche Zahl $n \geq 1$ und ein gegebenes Grammatiksystem Γ unentscheidbar:*
a) Gilt $n = Deg(L_t(\Gamma))$ *?*
b) Gilt $n = Prod(L_t(\Gamma))$ *?*
c) Gilt $Deg(\Gamma) = Deg(L_t(\Gamma))$ *?*
d) Gilt $Prod(\Gamma) = Prod(L_t(\Gamma))$ *?*

Literatur

[1] E. Csuhaj-Varju, J. Dassow, On cooperating/distributed grammar systems. *J. Inf. Process. Cybern. EIK* 26 (1990), 49 - 63.

[2] J. Dassow, Gh. Păun, On the succinctness of descriptions of context-free languages by cooperating/distributed grammar systems. *Computers and Artificial Intelligence* 10 (1991), 513 - 527.

[3] J. Gruska, Some classifications of context-free languages. *Inform. Control* 14 (1969), 152 - 179.

[4] J. Gruska, Descriptional complexity of context-free languages. *Proc. MFCS '73 Symp.* (1973), 71 - 84.

[5] J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, Mass., 1979.

[6] V. Mitrana, Hybrid cooperating/distributed grammar systems. *Computers and Artificial Intelligence*, 1992.

[7] P. H. Nii, Blackboard systems. In *The Handbook of AI*, vol. 4 (A. Barr, P. R. Cohen, E. A. Feigenbaum, Eds.), Addison-Wesley, Reading, Mass., 1989.

[8] G. Rozenberg, A. Salomaa, *The Mathematical Theory of L Systems.* Academic Press, New York, 1980.

[9] A. Salomaa, *Formal Languages.* Academic Press, New York, London, 1973.

Darstellungssätze für Teilklassen von RI-Sprachen

Claudia Hümpel
Institut für Informatik
Universität Hannover

Indizierte Sprachen und Teilklassen von indizierten Sprachen sind seit ihrer Einführung von Aho [1] intensiv untersucht worden.
In diesem Vortrag werden einige Teilklassen der durch die *restricted indexed grammars* (im folgenden RI-Grammatiken genannt) definierten Sprachen (= RI-Sprachen), die ebenfalls von Aho in [1] eingeführt wurden, vorgestellt.
Nach einigen notwendigen Definitionen werden einige Sätze formuliert, mit deren Hilfe es möglich ist, Inklusionen und Nichtvergleichbarkeit der eingeführten Sprachklassen zu klären.

Zunächst folgt eine zur Definition von Aho [1] äquivalente

Definition : Eine Grammatik $G = (N_u, N_d, T, I, P, S)$ mit N_u, N_d, T und I paarweise disjunkte endliche Mengen, wobei
N_u die Menge der *flagaufbauenden Variablen*,
N_d die Menge der *flagabbauenden Variablen*,
T die Menge der *terminalen Zeichen*,
I die Menge der *Indizes* oder *Flags* ,
$S \in N_u$ *das Startsymbol* und
P eine endliche Teilmenge von $((N_u \cup N_d) \times (I \cup \{\epsilon\})) \times ((N_u \cup N_d)I^* \cup T)^*$ die *Menge der Produktionen* sind,
heißt *RI-Grammatik*, wenn jede Produktion aus P eine der Formen $A \longrightarrow \alpha$ mit $A \in N_u$, $\alpha \in ((N_u \cup N_d)I^* \cup T)^*$ oder $Bf \longrightarrow \beta$ mit $B \in N_d, f \in I \cup \{\epsilon\}$, $\beta \in (N_d \cup T)^*$ besitzt.

Die Begriffe *Ableitung* und *RI-Sprache* werden wie bei indizierten Sprachen üblich definiert. Die aus der Definition resultierende Sprachklasse wird mit $\mathcal{L}(RI)$, die Familie der indizierten Sprachen mit $\mathcal{L}(IND)$ bezeichnet.

Bemerkungen:

1) Es gilt offensichtlich $\mathcal{L}(RI) \subseteq \mathcal{L}(IND)$. Van Leeuwen hat in [3] gezeigt, daß $\mathcal{L}(RI) \subset \mathcal{L}(IND)$ gilt.
2) Betrachtet man einen Ableitungsbaum zu einem Wort $w \in L$, L RI-Sprache, so kann man jeden Weg von der Wurzel zu einem Blatt in eine *Flagaufbauphase* und in eine *Flagabbauphase* partitionieren, d.h. (vgl. Definition) vor dem "Umschalten" auf eine Variable $B \in N_d$ werden keine Flags abgebaut und nach dem Umschalten werden keine neuen Flags mehr erzeugt.

Nun werden wir die Produktionen, die man in der Flagaufbau- und in der Flagabbauphase anwenden kann, einschränken und die daraus resultierenden Sprachklassen untersuchen.

(g) gerade Linie:
$A \longrightarrow Bf$ oder $A \longrightarrow u$ bzw. $Cf \longrightarrow D$ oder $Cf \longrightarrow u$

(rl) rechtslinear indiziert:
$A \longrightarrow uBf$ oder $A \longrightarrow u$ bzw. $Cf \longrightarrow uD$ oder $Cf \longrightarrow u$

(ll) linkslinear indiziert:
$A \longrightarrow Bfu$ oder $A \longrightarrow u$ bzw. $Cf \longrightarrow Du$ oder $Cf \longrightarrow u$

(l) linear indiziert:
$A \longrightarrow uBfv$ oder $A \longrightarrow u$ bzw. $Cf \longrightarrow uDv$ oder $Cf \longrightarrow u$

(k) kontextfrei (bzw. indiziert):
$A \longrightarrow \alpha$ bzw. $Cf \longrightarrow \beta$

Dabei ist $A \in N_u$, $B \in N_u \cup N_d$, $C, D \in N_d$, $f \in I \cup \{\epsilon\}$, $\alpha \in ((N_d \cup N_u)I^* \cup T)^*$, $\beta \in (N_d \cup T)^*$ und $u, v \in T^*$.

In diesem Vortrag betrachten wir nur den linear indizierten Fall, d.h. Produktionen der Form (rl), (ll) und (l) und sämtliche Kombinationen für die Flagaufbau- und Flagabbauphase.

Bezeichnungen:

1) Mit $\mathcal{L}(x\text{-}y)$, $x, y \in \{rl, ll, l\}$, wird die Sprachklasse bezeichnet, die man erhält, wenn die flagaufbauenden Produktionen die Form x und die flagabbauenden Produktionen die Form y haben.
2) Mit $\mathcal{L}(LIN)$ werden die linear kontextfreien und mit $\mathcal{L}(lIND)$ die linear indizierten Sprachen (vgl. [2]) bezeichnet.

Satz 1:

1) $\mathcal{L}(ll\text{-}ll) = \mathcal{L}(rl\text{-}rl) = \mathcal{L}(LIN)$.
2) $\mathcal{L}(rl\text{-}ll) = \mathcal{L}(ll\text{-}rl)$.
3) $\mathcal{L}(rl\text{-}l) = \mathcal{L}(ll\text{-}l)$.

Satz 2:

Sei $L = XcY$ *mit* $X, Y \subseteq T^*$, $c \notin T$.

1) Ist $L \in \mathcal{L}(rl\text{-}ll)$, *so sind* X *oder* Y *regulär.*
2) Ist $L \in \mathcal{L}(rl\text{-}l)$, *so ist* X *regulär, oder* Y *ist regulär, oder* X *und* Y *sind beide linear kontextfrei.*
3) Ist $L \in \mathcal{L}(l\text{-}rl)$, *so ist* Y *regulär, oder* X *ist linear kontextfrei.*
4) Ist $L \in \mathcal{L}(l\text{-}ll)$, *so ist* X *regulär, oder* Y *ist linear kontextfrei.*
5) Ist $L \in \mathcal{L}(l\text{-}l)$, *so sind* X *oder* Y *linear kontextfrei.*

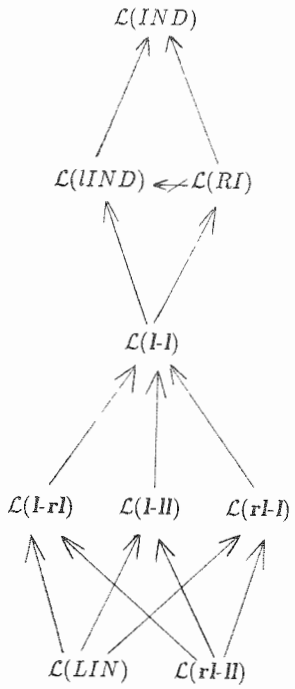
Folgerung 1:

$\mathcal{L}(l\text{-}rl)$ und $\mathcal{L}(l\text{-}ll)$ sind nicht abgeschlossen gegenüber Wortumkehr.

Satz 3:

1) *$L \subseteq T^*$ ist aus $\mathcal{L}$(rl-ll), gdw. es ein Alphabet $\hat{T}$, eine reguläre Menge $M \subseteq \hat{T}^*$ und zwei Homomorphismen $h_i : \hat{T} \longrightarrow T^*$, $i \in [1:2]$, gibt mit $L = \{h_1(w)h_2(w) \mid w \in M\}$.*
2) *$L \subseteq T^*$ ist aus $\mathcal{L}$(rl-l), gdw. es ein Alphabet $\hat{T}$, eine reguläre Menge $M \subseteq \hat{T}^*$ und drei Homomorphismen $h_i : \hat{T} \longrightarrow T^*$, $i \in [1:3]$, gibt mit $L = \{h_1(w)h_2(w)^R h_3(w) \mid w \in M\}$.*
3) *$L \subseteq T^*$ ist aus $\mathcal{L}$(l-rl), gdw. es ein Alphabet $\hat{T}$, eine reguläre Menge $M \subseteq \hat{T}^*$ und drei Homomorphismen $h_i : \hat{T} \longrightarrow T^*$, $i \in [1:3]$, gibt mit $L = \{h_1(w)h_2(w)^R h_3(w)^R \mid w \in M\}$.*
4) *$L \subseteq T^*$ ist aus $\mathcal{L}$(l-ll), gdw. es ein Alphabet $\hat{T}$, eine reguläre Menge $M \subseteq \hat{T}^*$ und drei Homomorphismen $h_i : \hat{T} \longrightarrow T^*$, $i \in [1:3]$, gibt mit $L = \{h_1(w)h_2(w)h_3(w)^R \mid w \in M\}$.*
5) *$L \subseteq T^*$ ist aus $\mathcal{L}$(l-l), gdw. es ein Alphabet $\hat{T}$, eine reguläre Menge $M \subseteq \hat{T}^*$ und vier Homomorphismen $h_i : \hat{T} \longrightarrow T^*$, $i \in [1:4]$, gibt mit $L = \{h_1(w)h_2(w)^R h_3(w)h_4(w)^R \mid w \in M\}$.*

Folgerung 2: (aus Satz 2 und Satz 3)



Bemerkungen:

1) In [2] werden für die linear indizierten Sprachen ähnliche Aussagen wie in den Sätzen 2 und 3 bewiesen.
2) Als weiteres interessantes Ergebnis ergibt sich $\mathcal{L}(l-k) = \mathcal{L}(ET0L)$.

Literaturverzeichnis:

[1] A. V. Aho, *Indexed languages — an extension of context-free grammars*, J. ACM **15** (1968), 647–671.

[2] J. Duske, R. Parchmann, *Linear indexed languages*, Theoret. Comput. Sci. **32** (1984), 47–60.

[3] J. van Leeuwen, *Variations of a new machine model*, Conf. Record 17th. Annual IEEE Symp. on Foundations of Computer Science, Houston, Texas, 1976, 228–235.

Deterministic asynchronous automata for infinite traces

A. Muscholl und V. Diekert

Universität Stuttgart, Institut für Informatik
Breitwiesenstr. 20-22, 7000 Stuttgart 80

The concept of traces forms a suitable semantics for concurrent systems. A concurrent system is given by a set of atomic actions Σ together with an independence relation $I \subseteq \Sigma \times \Sigma$ which specifies pairs of actions which can be performed concurrently. The relation I generates a congruence on Σ^*, which defines a quotient monoid of Σ^* called *trace monoid.*

In order to describe non-terminating concurrent processes, the notion of infinite trace was introduced. These infinite traces are now called *real traces.*

In this paper, we consider recognizable real trace languages. These languages play the central role since they correspond to infinite runs of concurrent processes which can be controled by some finite state advice. Various properties of this language class have been established recently, in particular the equivalence to c-rational expressions, and the characterization by monadic second logic, have been shown. However, no convenient automata for accepting recognizable real trace languages were known. As all recognizable languages they can be characterized by word automata, but these automata are inherently sequential and even the existence of word automata with the so-called I-diamond property was not known.

The first break-through is due to Gastin and Petit who introduced asynchronous Büchi automata for infinite traces and showed that every recognizable real trace language can be accepted by some nondeterministic asynchronous Büchi automaton. These asynchronous automata (first defined for finite traces by Zielonka) can describe concurrency in a very suitable way, due to the decentralized control which allows parallel execution of independent actions. The existence of deterministic asynchronous automata characterizing recognizable real trace languages remained open and will be solved here.

In the theory of infinitary word languages the theorem of McNaughton states the equivalence between nondeterministic Büchi automata and deterministic Muller automata. We provide here a proper generalization of McNaughton's Theorem to the family of recognizable real trace languages, thus solving one of the main open problems in this field:

Theorem: *The family of recognizable real trace languages is equivalent to the family of languages, which are accepted by deterministic Muller asynchronous cellular automata.*

This result is based on the notion of deterministic languages and the following result.

Logical Definability on Infinite Traces

W. Ebinger und A. Muscholl

Universität Stuttgart, Institut für Informatik
Breitwiesenstr. 20-22, 7000 Stuttgart 80

In order to extend the free partially commutative monoid $\mathbb{M}(\Sigma, D)$ over a dependence alphabet (Σ, D) to the infinite case, it is convenient to consider dependence graphs. In the monoid $\mathbb{G}(\Sigma, D)$ of infinite dependence graphs we consider the subset $\mathbb{R}(\Sigma, D)$ of real traces, where every node has a finite past. The notion of recognizability can be extended to real trace languages.

We show here the equivalence of monadic second order logic and recognizability for real trace languages and that first order definable, star-free, and aperiodic real trace languages form the same class of languages. This generalizes results on infinite words and on finite traces to infinite traces.

Thus we have completed the different characterizations for recognizable infinitary trace languages. This family of languages can be defined by means of recognizing morphisms and it is also equivalent to c-rational expressions. It has been also characterized from an automata theoretical point of view, using adequate Büchi and Muller acceptance conditions. Only the logical characterization of recognizability in terms of monadic second order remained open so far and will be provided here.

The proof of the main result, which is stated below, is independent of any special kind of trace automaton.

Theorem: *A trace language $A \subseteq \mathbb{R}(\Sigma, D)$ is recognizable if and only if A is definable in monadic second order logic.*

We transform formulae on words to formulae on traces using the lexicographic normal form and vice versa and obtain

Theorem: *A trace language $A \subseteq \mathbb{R}(\Sigma, D)$ is definable in first order logic if and only if $\varphi^{-1}(A)$ is definable in first order logic.*

Similar to the word case we show

Theorem: *A trace language $A \subseteq \mathbb{R}(\Sigma, D)$ is definable in first order logic if and only if it is star-free.*

Although the result below is a consequence of our logic results, we also give an independent proof.

Theorem: *The family of star-free subsets of $\mathbb{R}(\Sigma, D)$ coincides with the family of aperiodic real trace languages.*

Theorem: *The family of recognizable real trace languages is equivalent to the Boolean closure of the family of deterministic real trace languages.*

As a special case we obtain that every word language, which is closed w.r.t. the independence relation, can be accepted by some deterministic Muller automaton. We also determine the complexity of deciding whether a I-diamond Muller automaton accepts a closed language.

Theorem: *It is NL-complete to decide for a given a deterministic I-diamond Muller automaton, whether the accepted language is closed.*

TECHNISCHE UNIVERSITÄT CAROLO-WILHELMINA ZU BRAUNSCHWEIG
Institut für Theoretische Informatik

Stefan Gärtner

Partitions-limitierte Lindenmayersysteme

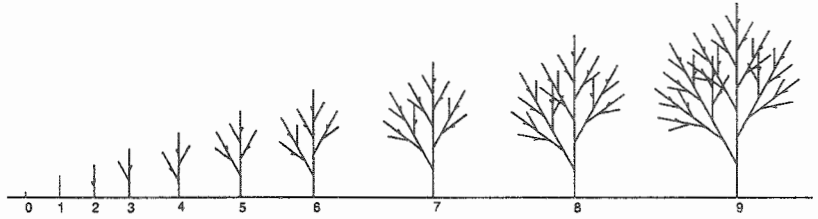
Lindenmayersysteme [Li68a, Li68b] wurden als Modell für parallele Zellentwicklung eingeführt und haben auch in der Mathematik und Informatik ein großes Echo gefunden. ([Her69],[vD71], [RD71],[RS80],etc.) Unlimitierte Lindenmayersysteme ersetzen in jedem Schritt jedes Zeichen. Dieser Umstand erschwert den Einsatz solcher Systeme als biologische Modelle, da dort das Wachstum stets durch äußere Faktoren beschränkt ist. Mit der von D. Wätjen [Wa88, Wa90] eingeführten k-Limitierung wurde eine solche Beschränkung geschaffen. Gleichzeitig konnten die Sprachfamilien der Lindenmayersysteme so erweitert werden. Die hier vorgestellten partitions-limitierten-Lindenmayersysteme basieren auf einer weiteren Limitierungsart, die zu einer nochmaligen Erweiterung der Sparchfamilien der Lindenmayersysteme führt.

Es werden erweiterte und nicht erweiterte partitions-limitierte Lindenmayersysteme eingeführt sowie einige Ergebnisse vorgestellt. Dazu gehört auch das Resultat, daß die Sprachfamilie der erweiterten partitions-limitierten Lindenmayersysteme mit Tafeln gleich der der rekursiv aufzählbaren Sprachen ist.

1 Lindenmayersysteme (0L-Systeme)

Beispiel 1 Es sei $B_1 = (\Sigma, h, a)$ ein D0L-System, mit $\Sigma = \{a, b, c, l, r, w, x, (,), +, -\}$ und h:

$$\begin{array}{lllll} a \to cwlb & l \to (+a) & w \to x & (\to (& + \to + \quad c \to c \\ b \to cwra & r \to (-b) & x \to wc &) \to) & - \to - \end{array}$$



Wir sehen die Turtle-Interpretation [AS82] der ersten 10 Wörter von $L(B_1)$ nach [SQ79].

2 k-limitierte Lindenmayersysteme (kl0L-Systeme)

Definition 1 Es sei $k \in \mathbb{N}$ und h eine endliche Substitution über einem Alphabet Σ. Die Abbildung $h_k : \Sigma^* \to 2^{\Sigma^*}$ heißt <u>k-limitiert</u>, wenn $h_k(w)$ für jedes $w \in \Sigma^*$ alle die Wörter liefert, die entstehen, wenn genau $\min\{\#_a w, k\}$ Zeichen von allen auftretenden Zeichen a in w ersetzt werden.

