5. Theorietag

„Automaten und Formale Sprachen“

Schloß Rauischholzhausen bei Gießen

28./29. September 1995

Martin Kutrib, Thomas Worsch (Hrsg.)

Preproceedings

JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN

Inhaltsverzeichnis

Programm

Donnerstag, 28. September 1995

9:20– 9:30 Begrüßung

9:30–10:30 Eingeladener Vortrag:
Rüdiger Reischuk (Universität Lübeck), *Turingmaschinen mit sublogarithmischen Platzschranken*

10:30–11:00 Kaffeepause

11:00–11:20 Martin Hühne (Universität Dortmund), *Effiziente Speicherstrukturen für Turingmaschinen mit* einem *Kopf*

11:20–11:40 Klaus P. Jantke (Hochschule für Technik, Wirtschaft und Kultur Leipzig), *Introducing a Two-Level Grammar Concept for Design*

11:40–12:00 Armin Kühnemann (Technische Universität Dresden), *A Pumping Lemma for Output Languages of Attributed Tree Transducers*

12:00–12:20 Gerhard Buntrock (Universität Würzburg), *Einige Bemerkungen zum Pumpen*

12:20–14:20 Mittagspause

14:20–14:40 Ludwig Staiger (Universität Halle-Wittenberg), *Uniformly optimal prediction of infinite words*

14:40–15:00 Andreas Schrader (Universität-GH Siegen), *Bildkompression mit endlichen Automaten*

15:00–15:20 Thomas Buchholz (Universität Gießen), *Konstruierbarkeit von Funktionen in zellularen Automaten*

15:20–15:50 Kaffeepause

15:50–16:10 Klaus-Jörn Lange (Universität Tübingen), *Automaten mit der Datenstruktur Menge*

16:10–16:30 Manfred Droste (Technische Universität Dresden), *Erkennbare, aperiodische und logisch definierbare Sprachen in Nebenläufigkeitsmonoiden*

16:30–16:50 Rudolf Freund (Technische Universität Wien), *Contextual Array Grammars*

16:50–17:50 Vollversammlung der GI Fachgruppe 0.1.5
Bericht der Fachgruppenleitung

19:00– „Wild Cat Session“

Freitag, 29. September 1995

9:00– 9:20 Thomas Worsch (Universität Karlsruhe), *Automaten mit ein bißchen Parallelität*

9:20– 9:40 Henning Fernau (Universität Tübingen), *Programmierte Grammatiken mit unbedingtem Übergang und verwandte Mechanismen*

9:40–10:00 Henning Bordihn (Universität Magdeburg), *Akzeptierende programmierte Grammatiken ohne Nichtterminale*

10:00–10:20 Markus Holzer (Universität Tübingen), *On the Complexities of Deterministic Indexed Grammars*

10:20–10:50 Kaffeepause

10:50–11:10 Martin Kutrib (Universität Gießen), *On language families of interacting automata*

11:10–11:30 Norbert Gutleben (Technische Universität Cottbus), *Minimierung von Muller-Automaten*

11:30–11:50 Helmut Seidl (Universität Trier), *A Modal Mu-Calculus for Durational Transition Systems*

11:50–12:10 David Damanik (Universität Frankfurt), *Endliche Automaten mit eingeschränkter 2-Wege-Bewegung*

12:10–12:30 Daniel Kirsten (Bad Dürrenberg), *Properties of formal languages of therapy plans created by graph grammars*

12:30– Mittagspause und Abreise

Akzeptierende programmierte Grammatiken ohne Nichtterminale

Henning Bordihn

Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg
Postfach 4120
D-39016 Magdeburg
email: bordihn@irb.cs.uni-magdeburg.de

Henning Fernau

Wilhelm-Schickhard-Institut
für Informatik,
Universität Tübingen,
Sand 13,
72076 Tübingen,
email: fernau@informatik.uni-tuebingen.de