Beispiel 2 Es sei $\Sigma = \{a, b, c\}$ ein Alphabet und $k = 2$ so haben wir folgende Möglichkeiten Zeichen in dem Wort $aaabcc$ zu ersetzen. (Die unterstrichenen Zeichen sollen jeweils ersetzt werden.)

$$\underline{aa}abcc,\ \underline{aa}abcc,\ \underline{aa}abcc$$

Definition 2 Es sei $K' = (\Sigma, H, \omega, \Delta)$ ein ET0L-System und $k \in \mathbb{N}$, dann heißt das Fünftupel $K = (\Sigma, H, \omega, \Delta, k)$ <u>k-limitiertes ET0L-System</u> (klET0L-System). Die von K erzeugte Sprache ist

$$L(K) = \{w \in \Delta^* | w = \omega \text{ oder } w \in h_k^t \dots h_k^1(\omega) \text{ mit } h^1, \dots, h^t \in H, t \in \mathbb{N}\}.$$

Jede Sprache, die von einem klET0L-System erzeugt wird, heißt klET0L-Sprache. Für ein festes k bezeichnen wir mit $\mathcal{L}(k\text{lET0L})$ die Familie aller klET0L-Sprachen. Mit $\mathcal{L}(\text{l}_\text{k}\text{ET0L})$ bezeichnen wir die Familie aller Sprachen, die von klET0L-Systemen erzeugbar sind, also

$$\mathcal{L}(\text{l}_\text{k}\text{ET0L}) = \bigcup_{k \in \mathbb{N}} \mathcal{L}(k\text{lET0L}).$$

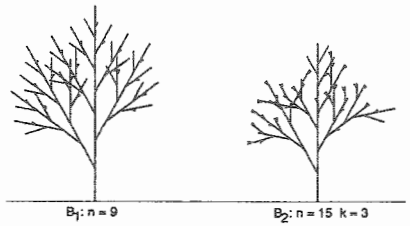
Beispiel 3 Für jedes $k \in \mathbb{N}$ definieren wir das kl0L-System

$$K_k = (\{a, b, c\}, h, abc, k)$$

mit $h(a) = \{a^2\}$, $h(b) = \{b^2\}$ und $h(c) = \{c^2\}$. Es sei $r = \lfloor \text{ld } k \rfloor + 1$, dann erhalten wir

$$L(K_k) = \{a^{2^i} b^{2^i} c^{2^i} \mid 0 \le i \le r\} \cup \{a^n b^n c^n \mid n = 2^r + ik, i \in \mathbb{N}\}.$$

Beispiel 4 Es seien $B_1 = (\Sigma, h, a)$ das D0L-System aus Beispiel 1 und $B_2 = (\Sigma, h, a, k)$ ein klD0L-System mit $k = 3$.



An diesem Beispiel wird deutlich, das k-limitierte Lindenmayersysteme eine Möglichkeit für ein biologisches Modell bilden. Eine Interpretation wäre z.B. die, daß Zellen auf einen begrenzten Nährstoffvorrat angewiesen sind und so in der Praxis nicht parallel wachsen können.

3 k-partitions-limitierte L-Systeme (kpl0L-Systeme)

Definition 3 Es sei Σ ein Alphabet. Wir nennen p eine <u>Partition von Σ</u>, wenn $p(\Sigma) = \{p_1, p_2, \dots, p_n\}$ mit $n \in \mathbb{N}$, $p_i \subset \Sigma$, $p_i \neq \emptyset$ und $\bigcup_{i=1}^{n} p_i = \Sigma$ und $p_i \cap p_j = \emptyset$ für alle i, j mit $1 \le i, j \le n$, $i \neq j$. Die p_i werden <u>Teile</u> der Partition genannt. $\mathcal{P}$ ist die Menge aller Partitionen beliebiger Alphabete.

Beispiel 5 Es sei $\Sigma = \{a, b, c\}$ ein Alphabet, dann gibt es folgende Partitionen:

$$p(\Sigma) = \{\{a, b, c\}\}, \quad p(\Sigma) = \{\{a, b\}, \{c\}\}, \quad p(\Sigma) = \{\{a\}, \{b, c\}\},$$
$$p(\Sigma) = \{\{a, c\}, \{b\}\}, \quad p(\Sigma) = \{\{a\}, \{b\}, \{c\}\}.$$

Definition 4 Es sei $k \in \mathbb{N}$, Σ ein endliches Alphabet, p eine Partition von Σ und h eine endliche Substitution über Σ. Die Abbildung $h_{kp} : \Sigma^* \to 2^{\Sigma^*}$ heißt <u>k-partitions-limitiert</u>, wenn $h_{kp}(w)$ für jedes $w \in \Sigma^*$ aus den Wörtern besteht, die man erhält, indem in w aus jedem Teil $p_i \in p(\Sigma)$ genau $\min\{\sum_{a \in p_i} \#_a w, k\}$ Zeichen ersetzt werden.

Beispiel 6 Es sei $\Sigma = \{a, b, c\}$ ein Alphabet, $k = 2$ und $p(\Sigma) = \{\{a, b\}, \{c\}\}$ eine Partition von Σ, dann sind folgende Ersetzungen möglich:

$$\underline{aa}ab|\underline{cc},\ \underline{aa}ab|\underline{cc},\ \underline{aa}ab|\underline{cc},\ aa\underline{ab}|\underline{cc},\ a\underline{aa}b|\underline{cc}, \dots$$

Definition 5 Es sei $K' = (\Sigma, H, \omega, \Delta)$ ein ET0L-System, p eine Partition von Σ und $k \in \mathbb{N}$. Dann heißt das Sechstupel $K = (\Sigma, H, \omega, \Delta, p, k)$ <u>k-partitions-limitiertes ET0L-System</u> (kplET0L-System). Die von K erzeugte Sprache ist

$$L(K) = \{w \in \Delta^* \mid w = \omega \text{ oder } w \in h_{kp}^t \dots h_{kp}^1(\omega) \text{ mit } h^1, \dots, h^t \in H, t \in \mathbb{N}\}.$$

Jede Sprache, die von einem kplET0L-System erzeugt wird, heißt kplET0L-Sprache. Für ein festes k und eine Partitionen p bezeichnen wir mit $\mathcal{L}(kp\text{lET0L})$ die Familie aller kplET0L-Sprachen. Weiter definieren wir

$$\mathcal{L}(k\text{l}_\text{p}\text{ET0L}) = \bigcup_{p \in \mathcal{P}} \mathcal{L}(kp\text{lET0L}), \quad \mathcal{L}(p\text{l}_\text{k}\text{ET0L}) = \bigcup_{k \in \mathbb{N}} \mathcal{L}(kp\text{lET0L}) \text{ und}$$

$$\mathcal{L}(\text{l}_\text{kp}\text{ET0L}) = \bigcup_{k \in \mathbb{N}, p \in \mathcal{P}} \mathcal{L}(kp\text{lET0L}).$$

Beispiel 7 Es sei $K_1 = (\Sigma, h, abbcdd, p, 1)$ ein 1pl0L-System mit $\Sigma = \{a, b, c, d\}$, $h(x) = \{x^2\}$ für alle $x \in \Sigma$ und $p(\Sigma) = \{\{a, c\}, \{b\}, \{d\}\}$.

$L_0(K_1)$	$L_1(K_1)$	$L_2(K_1)$	...
$abbcdd$	$aabbbcddd$	$aaabbbbcdddd$	...
	$abbbccddd$	$aabbbbccdddd$	...
		$abbbbcccdddd$	...

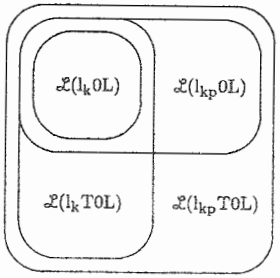
$$L(K_1) = \{a^i b^n c^j d^n \mid i + j = n,\ i, j \in \mathbb{N}\}$$

Satz 1 Es sei $K = (\Sigma, H, \omega, p, k)$ ein kplT0L-System und $n = |p(\Sigma)|$, $m = \max\{|w| \mid w \in h(a), h \in H, a \in \Sigma\}$, $w_1, w_2 \in \Sigma$ und $w_1 \underset{K}{\Longrightarrow} w_2$. Dann gilt

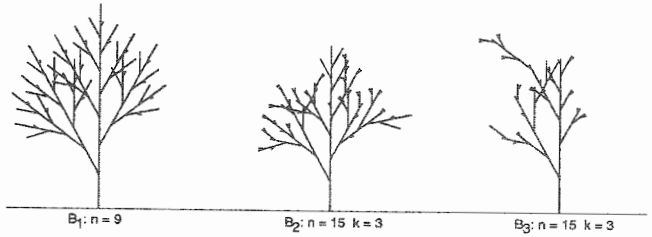
$$|w_1| - nk \leq |w_2| \leq |w_1| + nk(m-1).$$

Daraus folgt, daß die Sprache $L = \{a^{2^n} | n \in \mathbb{N}\}$ nicht von einem kplT0L-System erzeugbar ist.

Satz 2 Vergleich von kpl(T)0L-Sprachen mit kl(T)0L-Sprachen.



Beispiel 8 Es seien $B_1 = (\Sigma, h, a)$ das D0L-System aus Beispiel 1, $B_2 = (\Sigma, h, a, k)$ ein klD0L-System und $B_3 = (\Sigma, h, a, k, p)$ ein kplD0L-System mit $p = \{\{a, b\}, \{c\}, \{l\}, \{r\}, \{w\}, \{x\}, \{(\}, \{)\}, \{+\}, \{-\}\}$ und $k = 3$.



Betrachten wir den rechten Baum. Die Zeichen a und b erzeugen je einen linken bzw. rechten Teilbaum. Dadurch, daß sie in einem Teil der Partition liegen, können sie sich gegenseitig ausschließen. Auf diese Weise kann eine Wachstumsrichtung überwiegen. Eine mögliche biologische Interpretation ist, daß bestimmte Zelltypen auf einen gemeinsamen beschränkten Nährstoffvorrat angewiesen sind.

Satz 3 Für erweiterte partitions-limitierte Systeme mit Tafeln gilt:

$$\mathcal{L}(1l_p ET0L) = \mathcal{L}_0.$$

Beweisidee:

(a) Eine Sprache ist genau dann vom Typ 0, wenn sie von einer kontextfreien programierten Grammatik mit Vorkommenstest erzeugt wird.

(b) Wir konstruieren ein 1plET0L-System, das sich wie eine programmierte Grammatik mit Vorkommenstest verhält, dabei wird die Programmierung durch Tafelmechanismus und Kontrollzeichen und der Vorkommenstest durch spezielle Partitionen nachgebildet.

Literatur

[AS82] *Abelson, H. and A. A. diSessa*: Turtle geometry, MIT Press, Cambridge, 1982.

[An76] *Andrews, George E.*: The Theorie of Partitions, Addison Wesley, 1976.

[Her69] *G. T. Herman*: Computing Ability of a Developmental Model for Filamentous Organisms, Journal of Theoretical Biology, 25:421–435, 1969.

[Li68a] *Lindenmayer, Aristid*: Mathematical models for cellular interactions in development, Part I: Filaments with one-sided inputs, Journal of Theoretical Biology, 18:280–299, 1968.

[Li68b] *Lindenmayer, Aristid*: Mathematical models for cellular interactions in development, Part II: Simple and branching filaments with two-sided inputs, Journal of Theoretical Biology, 18:300–315, 1968.

[RD71] *G. Rozenberg and P. Doucet*: On 0L languages, Information and Control, 19:302–318, 1971.

[RS80] *Rozenberg, G., A. Salomaa*: The Mathematical Theory of L Systems, Academic Press, New York 1980.

[SQ79] *Szilard, A. L. and R. E. Quinton*: An interpretation for D0L systems by computer graphics, The Science Terrapin, 4:8–13, 1979

[vD71] *Dirk van Dalen*: A Note on Some Systems of Lindenmayer, Mathematical Systems Theory, 5:128–140, 1971.

[Wa88] *Wätjen, D.*: k-limited 0L Systems and Languages. J. Inf. Process. Cybern. EIK 24 (1988) 6, 267-285.

[Wa90] *Wätjen, D.*: On k-uniformly-limited 0L Systems and Languages J. Inf. Process. Cybern. EIK 26 (1990) 4, 229-238.

Springende Mehrkopfautomaten mit Zusatzspeichern in Labyrinthen

Mathias Bull*

1. Einleitung

Die in dieser Notiz angegebenen Resultate gehören zur Labyrinth-Theorie, in der die Kompliziertheit von Absuch- und Erkennungsprozessen auf nichtlinearen Objekten untersucht wird.

Motiviert durch das Ergebnis von W. Savitch [4] untersuchten A.C. Cook und C.W. Rackhoff in [2] endliche Automaten mit springenden Köpfen in Graphen. In [1] wurden für springende Mehrkopfzählerautomaten nicht absuchbare planare Labyrinthe, sogenannte Fallen konstruiert. Das Absuchverhalten von springenden Mehrkopfautomaten mit erweiterten Speichermöglichkeiten (Schieberegister, verschachtelte und vergleichbare Zähler) wird hier dargestellt

Für eine Einführung in die Labyrinth-Problematik sowie zur Einordnung der Ergebnisse wird auf [3] verwiesen.

2. Grundlegende Begriffe

Ein *Graphoid* ist ein Tupel $G=(V,H,I)$, wobei V und H nichtleere, disjunkte Mengen sind (*Knoten-* bzw *Halbkantenmenge*) und I eine irreflexive, symmetrische binäre Relation über $V \cup H$ ist (*Inzidenzrelation*), für welche die folgenden Bedingungen gelten:

1) $I \cap V^2 = \emptyset$;
2) zu jedem $h \in H$ existiert genau ein $v \in V$ mit $(h,v) \in I$;
3) zu jedem $h \in H$ existiert höchstens ein h' mit $(h,h') \in I$.

Eine Halbkante $h \in H$, für die es keine Halbkante $h' \in H$ mit $(h,h') \in I$ gibt, wird als *frei* und ein Graphoid ohne freie Halbkanten als ein *Graph* bezeichnet. Eine Menge $e=\{h,h'\} \subseteq H$ mit $(h,h') \in I$ heißt *Kante*. Für $v \in V$ bezeichne $H(v):=\{h:(h,v) \in I\}$.

Die aus der Graphentheorie bekannten Begriffe (wie z.B. Einbettung, Zusammenhang, Planarität, Regularität usw.) lassen sich auf Graphoide übertragen.

Im folgenden werden nur Graphoide betrachtet, die endliche Mengen $H(v)$ besitzen.

* Fachbereich Informatik, Universität Rostock, Albert-Einstein-Str.21, O-2500 Rostock

Sei D eine nichtleere, endliche Menge. Unter einem *Kompaßsystem* (mit der Richtungsmenge D) auf dem Graphoid G ist eine Abbildung $c:H \to D$ zu verstehen, deren Einschränkungen $c|_{H(v)}$ für alle $v \in V$ injektiv sind. Ein *Rotationssystem* ist eine Bijektion $r:H \to H$, so daß für jeden Knoten $v \in V$ die Einschränkung $r|_{H(v)}$ eine zyklische Permutation der Länge $\mathrm{card}(H(v))$ bildet.

Ein Tupel $L=(V,H,I,c)$ heißt *C-Graphoid*, falls (V,H,I) ein Graphoid ist und c ein Kompaßsystem auf (V,H,I) bildet. Ein kantengefärbtes Graphoid $L=(V,H,I,ec)$, kurz *EC-Graphoid*, ist ein C-Graphoid, so daß für alle Kanten $e=\{h,h'\}$ von L $ec(h)=ec(h')$ gilt. Seien $\mathbb{D}_2=\{\underline{\text{nord}},\underline{\text{süd}},\underline{\text{ost}},\underline{\text{west}}\}$ und $\mathbb{D}_3=\mathbb{D}_2 \cup \{\underline{\text{oben}},\underline{\text{unten}}\}$. Ein C-Graphoid, welches ein Kompaßsystem c mit der Richtungsmenge $\mathbb{D}_2$ besitzt, heißt *2D-Graphoid*, falls eine (2D-) Einbettung in die Ebene so existiert, daß die Halbkanten durch zu den Koordinatenachsen parallele, einander nicht schneidene Strecken repräsentiert werden und diese den Knoten in der entsprechenden Richtung verlassen. Analog sind die Begriffe *3D-Einbettung* und *3D-Graphoide* zu fassen.

Ein *R-Graphoid* $L=(V,H,I,r)$ ist ein mit einem Rotationssystem r versehenes Graphoid. Ein C- bzw. R-Graphoid heißt planar, falls das unterliegende Graphoid planar ist.

3. Mealyautomat mit Zusatzspeicher

Ein *Zusatzspeicher* $\mathcal{S}=(S,T,P,\tau,s_0)$ besteht aus einer Menge $S \neq \emptyset$ (Speicherzustände), einer endlichen Menge $T \neq \emptyset$ (Testmenge), einer endlichen Menge P mit $\emptyset \neq P \subseteq \{p:(p:S \to S)\}$ (Menge der Speicherinstruktionen), einer Abbildung $\tau:S \to T$ und eine Element $s_0 \in S$.

Ein Mealyautomat $\mathcal{A}=(X,Y,Z,\delta,\lambda,z_0)$ mit den nichtleeren, endlichen Menge X und Y (Eingabe- bzw Ausgabemenge), der Menge $Z \neq \emptyset$ (Zustandsmenge), den Abbildungen $\delta:Z \times X \to Z$ (Zustandsüberführungsfunktion), $\lambda:Z \times X \to Y$ (Ausgabefunktion) und $z_0 \in Z$ (Startzustand) heißt *Mealyautomat $\mathcal{A}$ mit dem Zusatzspeicher* $\mathcal{S}=(S,T,P,\tau,s_0)$, falls eine endliche Menge $Q \neq \emptyset$ sowie Abbildungen $\delta_P:Q \times T \times X \to P$, $\delta_Q:Q \times T \times X \to Q$ und $\lambda':Q \times T \times X \to Y$ existieren, so daß die folgenden Bedingungen erfüllt sind:

1. $Z=Q \times S$ und $z_0=(q_0,s_0)$;
2. für alle $q \in Q$, $s \in S$ und $x \in X$ gilt:
   $\delta((q,s),x)=(\delta_Q(q,\tau(s),x),p(s))$ mit $p=\delta_P(q,\tau(s),x)$
3. für alle $q \in Q, s \in S$ und $x \in X$ gilt: $\lambda((q,s),x)=\lambda'(q,\tau(s),x)$.

Zur Beschreibung der hier untersuchten Zusatzspeicher definieren wir folgende Speicherzustandsmengen:

Seien $C[1]:=\mathbb{N}_0\times\{0\}$ und $C[a+1]:=C[a]\cup(\mathbb{N}_0^{a+1}\times\{0,1,\ldots,a\})$ für $a\in\mathbb{N}_0$ mit $a\geq 1$.
Seien $C[a,b]:=\{w: (n_0,n_1,\ldots,n_i,j)=w\in C[a]$ und $j\leq b\}$, wobei $a,b\in\mathbb{N}_0$ und $a>b$, und $C[\infty]:=\bigcup\{C[a+1] : a\in\mathbb{N}_0\}$ sowie $C[\infty,b]:=\bigcup\{C[a,b] : a\in\mathbb{N}_0$ und $a>b\}$ für $b\in\mathbb{N}_0$.
Die Speicherinstruktionen inc,dec,pop,push,shl,shr,pred und succ seien die folgenden Abbildungen von $C[\infty]$ auf sich: ($n\dot{-}m= n-m$ falls $n>m$, 0 sonst)

$\mathrm{inc}((n_0,n_1,\ldots,n_i,j))=(n_0+1,n_1,\ldots,n_i,j))$; $\mathrm{dec}((n_0,n_1,\ldots,n_i,j))=(n_0\dot{-}1,n_1,\ldots,n_i,j)$;
$\mathrm{push}((n_0,n_1,\ldots,n_i,j))=(0,n_0,n_1,\ldots,n_i,j)$
$\mathrm{pop}((n_0,n_1,\ldots,n_i,j))=(n_1,\ldots,n_i,j)$ falls $i>j$, anderenfalls $(0,0)$
$\mathrm{shr}((n_0,n_1,\ldots,n_i,j))=(0,n_0,n_1,\ldots,n_{i-1},j)$;
$\mathrm{shl}((n_0,n_1,\ldots,n_i,j))=(n_1,\ldots,n_i,0,j)$ falls $i>j$, anderenfalls $(n_0,n_1,\ldots,n_i,j)$;
$\mathrm{pred}((n_0,n_1,\ldots,n_i,j))=(n_0,n_1,\ldots,n_i,j+1)$ falls $i>j$, anderenfalls $(n_0,n_1,\ldots,n_i,j)$;
$\mathrm{succ}((n_0,n_1,\ldots,n_i,j))=(n_0,n_1,\ldots,n_i,j\dot{-}1)$

Weiterhin seien die Funktionen zero und comp von $C[\infty]$ auf $\{0,1\}$ wie folgt erklärt:

$\mathrm{comp}((n_0,n_1,\ldots,n_i,j))=1$ falls $n_0=n_j$, 0 sonst.
$\mathrm{zero}((n_0,n_1,\ldots,n_i,j))=1$ falls $n_0=0$, 0 sonst.

Wenn $\emptyset\neq U\subseteq W$ und $f:W\to W$, so bezeichne f_U die wie folgt definierte Funktion von U auf sich: $f_U(u)=f(u)$ falls $f(u)\in U$, u anderenfalls.
Als *verschachtelte und vergleichbare Zähler* wird ein Zusatzspeicher $\mathcal{C}(C[\alpha,b],s_0)=(C[\alpha,b],\{1,0\}^2,P_{C[\alpha,b]},\tau_{C[\alpha,b]},s_0)$ mit $\alpha\in\mathbb{N}_0\cup\{\infty\}$, $b\in\mathbb{N}_0$ und $s_0\in S$ bezeichnet, wobei $P_{C[\alpha,b]}=\{\mathrm{inc}_{C[\alpha,b]},\mathrm{dec}_{C[\alpha,b]},\mathrm{push}_{C[\alpha,b]},\mathrm{pop}_{C[\alpha,b]},\mathrm{pred}_{C[\alpha,b]},\mathrm{succ}_{C[\alpha,b]}\}$, und für alle $s\in S$ $\tau_{C[\alpha,b]}(s)=(\mathrm{zero}_{C[\alpha,b]},\mathrm{comp}_{C[\alpha,b]})$ gilt. Für $a,b\in\mathbb{N}_0$ und $0\leq b<a$ sei $SR[a,b]:=\mathbb{N}_0^a\times\{0,\ldots,a\}$. Als *Schieberegister mit Vergleich* wird ein Zusatzspeicher $\mathcal{C}(SR[a,b],s_0)=(SR[a,b],\{1,0\}^2,P_{SR[a,b]},\tau_{SR[a,b]},s_0)$ mit $a,b\in\mathbb{N}_0$ und $0\leq b<a$ bezeichnet, wobei für alle $s\in S$ $\tau_{SR[a,b]}(s)=(\mathrm{zero}_{SR[a,b]},\mathrm{comp}_{SR[a,b]})$ gilt und $P_{SR[a,b]}=\{\mathrm{inc}_{SR[a,b]},\mathrm{dec}_{SR[a,b]},\mathrm{shl}_{SR[a,b]},\mathrm{shr}_{SR[a,b]},\mathrm{pred}_{SR[a,b]},\mathrm{succ}_{SR[a,b]}\}$.

Ein springender Mehrkopfautomat, der in der Lage ist, auf einem Graphen mit Kompaßsystem $L=(V,H,I,c)$ zu operieren, besteht aus einer endlichen Steuereinheit, die gegebenenfalls mit einem Zusatzspeicher versehen ist, und endlich vielen Köpfen. Ein Kopf steht dabei auf einer Halbkante $h'\in H$ und sieht die Menge der Richtungen $c(H(v))=\{c(h):h\in H\}$ des betretenen Knotens v, für den $(h',v)\in I$ gilt, die Richtung $c(h')$ der eigenen Position sowie gegebenenfalls die Richtungen weiterer Köpfe, die v betreten. Dabei sind die Köpfe in der Lage, einander zu unterscheiden.
Mittels Steuereinheit und den über die Köpfe erhaltenen Informationen geht der Automat in einen Zustand über und steuert die Bewegungen der Köpfe, so daß

diese auf ihrer Position verbleiben, durch eine Halbkante zur nächsten Position geführt werden oder zur Position eines anderen Kopfes springen.
Ein Kopf eines springenden Mehrkopfautomaten, der auf R-Graphen arbeitet, sieht die Anzahl der mit dem betretenen Knoten v inzidierenden Halbkanten card(H(v)) und gegebenenfalls die Position anderer Köpfe relativ zu seiner Position bezüglich des Rotationssystems. Die Steuerung der Köpfe erfolgt hinsichtlich des Rotationssystems.
Ein springender Mehrkopfautomat *sucht einen C- bzw. R-Graphen ab*, wenn jeder Knoten des Graphen während seiner Arbeit durch mindestens einen Kopf betreten wird.
Für eine formale Beschreibung eines springenden Mehrkopfautomaten als Mealyautomat $\mathcal{U}=(X,Y,Z,\delta,\lambda,z_0)$ für C- bzw R-Graphen wird auf [1] verwiesen.

4. Resultate

Ein springender Mehrkopfautomat mit einem Zusatzspeicher $\mathcal{C}(C[\alpha,b],s_0)$ bzw $\mathcal{C}(SR[a,b],s_0)$ wird mit JMCA[α,b] bzw. JMSRA[a,b] abgekürzt.

Satz 1: Es gibt sowohl keinen JMCA[a,b] als auch keinen JMSRA[a+1,1], wobei $a,b\in\mathbb{N}_0$ mit $0\leq b<a$, der eine der nachfolgenden Klassen absucht:
a) alle endlichen, zusammenhängenden, planaren 3-regulären EC-Graphen,
b) alle endlichen, zusammenhängenden, planaren 3-regulären R-Graphen,
c) alle endlichen, zusammenhängenden, planaren 3D-Graphen.

Satz 2: Es gibt sowohl einen JMCA[∞,1] als auch einen JMSRA[3,2], der alle zusammenhängenden C-Graphen mit einer Richtungsmenge D absucht
Es gibt sowohl einen JMCA[∞,1] als auch einen JMSRA[3,2], der alle zusammenhängenden R-Graphen mit einem Grad $\leq B$, $B\in\mathbb{N}_0$, absucht.

Aus Platzgründen muß hier auf die Beweise verzichtet werden.

[1] Bull, M., A. Hemmerling: Traps for jumping multihead counter automata, erscheint in EIK
[2] Cook, S.A., C.W. Rackoff, Space lower bounds for maze threadability on restricted machines. SIAM J. Comput., Vol.9, No.3, 1980, 636-652
[3] Hemmerling, A., Labyrinth problems - labyrinth-searching abilities of automata. Teubner-Texte zur Mathematik, Vol.114, Leipzig 1989
[4] Savitch, W., Maze recognizing automata and nondeterministic tape complexity. JCSS 7 (4), 1973, 38

Abschlußeigenschaften der wachsend kontextsensitiven Sprachen

(Zusammenfassung, Oktober 1992)

Gerhard Buntrock

Institut für Informatik, Universität Würzburg, 8700 Würzburg

Die wachsend kontextsensitiven Sprachen sind durch kontextsensitive Grammatiken definiert, deren Überführungsregeln Satzformen stets in längere Satzformen überführen. In Wien auf der ICALP wurde bewiesen, daß diese Sprachklasse unter den AFL-Operationen abgeschlossen ist.

Für die kontextfreien Sprachen ist bekannt, daß sie eine echte Durchschnittshierarchie bilden. Für die wachsend kontextsensitiven ist dies nicht bekannt. Hier werden die Operationen „Durchschnitt von Sprachen" sowie „Mischen", „perfektes Mischen" und „Twist von Wörtern" betrachtet. Letztere Operation ist kürzlich von Matthias Jantzen und Holger Petersen eingeführt worden Es ist nicht schwierig nachzuweisen, daß weder die kontextfreien noch die hier betrachteteten wachsend kontextsensitiven Sprachen abgeschlossen unter einer der genannten Operationen sind. Für die letztere Klasse benutzt man hier im wesentlichen die Eigenschaft, daß wachsend kontextsensitive Sprachen von logarithmisch platzbeschränkten Einweg-Hilfskellermaschinen akzeptiert werden (siehe Vortrag in Magdeburg auf dem ersten *Theorietag für Automaten und Formale Sprachen*). Aber man kann zeigen, daß der Abschluß unter Durchschnitt im Abschluß unter perfekten Mischen und den AFL-Operationen und dieser wiederum im Abschluß unter Twist und den AFL-Operationen und gleichzeitig im Abschluß unter Mischen und den AFL-Operationen enthalten ist.

ON SOLVING DECISION PROBLEMS FOR SUBGROUPS OF CONTEXT-FREE GROUPS *

- Abstract -

Friedrich Otto
Fachbereich Mathematik/Informatik, Gesamthochschule Kassel
3500 Kassel, West Germany

October 21, 1992

Let G be a group that is presented by a monoid-presentation $(\Sigma; R)$, and let U be a finite set of strings from Σ^*. Then $\langle U \rangle$ denotes the subgroup of G that is generated by U. Here we are concerned with the following algorithmic problems concerning $\langle U \rangle$:

- Given a string $w \in \Sigma^*$, decide whether $w \in \langle U \rangle$! This is the **membership problem** for $\langle U \rangle$, and if the set U is also taken as a part of the problem instance, then this problem is known as the **generalized word problem** for G.
- Decide whether the index $|G : \langle U \rangle|$ of $\langle U \rangle$ in G is finite, and if so, compute this number!
- From $(\Sigma; R)$ and U determine a monoid-presentation for the subgroup $\langle U \rangle$!

Since these problems are undecidable in general we restrict our attention to a particular class of groups: the context-free groups. A group G is called **context-free** if it has a finite monoid-presentation $(\Sigma; R)$ such that the congruence class $[\lambda]_R$ is a context-free language. Here λ denotes the empty string. Although this definition is based on a particular presentation, it is easily verified that the actually chosen finite monoid-presentation for G is not important, since the property that the congruence class of the empty string is a context-free language is an invariant of finite monoid-presentations. Thus, the context-free groups are defined by a language-theoretical property of their presentations. An algebraic characterization of these groups has been given by Muller and Schupp [MuSc83]: A finitely generated group is context-free if and only if it is virtually free. Finally, these groups have been characterized by syntactic properties of presentations by Autebert et al [ABS87]: A group is context-free if and only if it admits a finite monoid-presentation $(\Sigma; R)$ such that the string-rewriting system R is monadic and λ-confluent. Here a string-rewriting system R is called **monadic**, if $|\ell| \geq |r|$ and $|r| \leq 1$ hold for each rule $(\ell \to r) \in R$, and it is called **λ-confluent**, if $w \longrightarrow_R^* \lambda$ holds for each string $w \in [\lambda]_R$.

Let $\mathcal{C}_{m,\lambda}$ denote the following class of finite monoid-presentations:

*Joint work with Norbert Kuhn, DFKI, 6600 Saarbrücken, and Klaus Madlener, Fachbereich Informatik, Universität Kaiserslautern, 6750 Kaiserslautern [KMO92]

$$\mathcal{C}_{m,\lambda} \quad := \quad \{(\Sigma; R) \mid R \text{ is monadic and } \lambda\text{-confluent, and the monoid } M_R \text{ presented by } (\Sigma; R) \text{ is a group } \}.$$

It is decidable in polynomial time whether a presentation $(\Sigma; R)$ belongs to $\mathcal{C}_{m,\lambda}$ [MNOZ91], and for $(\Sigma; R) \in \mathcal{C}_{m,\lambda}$, we can determine a function $^{-1} : \Sigma^* \to \Sigma^*$ in polynomial time such that, for $w \in \Sigma^*$, w^{-1} is a **formal inverse** of w.

For $w \in \Sigma^*$, let $I_R(w) := [w]_R \cap IRR(R)$, and for $S \subseteq \Sigma^*$, let $I_R(S) := [S]_R \cap IRR(R)$, where $IRR(R)$ denotes the set of strings that are irreducible *mod* R. Then $I_R(w)$ is always a finite set, implying that the group G presented by $(\Sigma; R)$ is finite if and only if the set $IRR(R)$ is finite, and $I_R(S)$ is a regular set, whenever S is. Thus, if $U \subseteq \Sigma^*$ is a finite set, then $I_R((U \cup U^{-1})^*)$ is a regular set of irreducible representatives of $\langle U \rangle$. Since a nondeterministic finite-state acceptor for this set can be constructed in polynomial time, this means that the generalized word problem for G is tractable.

By considering prefix-rewriting systems we now replace the regular set $I_R((U \cup U^{-1})^*)$ of representatives of $\langle U \rangle$ by the single representative λ. To simplify the notation we assume that U is closed under taking inverses. Let $P_R := \{(x\ell, xr) \mid x \in \Sigma^*, (\ell \to r) \in R\}$, let $P_U = \{(u_i, v_i) \mid i = 1, \ldots, m\}$ be such that, for all $i = 1, \ldots, m$, $u_i v_i^{-1} \longleftrightarrow_R^* w$ for some $w \in U$, and, for each $w \in U$, there exists some i such that $w \longleftrightarrow_R^* u_i v_i^{-1}$, let $P := P_U \cup P_R$, and let $\Longrightarrow_P$ denote the **prefix-reduction relation** induced by P. Then $x \Longleftrightarrow_P^* y$ if and only if $xy^{-1} \in \langle U \rangle$ [KuMa89], i.e., $\Longleftrightarrow_P$ represents the right-congruence induced by $\langle U \rangle$. If $\Longrightarrow_P$ does not admit infinite reduction sequences, and if it is λ-confluent, then we can conclude that, for all $w \in \Sigma^*$, $w \in \langle U \rangle$ if and only if $w \Longrightarrow_P^* \lambda$, i.e., in this situation λ is the unique representative of $\langle U \rangle$ *mod* $\Longrightarrow_P$.

It remains to determine a finite set P_U of prefix-rules having all the above properties. As it turns out such a set can be extracted from a nondeterministic finite-state acceptor for the set $I_R(U^*)$. In fact, P_U will then contain length-reducing rules only. Thus, we have the following result.

Theorem 1 *For each $(\Sigma; R) \in \mathcal{C}_{m,\lambda}$ and each finite subset $U \subseteq \Sigma^*$, there exists a finite, length-reducing set of prefix-rules P_U such that the prefix-rewriting system $P := P_U \cup P_R$ presents the right-congruence induced by $\langle U \rangle$ and is λ-confluent. In fact, P_U can be constructed effectively from $(\Sigma; R)$ and U.*

In addition, given a string $w \in \Sigma^*$, if $w \in \langle U \rangle$, then a product $u \in U^*$ can be determined effectively such that $u \longleftrightarrow_R^* w$, i.e., we have an effective **rewrite process** with respect to the given generators of $\langle U \rangle$.

With respect to $\Longrightarrow_P$, λ is the only irreducible representative of $\langle U \rangle$. However, the other cosets of $\langle U \rangle$ may contain more than one irreducible string *mod* $\Longrightarrow_P$. Now using properties of the prefix-rewriting relation $\Longrightarrow_P$ it can be shown that each such coset contains only finitely many of these irreducible strings. Thus, the index $|G : \langle U \rangle|$ is finite if and only if the set $IRR(\Longrightarrow_P)$ is finite. Since this set is regular, we can decide whether or not it is finite, and if it is, then the index $|G : \langle U \rangle|$ can actually be determined.

Corollary 2 *Let $(\Sigma; R) \in \mathcal{C}_{m,\lambda}$, and let $U \subseteq \Sigma^*$ be a finite set. Then the index $|G : \langle U \rangle|$ of the subgroup $\langle U \rangle$ in the group G defined by $(\Sigma; R)$ is effectively computable.*

If R is a finite string-rewriting system on Σ such that there is an involution $^{-} : \Sigma \to \Sigma$ with $a\bar{a} \longleftrightarrow_R^* \lambda$ $(a \in \Sigma)$, then we call $\langle \Sigma; R \rangle$ a **group-presentation**. Each context-free group G can be presented by a finite group-presentation $\langle \Sigma; R \rangle$ such that R is monadic

and λ-confluent. Since each finitely generated subgroup H of a context-free group G is itself context-free, it also has a presentation of this very form. In fact, given $\langle \Sigma; R \rangle$ and U, a group-presentation $\langle \Gamma; T \rangle$ for $\langle U \rangle$ can be constructed such that T is monadic and λ-confluent. This construction consists of three phases.

First, a deterministic finite-state acceptor A is constructed such that $\Delta_R^*(U^*) \subseteq L(A) \subseteq \langle U \rangle$. Then from $\langle \Sigma; R \rangle$ and A a presentation $\langle \Gamma; S \rangle$ for $\langle U \rangle$ is extracted such that S is monadic. This part of the construction is essentially the literal translation of a well-known construction by Reidemeister and Schreier [MKS66]. However, the system S will in general not be λ-confluent. By normalizing S we finally obtain a finite, monadic, and λ-confluent system T that is equivalent to S, i.e., $\langle \Gamma; T \rangle$ is the intended presentation for $\langle U \rangle$.

Theorem 3 *Let $\langle \Sigma; R \rangle \in \mathcal{C}_{m,\lambda}$, and let U be a finite subset of Σ^*. Then a presentation $\langle \Gamma; T \rangle \in \mathcal{C}_{m,\lambda}$ for the subgroup $\langle U \rangle$ of the group G presented by $\langle \Sigma; R \rangle$ can be constructed in polynomial time.*

In addition, a rewrite process $\tau : \langle U \rangle \to \Gamma^*$ can be obtained.

References

[ABS87] J.Autebert, L.Boasson, G.Senizergues; *Groups and NTS languages*; J. Computer System Sciences 35 (1987) 243-267.

[KuMa89] N.Kuhn, K.Madlener; *A method for enumerating cosets of a group presented by a canonical system*; in: G.Gonnet (ed.), Proceedings ISSAC'89 (ACM, New York, 1989) 338-350.

[KMO92] N.Kuhn, K.Madlener, F.Otto; *Computing presentations for subgroups of context-free groups*; in: P. Wang (ed.), Proceedings of ISSAC'92 (ACM, New York, 1992) 240-250.

[MNOZ91] K.Madlener, P.Narendran, F.Otto, L.Zhang; *On weakly confluent monadic string-rewriting systems*; Mathematische Schriften, Preprint No. 11/91 (Gesamthochschule Kassel, 1991); also: Theoretical Computer Science, to appear.

[MKS66] W.Magnus, A.Karrass, D.Solitar; *Combinatorial Group Theory* (Wiley-Interscience, New York, 1966).

[MuSc83] D.E.Muller, P.E.Schupp; *Groups, the theory of ends and context-free languages*; J. Computer System Sciences 26 (1983) 295-310.

Bisimulation of Pushdown Automata

Didier Caucal

Irisa, Campus de Beaulieu
35042 Rennes, France
E-mail: caucal@irisa.fr

Two vertices of a labelled graph are bisimilar [Park 81] if their unfolded trees are reduced to the same tree by iterated identifying of identical subtrees. Two pda (pushdown automata) are bisimilar if the axioms of their transition graphs are bisimilar. Note that this bisimulation is strong in the sense that it does not take into account the empty transitions: the empty word is like an ordinary terminal.

First, we recall some basic results. The class of pushdown transition graphs are the rooted regular (equational) graphs of finite degree, and contains strictly the class of prefix transition graphs of context-free grammars. Furthermore, the bisimulation of (the axioms of the prefix transition graphs of) context-free grammars is decidable [Baeten, Bergstra, Klop 87], even when they are not reduced [Hüttel, Stirling 92]. Moreover, the bisimulation for reduced (real-time) dpda (d for deterministic) is decidable because it is interreducible to the decidable equivalence problem [Oyamaguchi, Honda 78] for real-time dpda. This implies that the bisimulation is decidable for the "finitely coaccessible" deterministic graphs with finite degree.

Then, we show that the bisimulation for (real-time) dpda (even if they are not reduced) is interreducible to the equivalence problem for dpda. This means that the bisimulation of deterministic regular graphs with finite degree is interreducible to the equivalence problem for dpda. This interreducibility remains true for the bisimulation of deterministic regular graphs with finite out-degree (the in-degree may be infinite). As a corollary, the equivalence problems for dpda and recursive program schemes are interreducible [Courcelle, Gallier 87].

Graph Grammars with Set Productions

Rudolf Freund
Institut für Computersprachen, Technische Universität Wien
A-1040 Wien, Karlsplatz 13, AUSTRIA
email: freundαcsdec1.tuwien.ac.at

Abstract. A new mechanism for generating graph languages is introduced, which is based on the controlled rewriting of graphs using sets of strictly locally operating graph productions, corresponding to the adding or the deleting of a node or an edge with a specified label or the changing of the label of a node or an edge. A hierarchy of these *programmed graph grammars with set productions* (**PROSE** grammars) and normal forms for various kinds of PROSE grammars are established. Finally some unsolved problems are stated.