In unseren Arbeiten über akzeptierende Grammatiken und Systeme [1, 2, 5] haben wir nachgewiesen, daß die triviale Äquivalenz zwischen generierenden und akzeptierenden Typ-i-Grammatiken der Chomsky-Hierarchie (s. z.B. [6]) auf eine Vielzahl von Grammatiktypen nicht übertragen werden kann. Ein Beispiel für dieses Phänomen stellen die programmierten Grammatiken (mit Vorkommenstest und kontextfreien Kernregeln) dar, die im akzeptierenden Modus eine echt größere Beschreibungsmächtigkeit besitzen als im generierenden Modus. Diese Erkenntnis wird in den genannten Arbeiten aus dem Beweis gewonnen, daß die akzeptierenden programmierten Grammatiken ohne löschende Regeln exakt die Familie der kontextsensitiven Sprachen definieren bzw. jede rekursiv-aufzählbare Sprache beschreiben können, falls λ-Regeln erlaubt sind, während bekanntermaßen die Sprachfamilien der zugehörigen generierenden Grammatiken echte Teilmengen der Familien $\mathcal{L}(\mathrm{CS})$ bzw. $\mathcal{L}(\mathrm{RE})$ sind (vgl. z.B. [3]). Die Simulation der Typ-1- bzw. Typ-0-Grammatiken durch akzeptierende programmierte Grammatiken erfordert die Erweiterung des Alphabets der Nichtterminale um eine Reihe von Hilfssymbolen. In diesem Vortrag untersuchen wir nun die reinen Versionen dieser Grammatiken, bei denen (wie im Falle der Lindenmayer-Systeme) nicht zwischen terminalen und nichtterminalen Symbolen unterschieden wird.

Eine reine programmierte Grammatik [3, 3] ist ein Tripel $G = (V, P, S)$. Hier ist V ein Alphabet, $S \subseteq V^+$ eine endliche Menge von Axiomen und P eine endliche Menge von Produktionen der Form $(r : u \to v, \sigma(r), \phi(r))$, wobei r

die Marke der Produktion, $u \to v$ eine Ersetzungsregel mit $u \in V^*$, $v \in V^*$ ist und $\sigma(r)$ und $\phi(r)$ Mengen von Marken ausgewählter Produktionen aus P sind. Eine Folge $y_0, y_1, \ldots, y_n$ von Wörtern über V heißt genau dann Ableitung in G, wenn für jedes $i \in \{1, \ldots, n\}$ eine Produktion $(r_i : u_i \to v_i, \sigma(r_i), \phi(r_i))$ existiert, so daß

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

oder

$$u_i \text{ ist kein Teilwort von } y_{i-1},\ y_{i-1} = y_i, \text{ und, falls } 1 \leq i < n,\ r_{i+1} \in \phi(r_i)$$

gelten. Dann schreiben wir $y_0 \stackrel{n}{\Rightarrow} y_n$ und definieren die von G generierte Sprache als

$$L_{\text{gen}}(G) = \{w \in V^* \mid s \stackrel{n}{\Rightarrow} w \text{ für ein } s \in S \text{ und ein } n \geq 0\}$$

und die von G akzeptierte Sprache als

$$L_{\text{acc}}(G) = \{w \in V^* \mid w \stackrel{n}{\Rightarrow} s \text{ für ein } s \in S \text{ und ein } n \geq 0\}\,.$$

$L_{\text{gen}}(G) \in \mathcal{L}_{\text{gen}}(\text{pP,CF,ac})$ gelte genau dann, wenn jede Kernregel $u \to v$ aus der Regelmenge von G eine reine kontextfreie Regel ist, d.h. falls u ein Symbol ist. Entsprechend sei $L_{\text{acc}}(G) \in \mathcal{L}_{\text{acc}}(\text{pP,CF,ac})$ genau dann, wenn für alle Kernregeln $u \to v$ aus G gilt, daß v ein Symbol ist. Gibt es keinen Vorkommenstest (appearance checking) in G, d.h. gilt $\phi(r) = \emptyset$ für jede Marke r von Produktionen in G, so schreiben wir genauer $L_{\text{gen}}(G) \in \mathcal{L}_{\text{gen}}(\text{pP,CF})$ bzw. $L_{\text{acc}}(G) \in \mathcal{L}_{\text{acc}}(\text{pP,CF})$. Gilt für jede Marke r, daß $\sigma(r) = \phi(r)$, so schreiben wir statt dessen $L_{\text{gen}}(G) \in \mathcal{L}_{\text{gen}}(\text{pP,CF,ut})$ bzw. $L_{\text{acc}}(G) \in \mathcal{L}_{\text{acc}}(\text{pP,CF,ut})$ (unconditional transfer). Sind λ-Produktionen verboten, ersetzen wir CF durch CF$-\lambda$ in dieser Notation.