0. Introduction

In [Freund 91] we introduced *elementary programmed graph grammars*, which are based on six elementary types of strictly locally operating graph productions, namely the adding of a new node or a new edge with a given label, the deleting of a node or an edge with the required label and the changing of the label of a node or an edge between two nodes with specified labels. Programmed graph grammars with set productions (**PROSE grammars**) are a natural extension of elementary programmed graph grammars (which already are powerful enough to generate any recursively enumerable set of equivalence classes of directed graphs) and allow efficient and elegant solutions for various problems in the theory of graph languages.
The applicability of the attributed version of elementary programmed graph grammars was shown in [Freund Haberstroh]. Attributed PROSE grammars can be used as efficient rewriting systems for attributed graphs in several fields of computer science, e. g. in software engineering ([Freund Haberstroh Stary]) and the description of neural networks ([Freund Haberstroh Bischof]). Various applications in these areas will also be part of future research.
Before introducing the new approach of *programmed graph grammars with set productions* we shall recall some necesssary definitions for graphs.

1. Preliminary Definitions

The structures to be considered in the following are finite directed graphs with labelled edges and nodes (loops are allowed, whereas multiple edges are not).

Definition 1.1. A ***graph*** over (V, W) is a tuple $g = (\mathcal{N}, n, \mathcal{E})$, where V and W are alphabets for labelling the nodes and edges, $\mathcal{N}$ is a finite set of ***nodes***, $\mathcal{E} \in \mathcal{P}(\mathcal{N} \times W \times \mathcal{N})$ ($\mathcal{P}(M)$ denotes the powerset of the set M) and $n: \mathcal{N} \to V$ is the labelling function for the nodes in $\mathcal{N}$. Any tuple (u, k, v) with $u, v \in \mathcal{N}$ and $k \in W$ can be interpreted as a ***(directed) edge*** from the node u to the node v and labelled by k. In the following, the notations $\mathcal{N}(g)$, $\mathcal{E}(g)$ resp. $n(g)$ will be used to identify the nodes and edges respectively the labelling function for the nodes of the graph g.

Definition 1.2. Let $\gamma(V, W)$ denote the set of all graphs over (V, W) and g_e the ***empty graph***, i. e. the graph with an empty set of nodes.

Definition 1.3. Let $g, g' \in \gamma(V, W)$; then g and g' are called *isomorphic* – abbreviated $g \cong g'$ – iff there exists a bijective mapping $f: \mathcal{N}(g) \to \mathcal{N}(g')$ with $(u, k, v) \in \mathcal{E}(g) \Leftrightarrow (f(u), k, f(v)) \in \mathcal{E}(g')$ for any $u, v \in \mathcal{N}(g)$ and $k \in W$; g and g' are called *equivalent* – abbreviated $g \equiv g'$ – iff $g \cong g'$ and, moreover, $n(g) = n(g') \circ f$, where "$\circ$" denotes the composition of functions or relations, e. g. $(h \circ f)(x) = h(f(x))$. For any $g \in \gamma(V,W)$ let $[g]$ denote the class of all graphs of $\gamma(V,W)$ equivalent to g.
Furthermore we define $\Gamma(V,W) := \gamma(V,W)/\equiv$, i. e. $\Gamma(V,W) = \{ [g] \mid g \in \gamma(V,W) \}$.

Definition 1.4. A graph $g \in \gamma(V,W)$ is called ***discrete*** iff $\mathcal{E} = \emptyset$. A graph $g \in \gamma(V,W)$ is called ***complete*** iff $\mathcal{E} = \mathcal{N} \times W \times \mathcal{N}$. Furthermore, $h \in \Gamma(V,W)$ is called discrete (respectively complete) iff there exists a discrete (respectively complete) graph $g \in h$.

2. PROSE grammars

In the following we will introduce the new approach of *programmed graph grammars with set productions* (***PROSE grammars*** for short).

Definition 2.1.
Let λ be a special (*blank*) symbol, and let V,W be alphabets; then we define $\mathbf{V_\lambda} := V \cup \{\lambda\}$ and $\mathbf{W_\lambda} := W \cup \{\lambda\}$. The set of *node productions* over (V,W) is defined by $R_N(V,W) := V_\lambda{}^2 - \{\lambda\}^2$; the set of *edge productions* over (V,W) is defined by $R_E(V,W) := (V \times W_\lambda \times V)^2 - (V \times \{\lambda\} \times V)^2$. Thus an element of $R_N(V,W)$ is of the form (A,B) with $A,B \in V_\lambda$, but $\{A,B\} \neq \{\lambda\}$, and an element of $R_E(V,W)$ is of the form (A,j,B;C,k,D) with $A,B,C,D \in V$ and $j,k \in W_\lambda$, but $\{j,k\} \neq \{\lambda\}$.
Any element of $R(V,W) := R_N(V,W) \cup R_E(V,W)$ is called a ***graph production***. any subset of R(V,W) is called a ***set of graph productions*** *or* ***set production***.
The result of applying such a graph production to a graph $g \in \gamma(V,W)$ yields a new graph $g' \in \gamma(V,W)$, which depends on the special form of this graph production:

(1) *(λ,B): add a new node v with label B.*
(2) *(A,B): change the label of a node u from A to B.*
(3) *(A,λ): delete a node u with label A* (provided u is isolated).
(4) *(A,λ,B;C,k,D): add an edge with label k between a node u with label A and a node v with label B and change these node labels from A to C and from B to D, respectively* (provided an edge (u,k,v) does not already exist).
(5) *(A,j,B;C,k,D): change the label of an edge between a node u with label A and a node v with label B from j to k* (provided an edge (u,k,v) does not already exist) *and change these node labels from A to C and from B to D, respectively.*
(6) *(A,j,B;C,λ,D): delete an edge with label j between a node u with label A and a node v with label B and change these node labels from A to C and from B to D, respectively.*

Observe that in the case of the edge productions of the form (A,j,B;C,k,D) we may also choose the same node for u and v, if A = B, but only when C = D, too.
In [Freund 91] we restricted ourselves to ***elementary graph productions***, i. e. node productions as described above and edge productions of the form $(A,j,B;A,k,B) \in R_E(V,W)$.

Definition 2.2.
Let $p \subseteq R(V,W)$, i. e. p is a *set production*. If the graph $g' \in \gamma(V,W)$ can be obtained by applying a graph production $q \in p$ to the graph $g \in \gamma(V,W)$, then g' is said to be ***directly derivable*** *from g by p* - abbreviated $g \overset{\Rightarrow}{p} g'$; $h' \in \Gamma(V,W)$ is *directly derivable from* $h \in \Gamma(V,W)$ *by p* - abbreviated $h \overset{\Rightarrow}{p} h'$ - iff there is a $g \in h$ and a $g' \in h'$ with $g \overset{\Rightarrow}{p} g'$. If no graph production $q \in p$ can be applied to g, then p is said to be ***not applicable*** to g.

Definition 2.3. A ***programmed graph grammar with set productions*** *(PROSE grammar for short)* is a 7-tuple $G = (V_N, W_N, V_T, W_T, P, P_0, P_f)$ such that

(1) $V_N \cup V_T$ and $W_N \cup W_T$ are alphabets for labelling the nodes and the edges, respectively, where the elements of V_N and W_N are nonterminals and the elements of V_T and W_T are terminals; $V_N \cap V_T = W_N \cap W_T = \emptyset$; moreover, we define $V = V_N \cup V_T$ and $W = W_N \cup W_T$;
(2) P is a finite set of triples ***(rules)*** $(r: p, \sigma(r), \varphi(r))$ where p is a set of graph productions in R(V,W) labelled by r and $\sigma(r)$ and $\varphi(r)$ are two sets of labels of such core rules in P, i. e. σ and φ are two mappings from Lab(P) to $\mathfrak{P}(\mathrm{Lab}(P))$, Lab(P) denoting the set of labels of the productions in P, i. e.
$\mathrm{Lab}(P) = \{ r \mid (r: p, \sigma(r), \varphi(r)) \in P \}$;

(3) $P_0 \subseteq \mathrm{Lab}(P)$ is a set of initial labels (of initial rules);
(4) $P_f \subseteq \mathrm{Lab}(P)$ is a set of final labels (of final rules).

Moreover, we define $\mathfrak{p}(r) := p$ for each rule $(r: p,\sigma(r),\varphi(r))$ in P.
Furthermore, a PROSE grammar $G = (V_N, W_N, V_T, W_T, P, P_0, P_f)$ is called ***monotone***, if no set $\mathfrak{p}(r)$ in P contains a node production of the form (A,λ) with $A \in V$; if, in addition, no set $\mathfrak{p}(r)$ in P contains an edge production of the form (A,j,B;C,λ,D) with $A,B,C,D \in V$ and $j \in W$, then G is called ***strictly monotone***. If every set production in P contains only node productions, then G is said to be ***discrete***.

Definition 2.4. Let $G = (V_N, W_N, V_T, W_T, P, P_0, P_f)$ be a PROSE grammar, $r,r' \in \mathrm{Lab}(P)$, $(r: p, \sigma(r), \varphi(r)) \in P$ and $g,g' \in \gamma(V,W)$. The pair (g',r') is ***directly derivable*** from (g,r) ***in G*** - abbreviated $(g,r) \vdash_G (g',r')$ - iff either

(1) $g \overset{\Rightarrow}{p} g'$ and $r' \in \sigma(r)$ (which means that the set production p can be applied to the graph g, yielding the new graph g') and we also write $(g,r) \overset{Y}{\vdash_G} (g',r')$ - or
(2) $g' = g$, the production p cannot be applied to the graph g, and $r' \in \varphi(r)$; in this case we also write $(g,r) \overset{N}{\vdash_G} (g',r')$.

For $h,h' \in \Gamma(V,W)$ we define $(h,r) \vdash_G (h',r')$ (respectively $(h,r) \overset{Y}{\vdash_G} (h',r')$ and $(h,r) \overset{N}{\vdash_G} (h',r')$) iff there exists a $g \in h$ and a $g' \in h'$ such that $(g,r) \vdash_G (g',r')$ (respectively $(g,r) \overset{Y}{\vdash_G} (g',r')$ and $(g,r) \overset{N}{\vdash_G} (g',r')$).
As usual, $\overset{*}{\vdash_G}$ denotes the reflexive and transitive closure of the relation $\vdash_G$.
The ***graph language generated by G*** is
$\mathcal{L}(G) = \{ h \mid h \in \Gamma(V_T,W_T)$ and $([g_\varepsilon], r_0) \overset{*}{\vdash_G} (h, r_f)$ for some $r_0 \in P_0$ and some $r_f \in P_f \}$.
Moreover, a **graph language** is called ***monotone***, ***strictly monotone*** *resp.* ***discrete***, if it can be generated by a monotone, strictly monotone resp. discrete PROSE grammar.

Example 2.1.
Using the rule $(r: \{(A,k,B;A,\lambda,B) \mid k \in W,\ B \in V\}, \{r\}, \varphi(r))$ gives us the possibility to delete all edges leaving a node with a given label A.

Example 2.2. *(Complete graphs)*
The PROSE grammar $G = (\emptyset, \emptyset, \{a\}, W_T, P, \{1\}, \{3\})$ with
$P = \{(1: \{(\lambda,a)\}, \{1,2\}, \emptyset), (2: \{(a,\lambda,a;a,k,a) \mid k \in W_T\}, \{2\}, \{3\})\}$
generates all complete graphs over $(\{a\}, W_T)$.
In a more illustrative way this PROSE grammar can be depicted as follows:

$$\begin{array}{ccccc} start \rightarrow & 1: \{(\lambda,a)\} & \overset{Y}{\rightarrow} & 2: \{(a,\lambda,a;a,w,a) \mid w \in W_T\} & \overset{N}{\rightarrow} end \\ & \circlearrowleft Y & & \circlearrowleft Y & \end{array}$$

In the diagram above a rule $(r: p,\sigma(r),\varphi(r))$ is represented by r:p and edges leaving this node of the diagram; these edges are labelled by Y(es) or N(o) and end up in the corresponding nodes representing the rules in $\sigma(r)$ and $\varphi(r)$, respectively.

Definition 2.5. An ***elementary programmed graph grammar*** is a PROSE grammar $G = (V_N, W_N, V_T, W_T, P, P_0, P_f)$ such that every set production in P contains exactly one elementary graph production.

As elementary programmed graph grammars only are restricted versions of PROSE grammars, the results proved in [Freund 91] immediately yield the following theorems:

Theorem 2.1. Any recursively enumerable graph language can be generated by a PROSE grammar.

Theorem 2.2. Any recursively enumerable graph language of discrete graphs can be generated by a discrete PROSE grammar.

Theorem 2.3. The family of graph languages generated by monotone PROSE grammars exactly coincides with the family of monotone graph languages as defined in [Nagl] and [Bunke].

Definition 2.6. Two *PROSE grammars* G and G' are called ***equivalent*** iff $\mathcal{L}(G) = \mathcal{L}(G')$.

The following theorem shows that elementary programmed graph grammars represent a special normal form for PROSE grammars (for a proof of this result see [Freund 92]).

Theorem 2.4. For any (unrestricted, monotone, strictly monotone, discrete) PROSE grammar we can effectively construct an equivalent (unrestricted, monotone, strictly monotone, discrete) elementary programmed graph grammar.

Because of theorem 2.4 equivalent elementary programmed graph grammars for the PROSE grammars in the examples 2.1 and 2.2 must exist, but this construction would yield graph grammars which are much more difficult to describe and therefore much less descriptive.

Investigating the influence of the three different types of edge productions, several interesting problems arise, most of them still waiting to be solved:

First of all, it is easy to show that in the case of arbitrary and monotone PROSE grammars graph productions changing the label of an edge are not necessary (they can be simulated by using only edge productions deleting respectively adding an edge).

But when we turn to strictly monotone graph grammars the following problems arise:

Problem 1. Is the family of graph languages generated by strictly monotone PROSE grammars strictly included in the family of monotone graph languages?

Problem 2. Which is the effect on the generative power of PROSE grammars when we only allow edge productions of the form (A,λ,B;C,k,D)?

Acknowledgements. I gratefully appreciate the interesting and fruitful dicussions with Brigitte Haberstroh on the topics presented in this paper.

3. References

[Bunke]: Bunke. Horst: On the Generative Power of Sequential and Parallel Programmed Graph Grammars. Computing 29, 89-112, 1982.

[Dassow Păun]: Dassow, Jürgen, Păun, Gheorghe: Regulated Rewriting in Formal Language Theory. Springer-Verlag, Berlin, Heidelberg, New York, 1989.

[Freund 91]: Freund, Rudolf: Elementary Programmed Graph Grammars, Technical report 185/2/F/GG1/91, Technical University of Vienna, 1991.

[Freund 92]: Freund, Rudolf: Graph Grammars with Set Productions, Technical report 185/2/F/GG1/92, Technical University of Vienna, 1992.

[Freund Haberstroh]: Freund, Rudolf, Haberstroh, Brigitte, Attributed Elementary Programmed Graph Grammars, Proceedings of the 17th International Workshop on Graph-Theoretic Concepts in Computer Science, LNCS 570, Springer, 1992.

[Freund Haberstroh Bischof]: Freund, Rudolf, Haberstroh, Brigitte, Bischof, Horst: Tools for Dynamic Network Structures: GRAPE Grammars, IJCNN '92, Baltimore, 1992.

[Freund Haberstroh Stary]: Freund, Rudolf, Haberstroh, Brigitte, Stary, Christian: Applying Graph Grammars for Task-Oriented User Interface Development, 4th International Conference on Computing and Information '92, Toronto, 1992.

[Nagl]: Nagl, Manfred: Formal Languages of Labelled Graphs, Computing 16, 113-137, 1976.

[Salomaa]: Salomaa, Arto: Formal Languages. Academic Press, New York, London, 1973.

Automata Theoretic Considerations in Circuit Testing[1]

J. A. Brzozowski[2], H. Jürgensen[3]

1. Introduction

There are numerous causes of failures in integrated circuits and larger digital networks, and careful testing is an indispensable step in the production of a digital system. The generation of tests and their application is a very time-consuming and expensive process. Usually it is impossible or at least far too costly to test for all possible causes of failures in a circuit. Instead, one considers only those physical faults that are likely to occur. The set of these physical faults forms the *physical fault model*.

The *stuck-at fault model* is an example of a frequently used fault model. In the stuck-at model one assumes that the only physical faults that could be present in the circuit under test result in some circuit components—like wires or memory cells—being unable to take one of the two Boolean values. For instance, a wire which is stuck-at-0 will carry the value 0 regardless of its input value; several physical faults could lead to this behaviour including that of a short to ground.

One needs to distinguish between physical faults and the faulty behaviour of the circuit under test resulting from these faults. Moreover, in some cases a physical fault may lead to a faulty behaviour of the circuit only under special operating conditions—like high temperature or a weak power supply. In the sequel, we assume that the operating conditions are fixed and that one is not really interested in detecting the presence of a physical fault, but just in diagnosing faulty behaviour of the circuit if it exists. Thus we replace the physical fault model by the corresponding *functional fault model* which describes the faulty types of behaviour of the circuit that would result from the given physical faults. Moreover, we focus on faults in sequential circuits

We describe both the intended behaviour of the circuit and the faults in the functional fault model by finite Mealy automta. A similar approach was already used in 1964 by Poage and McCluskey [Po1] and by Hennie [He1], [He2]. Also the work of Graf and Gössel [Gr1] published in 1987 employs similar ideas.

Thus the *good machine type* is a Mealy automaton $A_0 = (Q_0, X, Y_0, \delta_0, \lambda_0)$ where Q_0 is the set of states, X is the input alphabet, Y_0 is the output alphabet, $\delta : Q_0 \times X \to Q_0$ is the transition function, and $\lambda : Q_0 \times X \to Y_0$ is the output function. A *fault type* is another Mealy automaton which may differ from A_0 in the set of states, the output alphabet, the transition function, and the output function. A *good machine* (A_0, q_0) is the good machine type A_0 initialized to some state $q_0 \in Q_0$. Similarly, a *fault* is a fault type initialized to some state. A *fault model* $\mathcal{F}$ for A_0 is defined by the following items: A finite family of fault types $A_1, \ldots, A_n$; for $i = 0, 1, \ldots, n$, a set $P_i \subseteq Q_i$ of *potential initial states*. Given such a fault model, the circuit under test would be compared with every (A_i, q_i) where $i = 0, 1, \ldots, n$ and $q_i \in P_i$. Let $K = \{(i, q, q') \mid i = 0, 1, \ldots, n, q \in P_i, q' \in q_i\}$. To simplify the mathematical presentation we assume that all state sets Q_i are subsets of a common state set Q and that all output alphabets Y_i are subsets of a common alphabet Y.

2. Modelling the Typical Test Situation

In this section we show how to model circuit testing using automata. We then exhibit a general method for specifying the testing goal.

Let A_0 be the good machine type and let $\mathcal{F}$ be a fault model for A_0 involving the fault types $A_1, \ldots, A_n$ as above. Let A' be the circuit under test. A test generator feeds a sequence of inputs into A'. An evaluation unit receives both the test inputs and the outputs of A' and compares

[1] This work was supported by the Information Technology Research Centre of Ontario and by the Natural Sciences and Engineering Research Council of Canada under Grants OGP0000871 and OGP0000243.

[2] Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

[3] Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6A 5B7.

them to the behaviours of each of the machines described by the fault model. It then outputs a temporary diagnosis of A'. This evaluation unit is modelled (essentially) be the initialized deterministic semi-automaton $\Delta = \Delta(A_0, \mathcal{F}) = (D, X \times Y, \delta, d_0)$, called the *(deterministic) observer*, which is defined as follows.

For $i = 0, 1, \ldots, n$, let $\widetilde{Q}_i = Q_i \cup \{\omega\}$ where ω is a new state symbol. Moreover, for $q \in Q_i$ and $(x, y) \in X \times Y_i$, let

$$\widetilde{\delta}_i(q, (x, y)) = \begin{cases} \delta(q, x), & \text{if } \lambda(q, x) = y, \\ \omega, & \text{if } \lambda(q, x) \neq y, \end{cases}$$

and let $\widetilde{\delta}(\omega, (x, y)) = \omega$. A state $d \in D$ is a tuple with components $d_{i,q}$ for $i = 0, 1, \ldots, n$ and $q \in P_i$. Thus $D = \{d \mid d_{i,q} \in \widetilde{Q}_i,\ i = 0, 1, \ldots, n, q \in P_i\}$. The initial state d_0 of Δ is given by $[d_0]_{i,q} = q$. For the transition function, $\delta(d, (x, y)) = d'$ if and only if $d'_{i,q} = \widetilde{\delta}_i(d_{i,q}, (x, y))$ for all i and q.

The diagnosis goal is determined by a partition of the set K. Given such a partition B, a state d of Δ is said to be *B-decided* if the *contents of* d, that is, the set $\{(i, q, q') \mid (i, q, q') \in K,\ q' = d_{i,q} \neq \omega\}$ is contained in a block of B. An input word w is said to *B-diagnose* if and only if $\delta(d_0, (w, v))$ is B-decided for every output word v with $|v| = |w|$. The following are some typical useful partitions:

- The *maximum information partition* B_{MI} having the singleton sets $\{(i, q, q')\}$ for $i = 0, 1, \ldots, n$, $q \in P_i$, and $q' \in Q_i$ as blocks. A state d of Δ is B_{MI}-decided if its contents is empty or if there is exactly one triple (i, q, q') such that $d_{i,q} \neq \omega$ and, moreover, $d_{i,q} = q'$. In the test situation the latter case means that the behaviour of the circuit A' under test has been consistent with that of A_i started in state q, and that the present state of (A_i, q) is q'.
- The *machine partition* B_{M} with the sets $\{(i, q, q') \mid q' \in Q_i\}$ as blocks for $i = 0, 1, \ldots, n$ and $q \in P_i$.
- The *machine type and present state partition* B_{TPS} with the sets $\{(i, q, q') \mid q \in P_i\}$ as blocks for $i = 0, 1, \ldots, n$ and $q' \in Q_i$.
- The *machine type partition* B_{T} with the sets $\{(i, q, q') \mid q \in P_i, q' \in Q_i\}$ as blocks for $i = 0, 1, \ldots, n$.
- The *fault partition* B_{F} with the two sets $\{(0, q, q') \mid q \in P_0, q' \in Q_0\}$ and $\{(i, q, q') \mid i = 1, \ldots, n,\ q \in P_i, q' \in Q_i\}$ as blocks.

The names of these partitions indicate the diagnosis goals defined by the partitions. For instance B_{F} defines the goal of detecting the presence of a fault.

This method of expressing the diagnosis goal in terms of such a partition B and the notion of B-decidedness generalize the work of Hennie on *distinguishing* and *homing* sequences [He1], [He2], [Br3]. The complexity of determining distinguishing sequences has been studied recently in [Ya1]. These complexity results still need to be re-interpreted for our observer model.

Given a diagnosis goal B, the observer can be reduced in two ways. First, for *diagnosability reduction* one considers all B-decided states of Δ as equivalent and then reduces using standard techniques. Second, for *diagnosis reduction* one assigns Moore outputs $\gamma(d)$ to the states d of Δ as follows: $\gamma(d)$ is 0, j, or ∞ if the contents of d is not B-decided, non-empty and contained in block B_j of B, or empty, respectively. One then applies classical reduction techniques to this Moore machine to obtain the diagnosis reduction of Δ. When reduced for diagnosis, the observer will still provide a detailed diagnosis of the circuit under test.

3. Test Sequence Characterization

Given an observer $\Delta = (D, X \times Y, \delta, d_0)$ and a diagnosis goal B, one can construct a non-deterministic finite acceptor $\overline{\Delta} = (D, X, \overline{\delta}, \{d_0\}, F)$ where F is the set of B-decided states of Δ and $\overline{\delta}(d, x) = \{d' \mid \exists y \in Y : \delta(d, (x, y)) = d'\}$. Thus a word w B-diagnoses if and only if every computation of $\overline{\Delta}$ on w is an acccepting computation. A word w is said to be a *reduced B-diagnosing word* if w B-diagnoses and, if w' results from w by omitting a few symbols, then w' does not B-diagnose. Let $L_B(\mathcal{F})$ be the set of B-diagnosing words for the fault model $\mathcal{F}$.

Theorem 3.1 [Br3], [Jü1] *The set $L_B(\mathcal{F})$ is regular. An automaton accepting this set can be constructed from the fault model. The set of reduced B-diagnosing words is a hypercode—hence finite—which can also be constructed from the fault model.*

A program has been developed by C.-J. Shi of the University of Waterloo that performs the construction and reduction steps outlined so far.

4. Fault Coverage

Fault coverage typically refers to two classes of faults[4], the fault model $\mathcal{F}_1$ and another fault set $\mathcal{F}_2$. For instance, in the case of memory testing one could ask the question to what extent testing for stuck-at faults would also test for coupling faults, that is, faults such that an action on one memory cell could affect the contents of another memory cell. The set $\mathcal{F}_1$ is said to *B-cover* $\mathcal{F}_2$ if $L_B(\mathcal{F}_1) \subseteq L_B(\mathcal{F}_2)$ [Jü1], [Wo1]. When $\mathcal{F}_1$ B-covers $\mathcal{F}_2$ any test for faults in $\mathcal{F}_1$ is also a test for the faults in $\mathcal{F}_2$.

Theorem 4.1 [Jü1], [Wo1] *B-covering is a decidable pre-order.*

The following property indicates how the set of B-diagnosing words for a set of faults can be constructed from observers for parts of the set.

Theorem 4.2 [Jü1], [Wo1] *Let $\mathbb{F}$ be a family of fault models. Then $L_B\left(\bigcup_{\mathcal{F} \in \mathbb{F}} \mathcal{F}\right) = \bigcap_{\mathcal{F} \in \mathbb{F}} L_B(\mathcal{F})$.*

Detailed investigations of fault coverage for certain classes of single and multiple faults in random access memories are provided in [Jü1], [Wo1], and to some extent also in [Co1]. For instance, the set of single coupling faults, that is, of faults such that a write operation acting on one cell causes another cell to change its state B_{F}-covers the set of stuck-at faults.

5. Probabilistic Testing and the Complexity of Testing

In probabilistic testing a random or pseudo-random test sequence is applied to the circuit under test. For an *error threshold* ε with $0 < \varepsilon < 1$ and a fault model $\mathcal{F}$ one considers the minimal length $t_0(\varepsilon)$ of a random test sequence such that the probability of a fault in $\mathcal{F}$ remaining undetected is no greater than ε.

Various assumptions can be made about the stochastic source generating the test sequence. To keep the mathematical analysis manageable it should at least be a Markov source. So far, actually only memoryless sources have been considered for this purpose. With a memoryless source $\mathcal{S}$ as input source, a Markov chain $\mathcal{C}(\mathcal{F}, \mathcal{S}, B)$ can be constructed from the observer or its reduced version.[5] This Markov chain, called the *Markov diagnoser*, describes the probabilistic testing process. Let μ_t be the probability of reaching a B-decided state by step t. With ε as above, $t_0(\varepsilon)$ is the smallest t such that $\mu_t \geq 1 - \varepsilon$.

A mathematical analysis of the Markov chain $\mathcal{C}(\mathcal{F}, \mathcal{S}, B)$ has been carried out for random access memories only. For bounded faults, that is, faults involving only a bounded number of memory cells which is independent of the size of the memory, the following result is proved in [Da1].

Theorem 5.1 [Da1], [Co2] *For probabilistic testing for a bounded fault in an n-bit random access memory one has $t_0(\varepsilon) = O(n/\varepsilon)$ (under some mild conditions). However, there are bounded faults which require a deterministic test length of $\Omega(n \log n)$.*

This result shows that probabilistic testing leads to a significant reduction of the test length when large memories are to be tested and the fault model contains certain complicated faults. A detailed discussion of the merits of probabilistic testing is provided in [Da1].

Note that the lower bound of $\Omega(n \log n)$ mentioned in Theorem 5.1 is tight for the faults in question. The observer and automata derived from the observer turn out to be valuable tools for obtaining such bounds.

[4] The term of 'fault coverage' is used in several different—and not always consistent—ways [Wo1]. Unfortunately, the assumptions made in the literature are not always completely clear.

[5] This construction can, of course, also be carried out for a Markov source. The result is again a Markov chain.

6. Modelling Multiple Faults as Compositions of Single Faults

A *multiple fault* is defined as the simultaneous presence of more than one fault. Most of the technical literature on testing only considers *single faults*. This is mainly due to the fact that the mathematical treatment of multiple faults seems to be very difficult. In view of the constructions outlined so far, the task is as follows. Given the Mealy automata A_0, A_1, and A_2 for the correct circuit and two fault types, construct a Mealy automaton $A_1 \circ A_2$ that describes the fault type resulting from the presence of both A_1 and A_2. We refer to the structure arising from such compositions of faults as the *fault algebra*.

It turns out that the simple functional fault model is insufficient to define a meaningful composition of faults. Properties of the physical faults may need to be taken into account. Our new functional model—restricted to the case of binary memories so far—involves so-called multi-step automata. Moreover, a new composition of Mealy automata—of the usual and of the multi-step types—can be defined to model the simultaneous presence of the respective faults. A characterization of the semigroup generated by the stuck-at, transition, and coupling faults in binary random access memories is provided in [Br4].

One of the important issues arising from this work is to determine to which extent the multi-step automata model the physical situation well enough or whether, in some cases, relevant details are missed for modelling composite faults.

References

[Br1] J. A. Brzozowski, H. Jürgensen: Deterministic Diagnosis of Sequential Machines. [Gé1], 35–49.

[Br2] J. A. Brzozowski, H. Jürgensen: Probabilistic Diagnosis of Sequential Machines. [Gé1], 51–66.

[Br3] J. A. Brzozowski, H. Jürgensen: A Model for Sequential Machine Testing and Diagnosis. *Journal of Electronic Testing: Theory and Applications*, to appear.

[Br4] J. A. Brzozowski, H. Jürgensen: Component Automata and RAM Faults. Report 324, Department of Computer Science, The University of Western Ontario, 1992.

[Co1] B. F. Cockburn: *Fault Models and Tests for Coupling Faults in Random-Access Memories.* Ph. D. Thesis, University of Waterloo, 1990.

[Co2] B. F. Cockburn, J. A. Brzozowski: Near-Optimal Tests for Classes of Write-Triggered Coupling Faults in RAMs. *Journal of Electronic Testing: Theory and Applications* 3 (1992), 251–264, 1992.

[Da1] R. David, J. A. Brzozowski, H. Jürgensen: Testing for Bounded Faults in RAMs. Report CS-92-30, Department of Computer Science, University of Waterloo, 1992.

[Gé1] F. Gécseg, I. Peák (eds.): *Proc. 2nd Conf. on Automata, Languages, and Programming Systems, Salgótarján, May 23–26, 1988.* Department of Mathematics, Karl Marx University of Economics, Budapest, 1988-4, 1988.

[Gr1] S. Graf, M. Gössel: *Fehlererkennungsschaltungen.* Akademie-Verlag, Berlin, 1987.

[He1] F. C. Hennie: Fault-Detecting Experiments for Sequential Circuits. *Proc. 5th Annual Sympos. Switching Theory and Logical Design*, IEEE, 1964, 95–110.

[He2] F. C. Hennie: *Finite-State Models for Logical Machines.* John Wiley & Sons, New York, 1968.

[Jü1] H. Jürgensen, P. Wong: How to Prove Fault Coverage—or: Testing for the Most Difficult Fault. *Proc. 6th Workshop on Testing*, Montréal, 1992, 285–294.

[Mo1] E. F. Moore: Gedanken-Experiments on Sequential Machines. *Automata Studies*, 129–153. Princeton University Press, Princeton, N. J., 1956.

[Po1] J. F. Poage, E. J. McCluskey: Derivation of Optimum Test Sequences for Sequential Machines. *Proc. 5th Annual Sympos. Switching Theory and Logical Design*, IEEE, 1964, 121–132.

[Wo1] P. Wong: *A Language Theoretic Approach to Fault Coverage and Fault Testing.* M. Sc. Thesis, 1992. Report 314, Department of Computer Science, The University of Western Ontario, 1992.

[Ya1] M. Yannakakis, D. Lee: Testing Finite State Machines. *Proceedings, 23rd Annual ACM Symposium on Theory of Computing.* ACM, New York, 1991, 476–485.

Fraktale Dimension konstruktiv erzeugter Bilder

L. Staiger

Technische Universität Cottbus, Informatik
O-7500 Cottbus

Seit dem Erscheinen der Monographie "**Fractals, Form, Chance and Dimension**" von *B. Mandelbrot* im Jahre 1977 gewann die Untersuchung fraktaler Bilder auch im Rahmen der Informatik (z.B. der Computergraphik) an Interesse. Dabei standen insbesondere die Generierung fraktaler Bilder mit Mitteln der Automatentheorie, der Theorie der Formalen Sprachen und anderer Bereiche der Theoretischen Informatik im Vordergrund.

Als geeignete Größen zur Messung des fraktalen Charakters bzw. der Textur eines Bildes haben sich die fraktale oder *Hausdorff* - Dimension und das zugehörige *(Hausdorff -)* Maß erwiesen. Aus der Fraktalen Geometrie bzw. Geometrischen Maßtheorie ist bekannt, daß diese Größen selbst für einfache Mengen schwierig zu bestimmen sind

Der Vortrag zeigt nun, daß Mittel der Theoretischen Informatik, ähnlich denen zur Generierung fraktaler Bilder, ebenfalls geeignet sind, *Hausdorff* - Dimension und - Maß dieser Bilder zu berechnen.

Als konstruktiv erzeugte Bilder sehen wir hierbei mittels endlicher Quellen (kantenmarkierter Graphen, Automaten) durch die r–adische Entwicklung ihrer Punkte dargestellte abgeschlossene Teilmengen des Einheitsquadrats an

Für derartige Bilder werden Algorithmen entwickelt, die es gestatten, *Hausdorff* - Dimension und *Hausdorff* - Maß exakt zu bestimmen. Es erweist sich darüberhinaus, daß Dimension und Maß in direktem Zusammenhang zur Dichte bzw. zum Schwärzungsgrad des Bildes stehen, was auch durch experimentelle Untersuchungen illustriert wird.

MRFS-Fraktale aus dem Blickwinkel der regulären ω-Sprachen

H. Fernau

Lehrstuhl Informatik für Ingenieure und Naturwissenschaftler
Universität Karlsruhe (TH)
Am Fasanengarten 5, Postfach 6980, W-7500 Karlsruhe 1 (Germany)

(Unendliche) Iterierte Funktionensysteme ((I)IFS) $F = (F(1), F(2), \ldots)$ erlauben (nach dem Banachschen Fixpunktsatz) eine rekursive Beschreibung von Fraktalen (gewissen abgeschlossenen und beschränkten Mengen) als Fixpunkten von Mengengleichungen der Form (sei $\Sigma \subseteq \mathbb{N}$)

$$A_F = \mathrm{cl} \bigcup_{i \in \Sigma} \mathrm{F(i)(A_F)}.$$

Dabei seien die $F(i)$ kontrahierende Ähnlichkeitsabbildungen auf einem vollständigen beschränkten metrischen Raum (X, ρ) [Bar88, Ban89]. Die Ähnlichkeitsfaktoren seien

$$R_F = (R_F(1), R_F(2), \ldots).$$

F heiße nicht kollabierend, wenn keiner der $R_F(i)$ verschwindet.
(Gewisse) Punkte x in A_F haben (wenigstens) ein ω-Wort w aus Σ^ω als Adresse. Die Lipschitz-stetige Modellabbildung ϕ_F ist gegeben durch

$$x = \phi_F(w) \triangleq \lim_{m \to \infty} F(w(1)) \circ F(w(2)) \circ \cdots \circ F(w(m))(y)$$

für beliebige $y \in X$. Es gilt $A_F = \mathrm{cl}\phi_F(\Sigma^\omega)$. Dabei betrachten wir Σ^ω mit der Metrik ρ_{R_F}, die gegeben ist durch die Vorschrift

$$(\forall x, y \in \Sigma^\omega)(\rho_{R_F}(x, y) \triangleq \inf_{m \in \mathbb{N}} \{\prod_{i=1}^{m} R_F(x(i)) \mid (\forall i = 1, \ldots, m)(x(i) = y(i))\}).$$

$(\Sigma^\omega, \rho_{R_F})$ ist zum Baireschen Raum homöomorph. $(\Sigma^\omega, \rho_{R_F})$ ist genau dann kompakt, wenn Σ endlich ist.
Häufig lassen sich (I)IFS F_L auch durch eine Sprache L über einem endlichen Alphabet $\Sigma_n = \{1, \ldots, n\}$ von kontrahierenden Ähnlichkeitsabbildungen kennzeichnen. Der Fixpunkt ist dann das Lipschitz-stetige Bild der Adhärenz von L^*, $A_{F_L} = \phi_F(\mathrm{cl}L^\omega)$. Die Ähnlichkeitsfaktoren $R(1), \ldots, R(n) \in (0, 1)$ der Abbildungen $1, \ldots, n$ lassen sich zu einem Halbgruppenmorphismus $R_* : (\Sigma_n^+, \cdot) \to ((0, 1), \cdot)$ fortsetzen. Sie definieren über $\Phi(s) \triangleq \sum_{w \in L}(R_*(w))^s = \sum_n (R_{F_L}(n))^s$ die Selbstähnlichkeitsdimension

$$s_{F_L} \triangleq \sup\{s \in [0, \infty) \mid \Phi(s) \geq 1\}.$$

Diese ist stets eine obere Schranke für die Hausdorff-Dimension von $\phi(L^\omega)$.
L bzw. F_L heißen endlich-summierbar, falls folgendes gilt: Es gibt eine natürliche Zahl m_0 und eine nicht negative reelle Zahl s_{m_0}, so daß

- $\Phi(s_{m_0}) < \infty$
- Die Teilsumme $\sum_{n=1}^{m_0}(R_{F_L}(n))^{s_{m_0}}$ ist gleich 1.