Betrachtet man reine programmierte Grammatiken ohne Vorkommenstest, so kann man die Beweistechnik zum Nachweis trivialer Äquivalenzen zwischen akzeptierendem und generierndem Modus analog zum nichtreinen Fall anwenden (s. [1, 2]). Obwohl die Beweise für die Grammatiken mit Vorkommenstest unter Verwendung von Hilfssymbolen nicht auf den reinen Fall übertragbar sind, gelingt es, die generierenden reinen programmierten Grammatiken auch in diesem Fall durch akzeptierende zu simulieren. Damit sind die entsprechenden Inklusionen im klassischen Fall noch einmal direkt (ohne den „Umweg" über die Typ-1- bzw. Typ-0-Sprachen) nachgewiesen. Die Echtheit dieser Teilmengenbeziehungen für die reinen Versionen wird mit Hilfe der Srache

$$L = \{a^n b^n \mid n \geq 1\} \cup \{a^{m+1} b a^n b^k \mid m \geq 1,\ n \geq 1,\ m + n = k\}\,,$$

für die sogar $L \in \mathcal{L}_{\text{acc}}(\text{pP,CF}-\lambda, \text{ut}) \setminus \mathcal{L}_{\text{gen}}(\text{pP, CF, ac})$ gilt, nachgewiesen.

Allerdings benötigt die verwendete Simulationsmethode die Möglichkeit von „echten" Vorkommenstests, so daß sie für reine Grammatiken mit unconditional transfer nicht verwendbar ist. In diesem Fall erhalten wir ein Unvergleichbarkeitsresultat, obwohl bei Zulassung von Nichtterminalen ebenfalls eine echte Inklusion zwischen der Familie der generierten Sprachen und der Familie der akzeptierten Sprachen besteht.

Wir fassen die Ergebnisse wie folgt zusammen: Für $\alpha \in \{\text{CF}, \text{CF}-\lambda\}$ gilt:

- $\mathcal{L}_{\text{gen}}(\text{pP},\alpha) = \mathcal{L}_{\text{acc}}(\text{pP},\alpha)$
- $\mathcal{L}_{\text{gen}}(\text{pP},\alpha,\text{ac}) \subset \mathcal{L}_{\text{acc}}(\text{pP},\alpha,\text{ac})$
- $\mathcal{L}_{\text{gen}}(\text{pP},\alpha,\text{ut})$ und $\mathcal{L}_{\text{acc}}(\text{pP},\alpha,\text{ut})$ sind unvergleichbar.

Damit haben wir zum ersten Mal einen Grammatiktyp gefunden, für den Unvergleichbarkeit zwischen den Familien der akzeptierten und der generierten Sprachen nachzuweisen war. Weitere Beispiele lassen sich bei der Untersuchung deterministischer L-Systeme ohne Nichtterminale oder ohne Tafeln finden. Ein umfassendes Papier zu diesem Thema ist in Vorbereitung.

Literatur

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

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

[3] J. Dassow. Pure grammars with regulated rewriting. *Revue Roumaine Math. Pures Et Appl.*, 31:657–666, 1986.

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

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

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

Einige Bemerkungen zum Pumpen[*]

Gerhard Buntrock[†]
Universität Würzburg

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

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

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

Nun gibt es eine unendliche Folge F von Zeichen aus $\{0,1\}$, so daß es für kein $i > 2$ eine endliche Teilfolge z von F gibt, von der Form $z = v^i$. Eine solche Folge ist 1906 von Axel Thue und 1921 von Marston Morse (wieder-)entdeckt worden [Thu06, Mor21] (siehe auch [Jac83]):

0
0 1
0 1 1 0
0 1 1 0 1 0 0 1
0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0
...
0110 1001 1001 0110 1001 0110 0110 1001 1001 0110 0110 1001 0110 1001 $\cdots$

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

Somit hat jede Sprache L, die aus einer unendlichen Menge von endlichen Teilwörtern der THUE-Folge besteht, die Eigenschaft, daß für kein $i > 2$ und

[*] Diese Arbeit wurde zum Teil mit Mitteln aus dem DFG-Projekt Wa 847/1-1 finanziert.
[†] bunt@informatik.uni-wuerzburg.de
http://haegar.informatik.uni-wuerzburg.de/person/mitarbeiter/bunt/

kein Wort $z \in L$ eine Zerlegung $z = uvw$ existiert, so daß v^i ein Teilwort eines Wortes aus L ist.