Für endlich-summierbare F gilt $\Phi(s_F) = 1$. Ist $R(1) = \cdots = R(n)$ konstant und L regulär, so ist F_L endlich-summierbar.
Offenes Problem: Gibt es nicht konstante R_F und reguläre Präfixcodes L, so daß F_L nicht endlich-summierbar ist?
Sind n Ähnlichkeitsabbildungen auf dem Euklidischen Raum (X, ρ_E) gegeben, die eine Moransche Bedingung [Mor46] erfüllen, und ist $L \subseteq \Sigma_n^+$ ein (endlicher oder endlich-summierbarer) Präfixcode, so wird diese Schranke auch angenommen
Bei 'mutual recursive' Funktionensystemen MRFS G [MW88, ČD90] ist ein wechselseitig rekursives System von k Mengengleichungen der Form

$$S_i^m \triangleq \bigcup_{(j,i) \in D(f)} F(j)(S_{f(j,i)}^{m-1}),$$

durch eine partielle Funktion $\Sigma_n \times \Sigma_k \to \Sigma_k$ mit k abgeschlossenen beschränkten Startmengen $S_i^0 \subseteq X$ gegeben. Zusammen mit einer Endzustandsmenge $E \subseteq \Sigma_k$ definiert G einen abgeschlossenen, beschränkten Attraktor (unabhängig von den Startmengen S_i^0), der sich als

$$A = \lim_{m \to \infty} (\bigcup_{i \in E} S_i^m)$$

beschreiben läßt. Solchen Rekursionsvorschriften entsprechen auf Adreßraumebene rechtslineare Gleichungssysteme mit den k Zeilen

$$X_i = jX_{f(j,i)}.$$

Es ist bekannt, daß ein rechtslineares Gleichungssystem genau einen Lösungsvektor

$$\mathbf{L} = (\mathbf{L_1}, \ldots, \mathbf{L_k})$$

von nicht leeren aber abgeschlossenen ω-Sprachen besitzt [LS77]. Damit ist $L = \bigcup_{i \in E} L_i$ der Attraktor des syntaktischen MRFS. Als endliche Vereinigung abgeschlossener Teilmengen eines kompakten Raumes ist L kompakt.
Umgekehrt ist es möglich, diesen Existenzsatz über ω-Sprachen aus dem Existenz und Eindeutigkeitssatz für MRFS-Attraktoren abzuleiten.
Die Abgeschlossenheit regulärer abgeschlossener ω-Sprachen gegen (endliche) Vereinigung zeigt, daß man sich bei MRFS auf einelementige Endzustandsmengen einschränken kann. Daher lassen sich MRFS durch ein IFS F und eine abgeschlossene reguläre ω-Sprache L spezifizieren.
Diese formalsprachliche Abgeschlossenheitsaussage zeigt auch, daß die Klasse der MRFS-Fraktale gegenüber endlicher Vereinigungsbildung abgeschlossen ist, was für spracherzeugte nicht kollabierende IFS und IIFS nicht gilt.
Durch MRFS beschriebene Fraktale kann man als Lipschitz-stetige Bilder abgeschlossener regulärer ω-Sprachen begreifen. Nach einem Satz von Choueka [Cho74] läßt sich eine (abgeschlossene) reguläre Sprache $L \subseteq \Sigma_n^\omega$ darstellen als

$$L = \bigcup_{i \in \Sigma_m} W_i(V_i)^\omega,$$

wobei die W_i, V_i reguläre Sprachen sind und die V_i darüberhinaus präfixfrei sind.
Es sei $F = (F(1), \ldots, F(n))$ ein IFS auf dem Euklidischen Raum (X, ρ_E), das einer Moranschen Bedingung genüge. Für den Attraktor A des durch F und L spezifizierten MRFS ergibt sich aus dieser Darstellung:

$$\begin{aligned} A &= \phi_F(L) \\ &= \phi_F(\bigcup_{i \in \Sigma_m} W_i(V_i)^\omega) \\ &= \bigcup_{i \in \Sigma_m} \bigcup_{w \in W_i} \phi_F(w)(\phi_F(V_i^\omega)) \end{aligned}$$

Es seien t_i die Selbstähnlichkeitsdimensionen der durch F und V_i spezifizierten (I)IFS und t das Maximum dieser t_i. Ist ein V_i mit $t_i = t$ endlich(-summierbar), so ist t die Hausdorff-Dimension von A.
Diese Idee funktioniert auch bei anderen metrischen Räumen.
Die erläuterte Betrachtungsweise liefert daher ein neues Verfahren zur Bestimmung der Hausdorff-Dimension von MRFS-Fraktalen. Die in der Literatur übliche Methode funktioniert wie folgt. Man ordnet dem GLS $X_i = jX_{f(j,i)}$ die nicht negative Matrix $P(s)$ mit

$$(P(s))_{ij} \triangleq \begin{cases} 0 & (j,i) \notin D(f) \\ (R_F(j))^s & \text{, sonst} \end{cases}$$

zu. Ist $P(s)$ unzerlegbar (ansonsten betrachtet man die Zusammenhangskomponenten einzeln), so besitzt diese nach dem Satz von Perron-Frobenius in Abhängigkeit von s einen reellen Eigenwert $\Phi(s)$ größten Betrages. Dann gibt ein eindeutig bestimmtes s_0 mit $\Phi(s_0) = 1$. Es sei L eine Komponente des eindeutig bestimmten Lösungsvektors aus lauter abgeschlossenen regulären ω-Sprachen. Genügt F einer Moranschen Bedingung, so ist s_0 die Hausdorff-Dimension des Attraktors A des durch F und L spezifizierten MRFS.
Offene Frage: Kann man direkt (matrizentheoretisch) einsehen, weshalb bzw. wann die beiden Verfahren denselben Schätzwert für die Hausdorff-Dimension eines MRFS-Fraktals liefern?
Als geometrisches Beispiel eines MRFS-Fraktals betrachten wir Bild 1. Sind $1, 2, 3$ die üblichen drei das Sierpiński-Dreieck definierenden Abbildungen und $4, 5, 6, 7$ diejenigen der Koch-Kurve, so beschreibt $L = \mathrm{adh}(\{1,2,3\}^+\{4,5,6,7\}^*)$ das Fraktal.

Barnsley betrachtet in seinem Buch [Bar88] noch die wie folgt definierte Abart von IFS.
Es sei (X, ρ) ein vollständiger metrischer Raum. $F_C = (F, C)$ heißt IFS mit Kondensation, falls $F = (F(1), \ldots, F(n))$ ein IFS ist und C eine nicht leere abgeschlossene und beschränkte Teilmenge von X. Wir können F_C als kontrahierende Abbildung von $(\mathcal{CB}(X, \rho), \rho_H)$ auf sich ansehen, indem wir

$$F_C(B) = \bigcup_{i=1}^{n} F(i)(B) \cup C$$

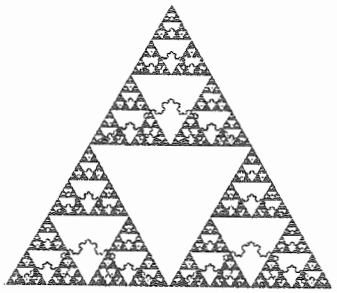


Abbildung 1: Mathematisches Mimikry

setzen. Nach dem Banachschen Fixpunktsatz definiert F_C so einen eindeutigen Fixpunkt A_{F_C}. Bezüglich der Hausdorff-Dimension gilt

$$\dim_H(A_{F_C}) = \max\{\dim_H(A_F), \dim_H(C)\}.$$

$F = (F(1), \ldots, F(n))$ heißt IFS n-ter Stufe oder kurz IFS^n, falls alle $F(i)$ IFS auf X sind. Der Attraktor eines IFS erster Stufe $F = (F(1))$ sei gleich dem Attraktor von dem IFS $F(1)$. Rekursiv definieren wir nun $A_{(F(1),\ldots F(n))} = A_{F(1)_C}$ mit $C = A_{(F(2),\ldots,F(n))}$.
Die Hausdorff-Dimension eines von einem IFS^n $F = (F(1), \ldots, F(n))$ beschriebenen Fraktals ist gleich dem Maximum aller Hausdorff-Dimensionen $\dim_H A_{F(1)}$ bis $\dim_H A_{F(n)}$.
Ist $\Sigma_{F(i)}$ das zur Spezifikation von $F(i)$ benötigte Teilalphabet und ist Σ das Abbildungsalphabet des zugrundeliegenden IFS G, so gilt

$$A_{(F(1),\ldots,F(n))} = \phi_G(\mathrm{adh}(\Sigma^*_{F(1)} \cdots \Sigma^*_{F(n)})).$$

Da $\{1\}^\omega \cup \{2\}^\omega$ nicht durch ein IFS n-ter Stufe beschreibbar ist, umfaßt die Klasse der durch MRFS beschreibbaren Fraktale (ω-Sprachen) die der durch mehrstufige IFS beschreibbaren echt.
Offene Frage: Die Klasse der durch IFS^n beschreibbaren kompakten Mengen umfaßt die Klasse der durch IFS^{n-1} beschreibbaren kompakten Mengen. (Wann) Ist diese Umfassung echt?

Literatur

[Ban89] C. Bandt, *Self-similar sets. I. Topological Markov chains and mixed self-similar sets*, Mathematische Nachrichten **142** (1989), 107–123.

[Bar88] M.F. Barnsley, *Fractals Everywhere*, Boston: Academic Press, 1988.

[ČD90] K. Čulik, II and S. Dube, *Methods for generating deterministic fractals and image processing*, LNCS: 464; IMYCS, 1990, Seiten 2–28.

[Cho74] Y. Choueka, *Theories of automata on ω-tapes: A simplified approach*, Journal of Computer and System Sciences **8** (1974), 117–141.

[LS77] R. Lindner and L. Staiger, *Algebraische Codierungstheorie; Theorie der sequentiellen Codierungen*, Elektronisches Rechnen und Regeln, Band 11, Berlin: Akademie-Verlag, 1977.

[Mor46] P.A.P. Moran, *Additive functions of intervals and Hausdorff measure*, Proceedings of the Cambridge Philosophical Society **42** (1946), 15–23.

[MW88] R.D. Mauldin and S.C. Williams, *Hausdorff dimension in graph directed constructions*, Transactions of the American Mathematical Society **309** (1988), Nr. 2, 811–829.

Wann ist eine funktionale Baumübersetzung deterministisch?

H. Seidl

Fachbereich Informatik, Universität des Saarlandes
Im Stadtwald 15, 6600 Saarbrücken 11

A (finite–state bottom–up) tree transducer is a finite state device. While traversing an input tree it produces an output tree in a bottom–up fashion. It is called functional iff there is at most one output tree for every input.

In practice, the concept of tree transducers is used, e.g., by tools for generating code generators from formal descriptions of target machine languages. Given a (regular) tree grammar which specifies the semantics of machine instructions in terms of operators of the intermediate representation language, a (typically non–deterministic) tree transducer is generated. The accepting computations describe all sequences of target machine code possibly generated for a given piece of program. Taking into account that different instructions may have different costs (e.g., need more or less processor cycles) a functional transducer is constructed which produces a cheapest instruction sequence.

Our main concern in this talk is the implementation of such functional transducers. Usually, this is done in two passes. The first one traverses the input tree to compute an overview over all possible computations from which the second one constructs an accepting computation and produces the output for it. We would like to avoid this second pass and produce output immediately. This can be done provided the corresponding tree transducer is deterministic. However, simple examples show that it is not always possible to construct an equivalent deterministic transducer. Therefore, the natural question arises: *When can the transduction of a functional transducer be realized by a deterministic one?*

Also, one would like to produce output only for subcomputations which finally is part of the resulting output. This can be guaranteed provided the tree transducer is *reduced deterministic*. Again, it is not always possible to construct even for a deterministic transducer an equivalent reduced deterministic transducer Therefore, we also have to deal with the question: *When can the transduction of a functional transducer be realized by a reduced deterministic one?*

The talk answers these two questions. Besides a structural characterization, we give a decision procedure to determine whether or not the transduction of a functional transducer can be realized by a deterministic (resp. reduced deterministic) transducer. In case this is possible we exhibit a general construction to build this transducer.

Die Komplexität linearer LL(1) und LR(1) Sprachen

M. Holzer, K.-J. Lange

Technische Universität München, Institut für Informatik
Arcisstr. 21, 8000 München

LL(1) und LR(1) Grammatiken wurden von Knuth [2] und Lewis, Stearns [3] in den Jahren 1965 und 1968 eingeführt. In [4], zeigte Sudborough, daß kontextfreie LL(1) und LR(1) Sprachen (Sprachen die durch LL(1) oder LR(1) Grammatiken erzeugt werden) logspace-vollständig für die Klasse $LOG(\mathcal{DCFL})$, der deterministischen Variante von $LOG(\mathcal{CFL})$, sind.

Da lineare Sprachen $\mathcal{NL}$-vollständig sind und Ibarra, Jiang und Ravikumar [1] zeigten, daß eine Unterklasse der linearen Sprachen, von den Autoren als deterministisch linear bezeichnet, $\mathcal{NC}^1$-vollständig ist, stellt sich die Frage, wie sich lineare LL(1) und LR(1) Sprachen im Gefüge logarithmisch platzbeschränkter Komplexitätsklassen verhalten.

Es wird gezeigt, daß lineare LL(1) Sprachen $\mathcal{NC}^1$-vollständig bzgl. $\mathcal{DLOGTIME}$ Reduktion sind, und das die Klasse der λ-freien linearen LL(1) Sprachen (Sprachen die von linearen LL(1) Grammatiken ohne λ-Produktionen erzeugt werden) mit der Klasse der deterministischen linearen Sprachen von Ibarra, Jiang und Ravikumar [1] übereinstimmt.

Für lineare LR(1) Sprachen konnte hingegen die $\mathcal{L}$-Vollständigkeit bzgl. $\mathcal{DLOGTIME}$ Reduktion nachgewiesen werden. Darüber hinaus wird gezeigt, daß die Klasse der linearen LR(1) Sprachen mit der Klasse der Sprachen die durch determinitische one-turn Kellerautomaten erkannt werden übereinstimmt.

Literatur

[1] O. H. Ibarra, T. Jiang, and B. Ravikumar. Some subclasses of context-free languages in $\mathcal{NC}^1$. *Information Processing Letters*, 29:111–117, October 1988.

[2] D. E. Knuth. On the translation of languages from left to right. *Information and Control*, 8:607–639, 1965.

[3] P. Lewis and R. Stearns. Syntax-directed transduction. *Journal of the Association for Computing Machinery*, 15:465–488, 1968.

[4] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the Association for Computing Machinery*, 25:405–414, 1978.

Eindeutigkeit von Dokumentengrammatiken in SGML

Anne Brüggemann-Klein

Universität Freiburg, Institut für Informatik
Rheinstr. 10–12, 7800 Freiburg

Die Standard Generalized Markup Language (SGML) definiert Dokumententypen durch kontextfreie Grammatiken in einer erweiterten Backus-Naur-Form [ISO86, Gol90]. Die rechte Seite einer Regel heißt *Content model* und ist ein erweiterter regulärer Ausdruck, der zusätzlich zu den üblichen sprachlichen Mitteln die drei Operatoren &, ? und $^+$ verwenden kann; dabei gilt für die Sprachen $L(F\&G) = L(FG+GF)$, $L(F?) = L(F+\epsilon)$ und $L(F^+) = L(FF^*)$. Der SGML-Standard schreibt vor, daß Content models eindeutig sein müssen in dem Sinne daß "an element [ein Symbol] ... that occurs in the document instance [Wort] must be able to satisfy only one primitive content token [Vorkommen des Symbols in dem Content model] without looking ahead in the document instance." Damit ist Eindeutigkeit im Sinne von SGML eine Ein-Symbol-Lookahead-Version von Eindeutigkeit wie sie von Book et al. [BEGO71] definiert wurde.

Wir zeigen zwei Resultate, die sowohl von theoretischem Interesse sind als auch von praktischer Bedeutung für die Implementierung von SGML-Systemen: Wir können erstens in linearer Zeit entscheiden, ob ein gegebenes Content model eindeutig ist. Und zweitens können wir entscheiden, ob eine gegebene reguläre Sprache durch ein eindeutiges Content model dargestellt werden kann; ist die Sprache durch einen determinierten Automaten gegeben, so ist die Laufzeit quadratisch in seiner Größe.

Ähnliche Resultate sind bereits für reguläre Ausdrücke anstelle von Content models erzielt worden [BW92, Bru92a, Bru92b]. Das Problem, welche Sprachen durch eindeutige Content models dargestellt werden können, wird durch Umwandlung eindeutiger Content models in äquivalente eindeutige reguläre Ausdrücke gelöst. Der &-Operator bewirkt jedoch, daß der kleinste eindeutige reguläre Ausdruck, der zu einem gegebenen Content model äquivalent ist, schlimmstenfalls exponentiell ist in der Größe des Content models. Deshalb erfodern Content models eine neue Technik, um einen effizienten Algorithmus zur Entscheidbarkeit von Eindeutigkeit zu entwickeln.

Die hier vorgestellten Algorithmen sind teilweise von Clark in seinem SGML-System *sgmls* implementiert [Cla92] und finden zur Zeit Eingang in den Revisionsprozess von SGML.

Literatur

[BEGO71] R. Book, S. Even, S. Greibach und G. Ott. Ambiguity in graphs and

expressions. *IEEE Transactions on Computers*, C-20(2):149–153, Februar 1971.

[Bru92a] A. Brüggemann-Klein. Regular expressions into finite automata. In I. Simon, Herausgeber, *Latin '92*, S. 87–98, Springer-Verlag, Berlin, 1992. Lecture Notes in Computer Science 583.

[Bru92b] A. Brüggemann-Klein. Regular expressions into finite automata. 1992. Erscheint in Theoretical Computer Science.

[BW92] A. Brüggemann-Klein und D. Wood. Deterministic regular languages. In A. Finkel and M. Jantzen, Herausgeber, *STACS 92*, S. 173–184, Springer-Verlag, Berlin, 1992. Lecture Notes in Computer Science 577.

[Cla92] J. Clark. 1992. Quellcode für *sgmls*. Erhältlich von ftp.uu.net oder sgml1.ex.ac.uk durch *ftp*.

[Gol90] C. F. Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1990.

[ISO86] ISO 8879. Information processing—Text and office systems—Standard generalized markup language (SGML). Oktober 1986. International Organization for Standardization.

Formale Sprachen von Output Strings: Eine allgemeine Methode zur Definition von Komplexitätsklassen

C. Lautemann und Th. Schwentick

Universität Mainz, FB 17
Saarstr. 21, 6500 Mainz

Als Output String einer nichtdeterministischen Turing-Maschine M bezeichnen wir die 0-1-Folge der Ergebnisse aller Berechnungen von M bei Eingabe x. Über die Struktur von Output Strings lassen sich Komplexitätsklassen definieren: z.B. läßt sich NP durch die Menge aller 0-1 Strings, die mindestens eine 1 enthalten charakterisieren. Es ergeben sich auf diese Art einige weitere Korrespondenzen zwischen Komplexitätsklassen und Formalen Sprachen: so zwischen der Polynomiellen Hierarchie und den stern-freien Regulären Sprachen, und zwischen ModPH und den Regulären Sprachen deren Syntaktischer Monoid nur auflösbare Gruppen enthält.

Wir sind gespannt auf Anregungen von Experten auf dem Gebiet der Formalen Sprachen.

Neuere Entwicklungen aus dem Bereich algebraischer Graphgrammatiken

M. Korff

TU Berlin
Franklinstr. 28/29, 1000 Berlin 10
martin@cs.tu-berlin.de

Der algebraische Ansatz für Graphgrammatiken basiert auf der aus Berlin stammenden Dopel-Pushout(DPO)-Variante. Detailierte theoretische Untersuchungen darüber lassen sich in [Ehr79] finden. 1990 wurden diese Ergebnisse in der Arbeit [Lœw90] im Rahmen des sogenannten Single-Pushout(SPO)-Ansatzes verallgemeinert.

Der algebraische Ansatz zeichnet sich vor allem durch die Kombination einer klaren operationalen Vorstellung des zugrundeliegenden Graphtransformationsbegriffes mit einer Anzahl von komplexen theoretischen Konsequenzen aus. Dies beruht wesentlich darauf, daß die grundlegenden Definitionen kategoriell formuliert sind, was sehr abstrakte und damit kompakte und sichere Beweise erlaubt. Neben rein theoretisch orientierten, existiert aber auch eine Anzahl anwendungs-motivierter Arbeiten, deren Ziel insbesondere der Einsatz von Graphgrammatiken zur Spezifikation und Programmierung von Software-Problemen ist. (vgl. [EKL91, EL92])

Der Single-Pushout-Ansatz

Wurde der DPO-Ansatz anfangs nur für 'klassische', also gerichtete Graphen mit Mehrfachkanten formuliert, läßt der SPO-Ansatz Algebren beliebiger algebraischer Signaturen zu, sofern diese nur monadischen Operationssymbole enthalten. Die entsprechende Kategorie von Algebren und *partiellen* Homomorphismen, also Homomorphismen, die nur auf einer Unteralgebra ihrer Domain definiert sind, liefert die notwendigen Begriffe von Graph-Struktur bzw. partiellem Graphmorphismus. Sowohl die o.g. Graphen, als auch Hypergraphen, bei denen eine (Hyper-)Kante mit mehr als 2 Knoten verbunden sein kann, lassen sich als Graph-Strukturen auffassen. Darüberhinaus sind weitere Graph-Strukturen, z.B. mit Kanten höherer Ordnung, also z.B. eine Kante zwischen zwei Kanten u.ä. modellierbar. Die Grundidee des auf partiellen Morphismen beruhenden Ansatzes ist es, eine Graph-Transformation als eine Beziehung t zwischen dem Originalgraphen O und dem abgeleiteten Graphen A zu sehen, bei dem drei Kategorien von Knoten bzw. Kanten unterschieden werden: *erhaltene*, die in beiden Graphen O und A vorkommen, *gelöschte*, die nur in O vorkommen und *zugefügte*, die nur in A vorkommen. t projeziert nun einfach die zu erhaltenen

Objekte im alten Kontext von O auf die im neuen Kontext von A, d.h. t ist ein partieller Graphmorphismus $t : O \rightarrow A$. Eine Regel r ist nichts anderes als eine Muster-Transformation, also ebenfalls ein partieller Graphmorhismus $r : L \rightarrow R$. r ist auf einen Graphen G anwendbar, falls sich L vollständig in G finden läßt, was formal die Existenz eines totalen Graphmorphismuses $a : L \rightarrow G$, genannt Ansatz, bedeutet. Die Anwendung von r auf G an a ergibt dann die Ableitung, die als (Single-)Pushout von r und a in der unterliegenden Kategorie von partiellen Graphmoprhismen definiert ist. Erfüllt der Ansatz die aus dem DPO-Ansatz bekannten Gluing Condition,

$$\begin{tikzcd} L \arrow[r, "r"] \arrow[d, "a"'] \arrow[dr, phantom, "\text{Pushout}"] & R \arrow[d, "a_r"] \\ G \arrow[r, "r_a"'] & H \end{tikzcd}$$

Abbildung 2: Ableiten ist Verkleben ist (Single-)Pushout

führen SPO- und DPO-Semantik zur gleichen Transformation. Der abgeleitete Graph läßt sich dann folgendermaßen erhalten: Aus G werden alle Objekte gelöscht, die L mittels a als gelöscht spezifiziert. Das Ergebnis G' wird mit R zu H verklebt, d.h. $H = (G' + R)_{/dom(r)}$. Aufgrund der Tatsache, daß die Kateorie mit partiellen Graphmorphismen co-vollständig ist, existieren aber auch Ableitungen für Ansätze, die die Gluing Condition nicht erfüllen. Wird ein Knoten gelöscht ohne dabei explizit die anhängenden Kanten zu löschen, so werden diese implizit gelöscht. Schafft der Ansatz Konflikte, indem er für ein Objekt in G sowohl Löschen als auch Erhalten spezifiziert, dominiert die Löschanweisung.