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

Andrzej Ehrenfeucht, Rohit Parikh und Grzegorz Rozenberg bewiesen 1981, daß es überabzählbar viele Sprachen gibt, die das Pumpinglemma für REG erfüllen [EPR81]. Sie benötigen für ihren Beweis 16 Alphabetsymbole. Mit einer einfachen Konstruktion, die von den möglichen Sprachen, die nur aus Teilwörtern der THUE-Folge bestehen, ausgeht, zeigen wir dieses Überabzählbarkeitsresultat für drei Alphabetsymbole. Mit einem Alphabetsymbol charakterisiert das Pumpinglemma für REG ebenso wie das Pumpinglemma für CFL die regulären Sprachen.

Literatur

[EPR81] Ehrenfeucht, Andrzej, Rohit J. Parikh und Grzegorz Rozenberg: *Pumping Lemmas for Regular Sets.* SIAM Journal on Computing, 10(3):536–541, 1981.

[Jac83] Jacobs, Konrad: *Einführung in die Kombinatork.* Walter de Gruyter, Berlin/New York, 1983.

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

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

Endliche Automaten mit eingeschränkter 2-Wege-Bewegung

David Damanik
J.W.Goethe-Universität
Fachbereich Mathematik, AG 8.1
Robert-Mayer-Str.10
60325 Frankfurt

14. September 1995

Das Basis-Automatenmodell für die Akzeptierung Regulärer Mengen ist der Deterministische Endliche 1-Wege-Automat ($1DEA$), welcher zwei naheliegende Erweiterungen zuläßt: Nichtdeterminismus und 2-Wege-Bewegung ($1NEA$ bzw. $2DEA$). Es ist bekannt, daß auf geeignet gewählten Regulären Mengen durch diese Erweiterungen gewaltige Einsparungen in der Beschreibungsgröße erreicht werden können, wobei die Größe eines Endlichen Automaten durch die Anzahl seiner Zustände gemessen wird, vgl. [MF71]. Es drängt sich die Frage auf, wieviel Nichtdeterminismus bzw. 2-Wege-Bewegung einem Endlichen Automaten erlaubt werden soll, um eine gegebene Reguläre Menge möglichst effizient zu beschreiben. Selbstverständlich sind für eine Konkretisierung geeignete Maße für diese beiden Erweiterungen einzuführen.

Im Falle des Nichtdeterminismus bietet es sich an, als Maß den maximalen Verzweigungsgrad der minimal verzweigten Berechnungen eines $1NEA$ auf den von ihm akzeptierten Wörtern zu benutzen: das Branching, vgl. [GKW90]. Goldstine, Kintala und Wotschke führen in dieser Arbeit mit Hilfe des Branchings ein Spektrum für eine Reguläre Menge L ein. Dieses ist eine monoton fallende Folge natürlicher Zahlen, deren Abfallverhalten aufzeigt, wieviel Verlangen nach nichtdeterministischer Abarbeitung inhärent in L enthalten ist.

Durch Zählen der auftretenden Linksbewegungen in den Konfigurationsfolgen eines $2DEA$ M bei Abarbeitung von Wörtern aus $T(M)$ definieren wir nun ein Maß für die Verwendung von 2-Wege-Bewegung. Durch Einschränkung bezüglich dieses Maßes erhalten wir die Automatenklasse $2DEA(k)$. Nun kann in analoger Art und Weise für reguläres L ein 2-Wege-Spektrum definiert werden, dessen Einträge gerade die Größen der minimalen $2DEA(k)$'s für L sind. Es stellt sich heraus, daß durchaus beide zu erwartenden Extremfälle auftreten: es gibt Sprachen, für die endliche 2-Wege-Bewegung keine Einsparungen liefert - erst der uneingeschränkte $2DEA$ ist kleiner als der minimale $1DEA$ für L. Andererseits gibt es Sprachen, welche durch sukzessives Hinzufügen von

2-Wege-Bewegung immer kleinere Beschreibungen erlauben.

Man kann die Klasse $2DEA(k)$ auch unter einem anderen Blickwinkel betrachten. Während im Spektrum für eine feste Sprache das k variiert wird, kann man natürlich auch das k festhalten und die Sprache variieren. Anders formuliert: man vergleicht für festes k Beschreibungen beliebiger Regulärer Mengen mittels Automaten aus $1DEA$, $2DEA(k)$, $2DEA$ und auch $1NEA$. Hier stellt sich unter anderem heraus, daß die Klassen $2DEA(k)$ und $1NEA$ in gewissem Sinne inkommensurabel sind. Es gibt Sprachen, für die die jeweilige minimale Beschreibung durch das eine Modell im Vergleich zur minimalen Beschreibung durch das andere Modell wesentlich kleiner ist.

Es gibt weitere Möglichkeiten, Maße für die Verwendung von 2-Wege-Bewegung anzugeben. Das von uns verwendete Maß ist insofern natürlich, als daß es das absolute Auftreten von Linksbewegungen zählt. Relative Ansätze liefern ferner einen erschwerten Zugang zur Berechenbarkeit des Spektrums und zum Vergleich mit den klassischen Automatenmodellen. Unser Ansatz hat allerdings den Nachteil, daß er die stärkste Einschränkung an den $2DEA$ induziert.

Literatur

[MF71] A.R.Meyer, M.J.Fischer : *Economy of Description by Automata, Grammars and Formal Systems*, 12th Annual SWAT, 1971, 188-191

[GKW90] J.Goldstine, C.Kintala, D.Wotschke : *On Measuring Nondeterminism in Regular Languages*, Information and Computation 86, 1990, 179-194

Erkennbare, aperiodische und logisch definierbare Sprachen in Nebenläufigkeitsmonoiden

Manfred Droste
Institut für Algebra
Technische Universität Dresden
Mommsenstr. 13
01062 Dresden
email: droste@@math.tu-dresden.de

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

Spuren bilden ein mathematisches Modell für ein paralleles System, in dem von der Reihenfolge zweier unabhängiger Aktionen abgesehen wird; die Unabhängigkeit wird hierbei durch eine binäre Relation auf dem zugrunde liegenden Alphabet der Aktionen beschrieben. Wir betrachten hier ein allgemeineres Modell, in dem die Unabhängigkeit zweier Aktionen auch von dem jeweiligen Zustand des zugrunde liegenden Systems abhängt. Ein *Automat mit Nebenläufigkeitsrelationen* ist ein Quadrupel $\mathcal{A} = (S, E, T, \|)$, in dem S die Menge der Zustände oder Situationen, E die Menge der Aktionen (Ereignisse), $T \subseteq S \times E \times S$ die deterministische Transitionsrelation und $\| = (\|_s)_{s \in S}$ ein System von binären Unabhängigkeitsrelationen $\|_s \quad (s \in S)$ auf E ist. Zwei Berechnungsfolgen $(s, a, p)(p, b, r)$ und $(s, b, q)(q, a, r)$ werden für äquivalent erklärt, falls $a \|_s b$. Dies induziert auf der Menge $\mathrm{BF}(\mathcal{A})$ aller endlichen Berechnungsfolgen von $\mathcal{A}$ eine Kongruenz $\sim$, und der Quotient $\mathrm{M}(\mathcal{A}) = \mathrm{BF}(\mathcal{A}) \cup \{0\}$, mit Konkatenation als Operation (und 0 als „Fehlerelement", um eine überall definierte Monoid-Operation zu erhalten) heißt *Nebenläufigkeitsmonoid* über $\mathcal{A}$. Falls $\mathcal{A}$ nur einen Zustand hat, erhalten wir so gerade die Spur-Monoide.