Theoretische Ergebnisse

Der SPO-Ansatz bedingt, daß die operationale Charakterisierung der Pushout-Konstruktion i.A. und damit vielleicht auch der der Ableitung relativ komplexer wird. Quasi als Belohnung ergibt sich aber, daß Beweise vergleichbarer Sätze noch abstrakter und weiter kompaktifiziert werden. Allerdings gelten nicht alle Theoreme in gleicher Weise, wenn man über das im DPO-Ansatz mögliche hinausgeht. U.a. existieren folgende Ergebnisse:

- Parallel unabhängige Ableitungen sind lokal konfluent.
- Ableitungssequenzen mit unabhängigen Regeln können in einem einzigem Ableitungsschritt mit Parallelregeln simuliert werden.
- Eindeutig löschende Ableitungen mit der Parallelregel können sequentialisiert werden.
- Ableitungssequenzen können in größerem Kontext eingebettet werden.
- Ableitungen mit abgeleiteten Regeln können durch Ableitungssequenzen mit den Originalregeln simuliert werden.

- Maximal parallele Ableitungssequenzen sind eindeutig (kanonisch).
- Synchronisierte Ableitungen können in Unterregel und Restregelableitungen zerlegt werden.
- Verteilte Ableitungen sind äquivalent mit synchronisierten Ableitungen auf entsprechenden globalen Zuständen.

Eine Normalform für nicht-kontextfreie Graphgrammatiken

Normalformensätze sind in der klassischen Theorie formaler Sprachen von einigem Interesse, ermöglichen sie es doch oftmals Beweise einfach und ohne grosse Fallunterscheidungen zu führen. Betrachten wir Ableitungen im SPO-Ansatz ohne implizite Kantenlöschung, so läßt sich eine Normalform für nicht-kontextfreie Hypergraphgrammatiken angeben, bei der alle Regeln auf der linken Seite maximal zwei Hyperkanten aufweisen. Man vergleiche dazu den Begriff *kontextfreier* Hyperkantenersetzung wie er in [Hab89]) zu finden ist. Der Schritt von 1 Hyperkante zu 2 erlaubten Hyperkanten stellt also bereits die volle Mächtigkeit einer beliebigen Hypergraphgrammatik zur Verfügung. Diese Idee wollen wir im Folgenden etwas detaillierter skizzieren.

Sig	$HG =$	sorts	$V, L, E_1, E_2, E_3, \ldots$
		opns	$c_1, c_2, \ldots c_n : E_n \to V$ für $n \in N$
			$l : E_n \to L$ für $n \in N$

Die algebraische Signatur HG induziert die Kategorie $\boldsymbol{HG^P}$. Ihre Objekte sind *Hypergraphen* wobei V eine Knotenmenge, L ein Kantenlabel-alphabet und $E_1, E_2, \ldots E_n \ldots$ Mengen von Hyperkanten fordert, die $1, 2, \ldots n \ldots$ Tentakel haben, über die die jeweilige Hyperkante mit mit ihren n Knoten verbunden ist. *Morphismen* sind labelerhaltende, partielle HG-Homomorphimsmen, d.h. der Definitionsbereich für $f : A \to B$ muß ein Teilgraph von A sein. *Regeln* sind partielle $\boldsymbol{HG^P}$morphismen. *Ableitungen* folgen der SPO-Semantik, wobei als *Ansätze* nur solche totalen Graphmorphismen $a : L \to G \in \boldsymbol{HG^P}$, für die G eingeschränkt auf Elemente aus $G - a(L)$ wieder ein Hypergraph ist.[1] Eine *Grammatik* $HGG = (\Sigma, T, Rg, S)$ besteht aus dem globalen Alphabet Σ, dem Terminalalphabet $T \subseteq \Sigma$, einer Regelmenge Rg und einem Hypergraphen S. Die *Sprache* $L(HGG)$ ist dann definiert als die Menge aller aus S mit Regeln aus Rg ableitbaren Hypergraphen G, die ausschließlich terminal markiert sind. Grammatiken sind *äquivalent*, wenn sie die gleiche Sprache erzeugen.

Satz *Für eine beliebige Grammatik $HGG = (\Sigma, T, Rg, S)$ existiert eine äquivalente Grammatik $HGG' = (\Sigma', T, Rg', S)$, wobei Rg' nur Regeln enthält, die links max. 2 Hyperkanten besitzen.*

[1] Das heißt wir betrachten beliebige Ansätze, die zwar die Dangling, nicht aber die Identification-Condition erfüllen müssen (vgl. [?]).

BEWEISSKIZZE: Fügen wir zu den Regeln Rg noch alle minimal abgeleiteten Regeln M hinzu (vergl. [Lœw90]), so ist $L_{inj}(\Sigma, T, Rg \cup M, S) = L(\Sigma, T, Rg \cup M, S) = L(HGG)$; dabei ist $L_{inj}(HGG'')$ die Sprache, deren Ableitungen auf ausschließlich injektiven Ansätzen basieren. Bei injektiven Ansätzen aber ändert sich die Sprache nicht, wenn außerdem alle Hyperkanten aus dem Definitionsbereich der Regeln $Rg \cup M$ genommen werden. Die enstandene Regelmenge sei Rg''. Für jede Regel $r'' : L'' \to R'' \in Rg''$ enthält Rg' nun, falls L'' mindestens 3 Hyperkanten aufweist, genau eine *simulierende Regel* und eine Menge von *Blockregeln*, anderfalls r'' selbst. Sei $h_1, h_2, \ldots, h_e$ eine Aufzählung aller Hyperkanten aus L. Dann existieren $e - 1$ Blockregeln $b_i : B \to C$ $1 \leq i < e$ Jede löscht 2 Hyperkanten in B und faßt sie zu einer einzigen in C zusammen, deren Tentakel genau mit denen auf der in B überein stimmen. Es wird eine frische nicht-terminale Markierung verwandt. Alle incidenten Knoten in B werden identisch abgebildet. Die erste Blockregel faßt die Kanten h_1 und h_2 zusammen. Die i-te verschmilzt die durch die $(i-1)$te Blockregel zusammengefaßte und h_{i+1}.

Die simulierende Regel $s(r'')$ enthält auf der linken Seite genau die Hyperkante, die sich aus der e-ten Blockregel ergibt. Das bedeutet, daß hier auch die gleichen Knoten vorliegen, die wie durch r'' abgebildet werden. Die rechte Seite ist R''.

Für jede Ableitung (mit injektivem Ansatz) $G \overset{r''}{\Longrightarrow} H$ existiert nun nach Konstruktion eine Ableitungssequenz $G \overset{b_1}{\Longrightarrow} G_1 \overset{b_2}{\Longrightarrow} G_2 \ldots G_e \overset{s(r'')}{\Longrightarrow} H$. Andererseits gilt für jede Sequenz $S \Longrightarrow H$ mit $H \in L(HGG')$, in der mindestens eine der bzgl $r'' \in Rg''$ definierten Regeln benutzt wurde aufgrund der frischen nicht-terminalen Markierungen, daß alle entsprechenden Blockregeln und abschließend die simulierende Regel in ihrer definierten Reihenfolge benutzt wurden. Allerdings kann die Sequenz durch andere als diese Regeln unterbrochen sein. Jene lassen sich aber wegen sequentieller Unabhängigkeit bis vor die erste Blockregel vorziehen (vgl. [Lœw90]). Aus der so kompaktifizierten Ableitungssequenz $S \overset{*}{\Longrightarrow} G \overset{b_1}{\Longrightarrow} G_1 \overset{b_2}{\Longrightarrow} G_2 \ldots G_e \overset{s(r'')}{\Longrightarrow} H' \overset{*}{\Longrightarrow} H$. läßt sich aus den entsprechenden Ansätzen $m_1, \ldots, m_e, m'$ ein Ansatz m'' für die Regel r'' konstruieren, sodaß $S \overset{*}{\Longrightarrow} G \overset{r''}{\Longrightarrow} H' \overset{*}{\Longrightarrow} H$. Dies gilt auch dann, wenn der Ansatz von b_1 nicht-injektiv (löschend) ist. □

Literatur

[Ehr79] Hartmut Ehrig, *Introduction to the algebraic Theory of graph grammars*, 1st Graph Grammar Workshop, Lecture Notes in Computer Science 73, Springer, 1979, pp. 1–69.

[EKL91] H. Ehrig, M. Korff, and M. Lœwe, *Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts*, 4th Int. Workshop on Graph Grammars and Their Application to Computer Science, Lecture Notes in Computer Science 532 (H. Ehrig, H.-J. Kreowski, and G. Rozenberg, eds.), Springer, 1991, pp. 24–37.

[EL92] H. Ehrig and M. Lœwe, *Computing by graph transformation (COMPUGRAPH): a survey*, Tech. Report 92/14, Technical University of Berlin, Department of Computer Science, 1992.

[Hab89] A. Habel, *Hyperedge replacement: Grammars and languages*, Ph.D. thesis, University of Bremen, 1989.

[Lœw90] M. Lœwe, *Extended algebraic graph transformation*, Ph.D. thesis, Technical University of Berlin, Department of Computer Science, 1990.

GI-Fachgruppe 0.1.5 "Automaten und Formale Sprachen"

Bericht über die Gründungsversammlung

Die Gründungsversammlung fand am 2. Oktober 1992 von 19.00 bis 19.50 Uhr während des 2. Theorietags "Automaten und Formale Sprachen" in Kiel statt. Anwesend waren die 45 Teilnehmer des Theorietags.

1. Verabschiedung der Fachgruppenordnung

Der den Anwesenden zuvor zugesandte Entwurf der Fachgruppenordnung wurde in einigen Punkten geändert. In der unten abgedruckten Fassung wurde die Fachgruppenordnung dann mit den Stimmen der 18 anwesenden GI-Mitglieder (ohne Gegenstimmen oder Enthaltungen) verabschiedet. Damit war die Fachgruppe gegründet. Die anwesenden Nicht-Mitglieder der GI wurden auf ihren Antrag hin durch die Gründungsorganisatoren (J. Dassow und W. Thomas als amtierende Fachgruppenleitung) als Mitglieder in die Fachgruppe aufgenommen.

2. Wahl der Fachgruppenleitung

Wahlleiter war Erich Valkema (Kiel), als Beisitzer(in) fungierten Anca Muscholl und Andreas Potthoff. Als Kandidaten für die Fachgruppenleitung wurden vorgeschlagen (in alphabetischer Reihenfolge): Franz-Josef Brandenburg, Jürgen Dassow, Volker Diekert, Helmut Jürgensen, Klaus-Jörn Lange, Helmut Seidl, Ludwig Staiger, Wolfgang Thomas. Die anwesenden Kandidaten erklärten ihr Einverständnis; von Klaus-Jörn Lange lag eine schriftliche Einverständniserklärung vor. Die schriftliche und geheime Wahl ergab folgende Stimmenverteilung: Franz-Josef Brandenburg 21, Jürgen Dassow 29, Volker Diekert 23, Helmut Jürgensen 19, Klaus-Jörn Lange 23, Helmut Seidl 8, Ludwig Staiger 14, Wolfgang Thomas 36. Damit waren Franz-Josef Brandenburg, Jürgen Dassow, Volker Diekert, Klaus-Jörn Lange und Wolfgang Thomas für drei Jahre als Fachgruppenleitung gewählt.

In der anschließend durchgeführten Versammlung der Fachgruppenleitung wurden (für ein Jahr) Wolfgang Thomas als Sprecher und Jürgen Dassow als stellvertretender Sprecher der Fachgruppe gewählt.

3. Nächster Theorietag

Volker Diekert erklärte sich zur Ausrichtung des 3. Theorietags "Automaten und Formale Sprachen" im Herbst 1993 in Stuttgart bereit.

4. Berichte

Es wurden folgende Punkte angesprochen:
- die zuweilen mangelhafte Vertretung des Gebiets Automaten und Formale Sprachen auf Tagungen zur Theoretischen Informatik,
- die für 1994 geplante Konzentration der Tagung CAAP auf die Thematik Automaten und Formale Sprachen (in einem weiten Sinn),
- die Tagung STACS 93 in Würzburg (mit 256 eingereichten Arbeiten!)
- die Deadlines für CAAP 93 (15.10.92), ICALP 93 (15.11.92), MFCS 93 (15.1.93).

5. Anregungen für die Fachgruppenarbeit

Anmeldungen zu Vorträgen zukünftiger Theorietage sollten nach Dringlichkeit abgestuft sein, um den Organisatoren die Zusammenstellung des Programms zu erleichtern. Weitere Punkte wurden in der anschließenden Diskussion der Fachgruppenleitung zusammengetragen:

(a) Der Versand von Mitteilungen (auch Einladungen zum Theorietag) soll zwecks Porto- und Papierersparnis soweit möglich durch E-mail erfolgen. Hierzu soll jedes Mitglied gefragt werden, ob es (für sich) mit der Benachrichtigung nur durch E-mail einverstanden ist. Eine entsprechende E-mail-Adressenliste soll angelegt werden.

(b) Eine Aufstellung über die im Bereich der Fachgruppe kürzlich durchgeführten bzw. laufenden Drittmittelprojekte erscheint sinnvoll. (Ziele: Information über "benachbarte" Projekte, Ermutigung zum Stellen von Anträgen, Bestimmung des Gewichts der Drittmittelprojekte auch im Bereich der Theoretischen Informatik.)

(c) Man kann an die Zusammenfassung (und elektronische Abrufbarkeit) weiterer Informationen denken, und zwar über
   1. abgeschlossene bzw. (soweit die Autoren einverstanden sind) laufende Dissertationsvorhaben,
   2. abgeschlossene Diplomarbeiten im Bereich "Automaten und Formale Sprachen" und Umgebung,
   3. Titel (oder sogar Titel und Abstracts) der im Bereich der Fachgruppe erstellten Technischen Berichte.

Es sollen zunächst (a), (b), (c1) umgesetzt werden. Die Punkte (c2) und (c3) brauchen eine organisatorische Absicherung und sollen auf dem nächsten Theorietag diskutiert werden.

J. Dassow
W. Thomas

Fachgruppenordnung

der GI-Fachgruppe 0.1.5 "Automaten und Formale Sprachen"

1. Übergeordnete Satzungen
Für die Fachgruppe (FG) sind verbindlich: die Satzung der GI, die Geschäftsordnung der GI-Gliederungen, die jeweiligen Beschlüsse des Präsidiums und die jeweiligen Beschlüsse der Leitung des Fachausschusses 0.1 "Theoretische Informatik" sowie des Fachbereichs 0 "Grundlagen der Informatik".

2. Name
Die Fachgruppe führt die Bezeichnung "Automaten und Formale Sprachen".

3. Einbettung
Die Fachgruppe ist dem Fachausschuß 0.1 "Theoretische Informatik" des Fachbereichs 0 "Grundlagen der Informatik" als Fachgruppe 0.1.5 zugeordnet.

4. Aufgaben und Ziele der Fachgruppe
Die Fachgruppe ist ein Forum der GI für die an der Thematik "Automaten und Formale Sprachen" interessierten Informatiker. Mit den anderen Fachgruppen des Fachbereichs 0 "Grundlagen der Informatik" bestehen enge Verbindungen.

Inhaltliche Schwerpunkte liegen unter anderem auf folgenden Gebieten:
(1) Automaten und Klassen formaler Sprachen (über Wörtern, Bäumen, Spuren, Graphen, Bildern etc.), Bezüge zu Komplexitäts-, Rekursions- und Schaltkreistheorie.
(2) Zeichen- und Term-Ersetzungssysteme, Graph-Grammatiken, L-Systeme und verwandte Modelle.
(3) Verteilte Automatenmodelle, darunter zellulare Automaten, systolische Automaten, Petrinetze, neuronale Netze.
(4) Algebraische Methoden (u.a. Halbgruppen, Formale Potenzreihen).
(5) Automaten und Logik bzw. Semantik (u.a. Transitionssysteme in Semantik und Spezifikation verteilter Systeme, Programmverifikation, dynamische Logiken).
(6) Kombinatorische und algorithmische Fragen (u.a. Codes, Pattern Matching).
(7) Automatentheorie und Programmierung (u.a. Übersetzerbau, Programmschemata).

Zu den Zielen der Fachgruppe gehören
- die Förderung der Kommunikation durch Treffen und Mitteilungen,
- die Bündelung von Aktivitäten durch spezifische Veranstaltungen und Koordinierung von Forschungsprojekten,
- die Vertretung des Gebiets innerhalb und außerhalb der GI.
- die Förderung der Lehre in diesem Gebiet.

5. Aktivitäten der Fachgruppe

Die Fachgruppe unterstützt Veranstaltungen und andere Aktivitäten zur Thematik "Automaten und Formale Sprachen". Sie veranstaltet in regelmäßigen Abständen, wenigstens einmal jährlich, einen "Theorietag Automaten und Formale Sprachen". Sie fühlt sich mitverantwortlich für regelmäßig stattfindende einschlägige Konferenzen wie STACS sowie für entsprechende Seminare im Internationalen Begegnungs- und Forschungszentrum Schloß Dagstuhl.

6. Mitgliedschaft

(a) Beitritt

Jedes persönliche GI-Mitglied oder jeder Mitgliedsvertreter eines fördernden GI-Mitglieds kann durch schriftliche Mitteilung an die Geschäftsstelle Mitglied in der Fachgruppe werden; Personen, die nicht GI-Mitglieder sind, kann die Fachgruppenleitung (FGL) auf schriftlichen Antrag als Mitglied in die Fachgruppe aufnehmen. Die Mitgliedschaft gilt mindestens für ein Kalenderjahr. Ein Jahresbeitrag wird nicht erhoben.

(b) Ende der Mitgliedschaft

Die Fachgruppenmitgliedschaft endet durch Tod, Austritt oder Ausschluß. Jedes Fachgruppenmitglied kann durch schriftliche Erklärung gegenüber der Geschäftsstelle zum Jahresende austreten. Die Erklärung muß bis 1. Oktober desselben Jahres oder spätestens vier Wochen nach Bekanntgabe einer Änderung dieser Fachgruppenordnung eingegangen sein. Die FGL kann, bei Vorliegen eines wichtigen Grundes, einstimmig den Ausschluß eines Mitglieds der Fachgruppe beschließen.

7. Fachgruppenleitung (FGL)

(a) Zusammensetzung

Die FGL setzt sich aus fünf Mitgliedern der FG zusammen, die von der FG auf maximal drei Jahre gewählt werden.

(b) Sprecher der FG

Die FGL wählt aus ihrer Mitte den Sprecher der FG und dessen Stellvertreter auf maximal drei Jahre; beide müssen Mitglieder der GI sein und von der Leitung des Fachausschusses 0.1 "Theoretische Informatik" bestätigt werden.