In vorhergehenden Arbeiten haben wir Zusammenhänge dieser Automaten mit der Theorie der Bereiche und mit Petri-Netzen mit Kapazitäten (place/-transition nets) untersucht. Ferner erhielten wir eine Kleene-Charakterisierung der erkennbaren Sprachen in Monoiden $\mathrm{M}(\mathcal{A})$, falls $\mathcal{A}$ ein endlicher stabil-nebenläufiger Automat ist. In einer Arbeit mit F.Bracho und D.Kuske wurden

Darstellungen der Berechnungen von $\mathcal{A}$ durch geeignete azyklische Graphen angegeben. Hier zeigen wir, wie sich durch Aussagen der Prädikatenlogik über derartige Graphen Sprachen von Berechnungen spezifizieren lassen. Wie im Satz von Büchi stellen sich die erkennbaren Sprachen in $\mathrm{M}(\mathcal{A})$ genau als die in der monadischen Logik zweiter Stufe definierbaren Sprachen heraus. Unter geeigneten Voraussetzungen (jedoch nicht im allgemeinen) fallen auch die aperiodischen und die in der Logik erster Stufe definierbaren Sprachen zusammen. Dies zeigt Grenzen auf, inwieweit sich die zu Anfang erwähnten klassischen Resultate auf allgemeinere Monoide übertragen lassen.
(Gemeinsame Arbeit mit D.Kuske, Dresden)

Programmierte Grammatiken mit unbedingtem Übergang und verwandte Mechanismen

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

Im September 1995

Auf dem letzten Theorietag haben wir auf seit langer Zeit offene Fragen im Bereich der regulierten Grammatiken hingewiesen. In diesem Vortrag möchten wir einige der angeschnittenen Fragen beantworten, so wie dies auch schon im Abstrakt zum letztjährigen Vortrag als nachgeschobener Hinweis enthalten ist. Bezüglich der Definitionen wollen wir auf [3] bzw. die elektronisch zugänglichen Berichte [1][1] sowie den letzten GI-Theorietagsbericht [6][2] verweisen.

Insbesondere konnten wir zeigen [4], [5], daß $\mathcal{L}_{\text{gen}}$(P,CF,ut) (und damit auch $\mathcal{L}_{\text{gen}}$(1lET0L), $\mathcal{L}_{\text{gen}}$(M,CF,ut) [9], $\mathcal{L}_{\text{gen}}$(rC,CF,ut) [8] und $\mathcal{L}_{\text{gen}}$(1lET0L) [11], ja sogar $\mathcal{L}_{\text{gen}}$(klET0L) für bel. $k \geq 1$, unentscheidbare Sprachen enthält.

Damit ist z.B. die Inklusion $\mathcal{L}_{\text{gen}}(\text{P,CF} - \lambda, \text{ut}) \subseteq \mathcal{L}_{\text{gen}}(\text{P,CF}, \text{ut})$ sogar 'echt modulo λ'. Offen bleibt jedoch die Frage der Echtheit der Inklusion $\mathcal{L}_{\text{gen}}(\text{P,CF} - \lambda) \subseteq \mathcal{L}_{\text{gen}}(\text{P,CF})$ (modulo λ).

Offen bleibt leider die Echtheit der Inklusionen (*) $\mathcal{L}_{\text{gen}}(\text{P,CF}[-\lambda],\text{ut}) \subseteq \mathcal{L}_{\text{gen}}(\text{P,CF}[-\lambda],\text{ac})$, auch wenn es starke Hinweise auf die Echtheit gibt, da Grammatiken mit unbedingtem Übergang ein entscheidbares Leerheitsproblem besitzen. Daher gibt es auch keinen Algorithmus, der eine Grammatik mit allgemeinem Vorkommenstest in eine äquivalente mit unbedingtem Übergang verwandelt. Weiterhin läßt sich $\mathcal{L}_{\text{gen}}(\text{P,CF}[-\lambda],\text{ac})$ als das von $\mathcal{L}_{\text{gen}}(\text{P,CF}[-\lambda],\text{ut})$ erzeugte volle Trio [bzw. Trio] kennzeichnen, ja, es gilt sogar, daß die fraglichen Inklusionen (*) genau dann echt sind, wenn $\mathcal{L}_{\text{gen}}(\text{P,CF}[-\lambda],\text{ut})$ gegen Durchschnitt mit regulären Sprachen abgeschlossen ist, was genau dann der Fall ist, wenn $\mathcal{L}_{\text{gen}}(\text{P,CF}[-\lambda],\text{ut})$ gegen inverse Homomorphismen abgeschlossen ist [falls diese Abschlußeigenschaften gelten, so sind sie allerdings nicht konstruktiv.].

Außerdem konnten wir noch ergänzend zu [2] zeigen, daß $\mathcal{L}_{\text{gen}}(\text{1lEPT0L}) = \mathcal{L}_{\text{gen}}(\text{P,CF} - \lambda, \text{ut})$ gilt [5].

[1] http://wsiserv.informatik.uni-tuebingen.de/~fernau/pub-fernau/
[2] http://papa.informatik.tu-muenchen.de/gitag/

Betrachten wir Linksableitungen, so ergibt sich ein teilweise anderes Bild, da insbesondere $\mathcal{L}_{\text{gen}}$(P-left,CF $-\lambda$, ut) im Gegensatz zu $\mathcal{L}_{\text{gen}}$(P,CF $-\lambda$, ut) ein lösbares Präfixproblem besitzt. Daher ist $\mathcal{L}_{\text{gen}}$(P-left,CF $-\lambda$, ut) echt in $\mathcal{L}_{\text{gen}}$(P-left,CF$-\lambda$, ac) enthalten [7, 10]. Daraus folgt auch, daß $\mathcal{L}_{\text{gen}}$(P-left,CF$-\lambda$, ut) nicht abgeschlossen ist gegen Durchschnitt mit regulären Sprachen, weshalb $\mathcal{L}_{\text{gen}}$(P-left,CF $-\lambda$, ut) von $\mathcal{L}_{\text{gen}}$(P,CF $-\lambda$) verschieden ist. $\mathcal{L}_{\text{gen}}$(P,CF $-\lambda$, ut) ist von $\mathcal{L}_{\text{gen}}$(P,CF $-\lambda$) verschieden, weil $\mathcal{L}_{\text{gen}}$(P,CF $-\lambda$) ein lösbares Präfixproblem besitzt.

Andererseits enthält auch $\mathcal{L}_{\text{gen}}$(P-left,CF, ut) unentscheidbare Sprachen, so daß diese Familie von $\mathcal{L}_{\text{gen}}$(P,CF) verschieden ist, wohingegen das Verhältnis zu $\mathcal{L}_{\text{gen}}$(P,CF, ut) offen ist.

Literatur

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

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

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

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

[5] H. Fernau. Characterizations of unconditional transfer. Manuscript (in preparation), 1995.

[6] H. Fernau and H. Bordihn. Programmierte Grammatiken und limitierte L Systeme als Sprachakzeptoren. In *4. GI Theorietag "Automaten und Formale Sprachen", Technical Report WSI-95-10, Universität Tübingen*, pages 21–24, 1995.

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

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

[9] A. K. Salomaa. On grammars with restricted use of productions. *Annales academiae scientarum Fennicae*, Serie A-454:1–32, 1969.

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

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

CONTEXTUAL ARRAY GRAMMARS

Rudolf Freund[1], Gheorghe Păun[2], Grzegorz Rozenberg[3]

[1]Technical University Wien, Institute for Computer Languages
Resselgasse 3, 1040 Wien, Austria
email: freund@csdec1.tuwien.ac.at, tel. and fax: ++43 1 5041589

[2]Institute of Mathematics of the Romanian Academy of Sciences
Grivitei Str. 21, PO Box 1-764, 70700 Bucureşti
email: gpaun@imar.ro

[3]University of Leiden,Department of Computer Science
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands

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

1 Introduction

For the string case, the contextual grammars were introduced in [3] with motivations arising from descriptive linguistics. Basically, such a contextual grammar consists of a finite set of strings (*axioms*) and a finite set of productions of the form (s, c), where s is a string (the *selector*) and c is a pair of strings (the *context*). Starting from an axiom, iteratively contexts are added as it is indicated by the productions, which yields new strings. There are two modes of adjoining contexts: In the *external* case, contexts are added at the ends [3]; usually infinitely many productions (s, c), with finitely many contexts are used, otherwise the obtained language is finite. In the *internal* case, contexts are added in the interior of strings by bracketting one occurence of a selector with the associated context [4]; the generated language consists of all strings obtained in this way, by finitely but arbitrarily many context adjoinings.

Recently, several variants of these grammars were considered (e. g. [2], [5]). One variant we shall also consider here is the maximal mode of derivation (called t-mode in cooperating grammar systems [1]), i.e. we only retain the set of strings produced derivations which cannot be continued.