(c) Vorzeitiges Ausscheiden von FGL-Mitgliedern

Scheidet der FG-Sprecher vorzeitig aus, werden der Sprecher und sein Stellvertreter neu gewählt. Scheidet der stellvertretende Sprecher vorzeitig aus, so wird ein Nachfolger aus der Mitte der FGL neu gewählt. Scheidet ein Mitglied der FGL aus, so wird ein Mitglied der FG neu in die FGL gewählt.

(d) Die FGL kann ihren Sprecher oder seinen Stellvertreter durch Neuwahl vorzeitig von seinen Aufgaben entbinden.

(e) Die Leitung des Fachausschusses 0.1 "Theoretische Informatik" kann mit der absoluten Mehrheit ihrer Mitglieder den Sprecher der FG von seinen Aufgaben entbinden; die Neuwahl des Sprechers erfolgt entsprechend 7(b). Erhebt die FGL gegen die Entbindung ihres Sprechers Einspruch, entscheidet das Präsidium; weist das

Präsidium den Einspruch zurück, muß die FGL entsprechend 8. neu gewählt werden.

8. Wahl für die Mitglieder der FGL

Die Wahl erfolgt durch die Einberufung einer Versammlung der Mitglieder der Fachgruppe, in der Regel anläßlich des "Theorietags Automaten und Formale Sprachen" 8(a), oder - nach Beschluß der amtierenden FGL - durch Briefwahl 8(b).

(a) Wahl der FGL durch eine Wahlversammlung

Der Sprecher der FG beruft mit einer Frist von sechs Wochen eine Versammlung der FG-Mitglieder zur Wahl der FGL ein; der Einladung muß eine Liste der Kandidaten für die FGL beigefügt sein. Der FG-Sprecher eröffnet die Wahlversammlung, veranlaßt die Wahl eines Wahlleiters und übergibt diesem dann die Versammlungsleitung. Der Wahlleiter eröffnet nochmals die Kandidatenliste; die endgültige Kandidatenliste muß mindestens so viele Plätze enthalten, wie FGL-Mitglieder zu wählen sind, und darf nur Kandidaten enthalten, die in der Versammlung einer Kandidatur mündlich oder zuvor schriftlich zugestimmt haben. Gibt es weniger Plätze in der FGL als Kandidaten vorhanden sind, sind die Kandidaten mit der höchsten Stimmzahl gewählt; ist die Zahl der Kandidaten gleich der Zahl der FGL-Mitglieder, sind die Kandidaten gewählt, die mehr Ja- als Nein-Stimmen erhalten haben. Der Wahlleiter schreibt ein Wahlprotokoll, das der Wahlversammlung zur Annahme vorgelegt wird. Der Wahlleiter übersendet das Wahlprotokoll an den Sprecher des Fachausschusses 0.1 "Theoretische Informatik".

(b) Briefwahl der FGL

(b.1) Wahlleiter

Die amtierende FGL bestellt eine Wahlkommission, bestehend aus einem Wahlleiter, seinem Stellvertreter und bis zu 2 Beisitzern zur Durchführung der Briefwahl für die Mitglieder der FGL.

(b.2) Kandidatenvorschläge

Der Wahlleiter sammelt Kandidatenvorschläge aus der Mitte der FG; kandidieren kann jedes Mitglied der FG mit Ausnahme der Mitglieder der Wahlkommission. Die endgültige Kandidatenliste muß mindestens so viele Namen enthalten, wie Mitglieder der FGL zu wählen sind.

(b.3) Briefwahlunterlagen

Die Unterlagen für die Briefwahl umfassen:

- den Stimmzettel, der die Kandidatenliste enthält; bei jedem Kandidaten kann entweder "Ja" oder "Nein" angekreuzt werden;
- einen unbeschrifteten Briefumschlag zur Aufnahme des ausgefüllten Stimmzettels;
- einen größeren Briefumschlag mit der Anschrift des Wahlleiters als Empfänger sowie mit Namen, Anschrift, Mitgliedsnummer und Unterschrift des absendenden Fachgruppenmitglieds;
- ein Informationsblatt, das stichwortartige Angaben zur Person der Kandidaten sowie eine Beschreibung des Wahlverfahrens und den Endtermin für den Eingang des Wahlbriefs beim Wahlleiter enthält.

(b.4) Feststellung des Wahlergebnisses
Zur Feststellung des Wahlergebnisses beruft der Wahlleiter den Wahlausschuß unverzüglich ein. Gewählt ist, wer mehr Ja- als Nein-Stimmen erhält. Wurden mehr Kandidaten vorgeschlagen als Positionen in der FGL zu besetzen sind, so sind die Kandidaten mit den höchsten Zustimmungsergebnissen (Ja-/Nein-Stimmen-Differenz) gewählt. Der Wahlleiter fertigt ein Protokoll über die Feststellung des Wahlergebnisses an, das die Mitglieder der Wahlkommission unterzeichnen.

9. Verfahren für die Wahl des FG-Sprechers
Der FG-Sprecher und sein Stellvertreter werden von der FGL aus ihrer Mitte in einer Wahlversammlung oder durch Briefwahl wie in 8. gewählt. Über die offene oder - auf Antrag - geheime Wahl des Sprechers und seines Stellvertreters fertigt der Wahlleiter ein Protokoll an und übersendet dieses der Leitung des Fachausschusses 0.1 "Theoretische Informatik" zur Bestätigung der Wahl. Bis zur Bestätigung des neuen Sprechers und seines Stellvertreters bleibt die alte FGL im Amt.

10. Verfahren bei Auflösung der FG
Die Auflösung der Fachgruppe erfolgt auf Antrag der übergeordneten GI-Gliederung(en) und bedarf der Bestätigung durch die Leitung des Fachbereichs 0 "Grundlagen der Informatik" und durch das Präsidium. Die FGL muß vor einem Auflösungsantrag an das Präsidium gehört werden.

11. Gültigkeit
Diese Ordnung der Fachgruppe wurde von der Versammlung der FG am 2.10.1992 verabschiedet und am vom Präsidium genehmigt.

Adressenliste der Teilnehmer

Dr. Henning Bordihn
Technische Universität Magdeburg
Fakultät für Mathematik
PF 4120
Universitätsplatz 2

O-3010 Magdeburg

Tel.: +49-391 5592 2851
e-mail:
reichel@dmdtu11.bitnet

Oliver Brock
Wachtelstr. 20

W-1000 Berlin 22

Tel.: +49-30 8326789
e-mail:
brock@cs.tu-berlin.de

Mathias Bull
Universität Rostock
Fachbereich Informatik
Albert-Einstein-Str. 21

O-2500 Rostock

Tel.:
e-mail:

Olaf Burkart
Lehrstuhl für Informatik II
RWTH Aachen
Ahornstr. 55

W-5100 Aachen

Tel.: +49-241 80 21241
e-mail:
burkart@zeus.informatik.rwth-aachen.de

Prof. Dr. Franz Josef Brandenburg
Universität Passau
Lehrstuhl Theoretische Informatik
Innstraße 33

W - 8390 Passau

Tel.: +49-851 509 343
e-mail:
brandenb@fmi.uni-passau.de

Dr. Anne Brüggemann-Klein
Universität Freiburg
Institut für Informatik
Rheinstr. 10-12

W-7800 Freiburg

Tel.: +49-261 203 3897
e-mail:
brueggemann@informatik.uni-freiburg.de

Dr. Gerhard Buntrock
Universität Würzburg
Lehrstuhl für Informatik IV
Am Exerzierplatz

W-8700 Würzburg

Tel.: +49-931 887810
e-mail:
bunt@informatik.uni-wuerzburg.de

Dr. Didier Caucal
Université de Beaulieu
Avenue du Général Leclerc

F-35042 Rennes Cedex

Tel.:
e-mail:
caucal@irisa.fr

Peter Dahm
Burgweg 66 A

W-5400 Koblenz

Tel.:
e-mail:

Prof. Dr. Jürgen Dassow
Technische Universität Magdeburg
Fakultät für Mathematik
PSF 4120
Universitätsplatz 2

O-3010 Magdeburg

Tel.: +49-391 55923538
e-mail:
dassow@dmtu11.bitnet

Prof. Dr. Volker Diekert
Universität Stuttgart
Institut für Informatik
Breitwiesenstr. 20-22

W-7000 Stuttgart 80

Tel.: +49-711 7816 328
e-mail:
diekert@informatik.uni-stuttgart.de

Prof. Dr. J. Duske
Universität Hannover
Institut für Informatik
Welfengarten 1

W-3000 Hannover 1

Tel.: +49-511 762 3295
e-mail:

Henning Fernau
TU Karlsruhe
Lehrstuhl Informatik für Ingenieure
und Naturwissenschaftler
Postfach 6980

W-7500 Karlsruhe 1

Tel.: +49-721 608 4336
e-mail:
fernau@ira.uka.de

Dr. Rudolf Freund
Technische Universität Wien
Institut für Computersprachen
Resselgasse 3,
A-1040 Wien

Tel.: +43-222 58801 4084
e-mail:
freund@csdec1.tuwien.ac.at

Stefan Gärtner
TU Braunschweig
Institut für Theoretische Informatik
Gaußstr. 11

W-3300 Braunschweig

Tel.: +49-531 391 2379
e-mail:
gaertner@ips.cs.inf-bs.de

Dr. Annegret Habel
Universität Bremen
Fachbereich Mathematik
Postfach 330440

W-2800 Bremen 33

Tel.: +49-421 218 3489
e-mail:
habel@informatik.uni-bremen.de

Prof. Dr. Armin Hemmerling
EMA-Universität Greifswald
Fachrichtung Mathematik/Informatik
Jahnstraße 15a

O-2200 Greifswald

Tel.: +49-3834 5271 220
e-mail:
hemmerling@math-inf.uni-greifswald.dbp.de

Markus Holzer
TU München
Institut für Informatik
Arcisstr. 21

W-8000 München 2

Tel.: +49-89 2105 2390
e-mail:
holzer@informatik.tu-muenchen.de

Dr. Eckehart Hotzel
Gesellschaft für Mathematik und Datenverarbeitung
Postfach 1316

W-5205 St. Augustin 1

Tel.: +49-2241 142309
e-mail:
hotzel@gmd.de

Juraj Hromkovic
Universität-Gesamthochschule Paderborn
FB 17: Mathematik-Informatik
Postfach 1621

W-4790 Paderborn

Tel.: 05251 603344
e-mail:
juraj@uni-paderborn.de

Michaela Huhn
Universität Hildesheim
Institut für Informatik
Samelsonplatz

W-3200 Hildesheim

Tel.:
e-mail:
huhn@informatik.uni-hildesheim.de

Claudia Hümpel
Universität Hannover
Institut für Informatik
Welfengarten 1

W-3000 Hannover 1

Tel.: +49-511 762 3295
e-mail:
ch@informatik.uni-hannover.dbp.de

Prof. Dr. Helmut Jürgensen
The University of Western Ontario
Department of Computer Science

London, Ont.
Canada N6A 5B7

Tel.: +1 519 661 3560
e-mail:
helmut@uwo.ca

Eija Jurvanen
Universität Kiel
Institut für Informatik u. Praktische Mathematik
Olshausenstr. 40

W-2300 Kiel

Tel.: +49-431 880 4663
e-mail:
ej@informatik.uni-kiel.dbp.de

Martin Korff
Technische Universität
FB 20 - Sekr. FR 6-1
Franklinstr. 28/29

W-1000 Berlin 10

Tel.: +49-30 314 27787
e-mail:
martin@cs.tu-berlin.de

Prof. Dr. Klaus-Jörg Lange
Technische Universität München
Institut für Informatik
Arcisstraße 21

W-8000 München

Tel.: +49-89 2105 2403
e-mail:
lange@informatik.tu-muenchen.dbp.de

Anca Muscholl
Universität Stuttgart
Institut für Informatik
Breitwiesenstr. 20-22

W-7000 Stuttgart 80

Tel.: +49-711 7816 491
e-mail:
muscholl@informatik.uni-stuttgart.de

Dr. Gheorghe Paun
Technische Universität Magdeburg
Fakultät für Mathematik
Universitätsplatz 2

O-3010 Magdeburg

Tel.:
e-mail:

Armin Kühnemann
Universität Ulm
Abt. Theoretische Informatik
Oberer Eselsberg

W-7900 Ulm

Tel.:
e-mail:
kuehne@informatik.uni-ulm.de

Dr. Martin Middendorf
Universität Karlsruhe
Inst. für Angewandte Informatik
und Formale Beschreibungsverfahren
Postfach 6980

W-7500 Karlsruhe

Tel.: +49-511 762 3295
e-mail:
mmi@aifb.uni-karlsruhe.de

Prof. Dr. Friedrich Otto
GH Kassel
Fachbereich Mathematik-Informatik
Postfach 10 13 80

W-3500 Kassel

Tel.: +49-561 804 4573
e-mail:
otto@theory.informatik.uni-kassel.de

Andreas Potthoff
Universität Kiel
Institut für Informatik u. Prakt. Mathematik
Olshausenstr. 40

W-2300 Kiel 1

Tel.: +49-431 880 4389
e-mail:
apo@informatik.uni-kiel.dbp.de

Friedrich Rancke
Sillemstr. 68

W-2000 Hamburg 20

Tel.:
e-mail:

Torsten Roßnick
Technische Universität Magdeburg
Fakultät für Mathematik
Universitätsplatz 2
PF 4120

O-3010 Magdeburg

Tel.: +49-391 5592 2851
e-mail:
reichel@dmtu11.bitnet

Michael Schlitt
Alleestr. 13

W-5440 Mayen

Tel.:
e-mail:

Sebastian Seibert
Universität Kiel
Institut für Informatik u. Prakt. Mathem.
Olshausenstr. 40
W-2300 Kiel 1

Tel.: +49-431 880 4483
e-mail:
ss@informatik.uni-kiel.dbp.de

Dr. Bernd Reichel
Technische Universität Magdeburg
Fakultät für Mathematik
Universitätsplatz 2
PF 4120

O-3010 Magdeburg

Tel.: +49-391 5592 2851
e-mail:
reichel@dmdtu11.bitnet

Dr. Jürgen Schäfer
Hoechst AG
Computer Lernzentrum
Postfach 800320

W-6230 Frankfurt

Tel.: +49-69 305 2377
e-mail:

Thomas Schwentick
Universität Mainz
FB 17
Saarstr. 21

W-6500 Mainz

Tel.:
e-mail:
tick@vaimzti.mathematik.uni-mainz.de

Dr. Helmut Seidl
Universität Saarbrücken
Fachbereich 14 - Informatik
Im Stadtwald 15

W-6600 Saarbrücken 11

Tel.: +49-681 302 2454
e-mail:
seidl@sol.cs.uni-sb.de

Stefan Skalla
Technische Universität Magdeburg
Fakultät für Mathematik
Universitätsplatz 2

O-3010 Magdeburg 4120

Tel.:
e-mail:

Dr. Ludwig Staiger
Techn. Universität Cottbus
Informatik

O-7500 Cottbus

Tel.: +49-355 69 2209 oder 2236
e-mail:
staiger@server.informatik.uni-siegen.de

Ralf Stiebe
Technische Universität Magdeburg
Fakultät für Mathematik
Universitätsplatz 2

O-3010 Magdeburg

Tel.:
e-mail:

Heiko Thiesen
Katharinen Str. 2 A

W-2390 Flensburg

Tel.:
e-mail:

Prof. Dr. Wolfgang Thomas
Universität Kiel
Institut für Informatik u. Prakt. Mathematik
Olshausenstr. 40

W-2300 Kiel 1

Tel.: +49-431 880 4480
e-mail:
wt@informatik.uni-kiel.dbp.de

Dr. Erich Valkema
Universität Kiel
Institut für Informatik u. Prakt. Mathematik
Olshausenstr. 40

W-2300 Kiel

Tel.: +49-431 880 4482
e-mail:
ev@informatik.uni-kiel.dbp.de

Prof. Dr. Walter Vogler
Institut für Mathematik
Universität Augsburg
Universitätsstr. 8

8900 Augsburg

Tel.: z.Z. +49-89 2105 8237
e-mail:
vogler@informatik.tu-muenchen.dbp.de

Thomas Wilke
Universität Kiel
Institut für Informatik u. Praktische Mathematik
Olshausenstr. 40

W-2300 Kiel

Tel.: +49-431 880 4662
e-mail:
tw@informatik.uni-kiel.dbp.de

Oliver Wittig
Tile-Wardenberg-10

1000 Berlin 21

Tel.: +49-30 31423 223
e-mail:
wittig@cs.tu-berlin.de

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität
Olshausenstr. 40
D - 2300 Kiel 1

Die folgenden Berichte sind kostenlos erhältlich.
The following reports are available free of charge.

9101 Dieter Ackermann: Analyse polymorpher Typen bei logischen Programmiersprachen mittels Substitutionsgrammatiken

9102 Bernhard Steffen, Jens Knoop, Oliver Rüthing: Optimal Code Motion within Flow Graphs: A Practical Approach

9103 Jens Knoop, Bernhard Steffen: Strength Reduction based on Code Motion: A Uniform Approach

9104 ESPRIT BRA Working Group 3166 "ASMICS"
("Algebraic and Syntactic Methods in Computer Science")
Proceedings of the Workshop "LOGICS AND RECOGNIZABLE SETS"
Dersau, Germany, October 8-10, 1990

9105 Willem-Paul de Roever: Foundations of Computer Science: Leaving the Ivory Tower

9106 Jos Coenen, Willem-Paul de Roever, Job Zwiers: Assertional Data Reificating Proofs: Survey and Perspective

9107 J. Hooman, S. Ramesh, W.-P. de Roever: A Compositional Axiomatization of Statecharts

9108 Wolfgang Thomas: On Logics, Tilings, and Automata

9109 Jozef Hooman, Willem-Paul de Roever: An introduction to compositional methods for concurrency and their application to real-time

9110 Wolfgang Thomas: Neue Aspekte in Rabins Theorie der Baumautomaten

9111 Jozef Hooman, Willem-Paul de Roever: Verification and Specification of Concurrent Programs

9112 Manfred Schimmler: Graphenalgorithmen auf gitterverbundenen Prozessorfeldern

9113 Stefan Sauter: On the Stability of the ILU Method for a Degenerate Grid

9114 Thorsten Schmidt: Eine Analyse der Finite-Volumen-Element-Methode auf Vierecksgittern am Beispiel der Poissongleichung

9115 Stefan Sauter: Der Aufwand der Panel-Clustering-Methode für Integralgleichungen

9201 Dietmar Gärtner: Π-Red$^+$: Ein interaktives codeausführendes Reduktionssystem zur vollständigen Realisierung eines angewandten λ-Kalküls

9202 Thomas Wilke: An Algebraic Theory for Regular Languages of Finite and Infinite Words

9203 Thomas Wilke: Locally Threshold Testable Languages of Infinite Words

9204 Sebastian Seibert: Quantifier Hierarchies over Word Relations

9205 Peter Kosmol: Sekantenverfahren für lineare Gleichungen

9206 Rüdiger Maaß: Testgenerierung für die Operandenauswahl über Multiplexer mit Voting-Verhalten

9207 Rüdiger Maaß, Manfred Schimmler, Hartmut Schmeck: Abschlußbericht über das Forschungsvorhaben Fehlertoleranz und Testbarkeit von VLSI-Algorithmen

9208 Jens Knoop, Oliver Rüthing, Bernhard Steffen: Lazy Code Motion

9209 Claus Aßmann: Fast: Eine Prozessor-Architektur mit einem neuartigen Stacksystem

9210 Kai Goetzke: Graphentheoretische und algebraische Verfahren für die Analyse von Kristallstrukturen

9211 Tätigkeitsbericht 1990/91

9212 Karl-Heinz Buth: Simulation of Transition Systems with Term Rewriting Systems

9213 Harald Blödorn: Organisatorische Maßnahmen zur effizienten Unterstützung höherer Sprachkonstrukte in einem interaktiven Reduktionssystem

9214 Karl-Heinz Buth: Using SOS Definition in Term Rewriting Proofs

9215 Frank A. Stomp, Willem-Paul de Roever: Principles for sequential reasoning about distributed algorithms

9216 Jens Knoop, Oliver Rüthing, Bernhard Steffen: Lazy Strength Reduction

9219 Wolfgang Hackbusch, Stefan A. Sauter: On the efficient use of the Galerkin method to solve Fredholm integral equations

9220 2. Theorietag "Automaten und Formale Sprachen"