The fundamental difference between contextual grammars and Chomsky grammars or Lindenmayer systems is that in contextual grammars we cannot *rewrite* symbols, we only can *adjoin* symbols to the current string; once introduced, a symbol cannnot be modified any more.

2 Definitions

The contextual style of generating strings can be extended to arrays in a natural way: We start from a given finite set of arrays (axioms) and apply to them productions (s, c) where both s and c are finite patterns. If in the current array we can identify a subarray identical with s in such a way that the places corresponding to c are empty, then c is adjoined, producing a new array. In the generated language we may retain either all the arrays produced in this way or only the arrays to which no productions are applicable (maximal derivations).

The family of n-dimensional array languages of k-connected arrays generated by contextual array grammars with at most i symbols in the mode $f \in \{*, t\}$ is denoted by $L(n, cont, k, f, i)$. When no bound on the number of symbols is considered, we replace i by ∞, thus obtaining $L(n, cont, k, f, \infty)$; when no bound on the degree of connectedness is considered, we replace k by ∞, thus obtaining $L(n, cont, \infty, f, i)$.

In the same way, for all $n, k \in N$ (where N is the set of positive integers), the family of n-dimensional array languages of k-connected arrays generated by monotonic respectively context-free array grammars G is denoted by $L(n, mon, k)$ respectively $L(n, cf, k)$; when no bound on the degree of connectedness is considered, we replace k by ∞, thus obtaining $L(n, mon, \infty)$ respectively $L(n, cf, \infty)$. Obviously, for all $n, k \in N$ we obtain

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

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

Two n-dimensional arrays are called *shape-equivalent,* if they coincide in the positions of the n-dimensional space that they occupy by non-blank symbols; we say they have the same shape. Two n-dimensional array languages are called *shape-equivalent,* if the n-dimensional arrays in these two array languages induce the same set of shapes.

3 Results

Without proofs, we state some of the results we obtained.

Lemma 1. $L(n, cont, k, f, i) \subsetneq L(n, cont, k+1, f, i)$ for all $f \in \{*, t\}$, $k \in N$, and $i \in N \cup \{\infty\}$.

Lemma 2. For every $n \in N$ as well as all $k \in N \cup \{\infty\}$ and $i \in N$,

$L\,(n, cont, k, *, i) \subsetneq L\,(n, cont, k, *, i+1)\,,\; L\,(n, cont, k, t, i) \subsetneq L\,(n, cont, k, t, i+1)\,.$

Theorem 1. For every $n \in N$ as well as $k \in N \cup \{\infty\}$ and $i \in N$, each language in $L\,(n, cont, k, *, i)$ is shape-equivalent with a language in $L\,(n, cont, k, t, \infty)$.

Example 1. For $n \geq 2$, the converse of Theorem 1 is not true, e.g. the two-dimensional array language Sq of solid squares is in $L\,(2, cont, 2, t, 1)$, but no language in $L\,(2, cont, \infty, *, \infty)$ is shape-equivalent with Sq: In any contextual grammar we start from a finite set of axioms, and in order to generate an arbitrarily large square we must pass through stages when the current array cannot be a square; yet such arrays would belong to the n-dimensional array language generated in the arbitrary derivation mode for any contextual array grammar.

Lemma 3. Every one-dimensional array language in $L\,(1, cont, 1, t, 1)$ is finite. For $n \geq 2$, array languages in $L\,(n, cont, 1, t, 1)$ need not be finite.

Theorem 2. Let L be a set of 1-connected one-dimensional arrays over the single letter alphabet $\{a\}$. Then L is regular if and only if $L \in L\,(1, cont, \infty, *, 1)$.

Lemma 4. For every $n \geq 1$, there exists a language in $L\,(n, cont, 1, *, 1) - L\,(n, cont, 1, t, \infty)$.

Lemma 5. There exists an array language L in $L\,(2, cont, 2, *, 2)$ such that there is no array language L' in $L\,(2, cont, \infty, *, 1)$ that is shape-equivalent with L.

Lemma 6. There is an array language L in $L\,(2, cont, 2, t, 2)$ such that there is no context-free array grammar L' which is shape-equivalent with L.

Theorem 3. For every $n \in N$ and all $k \in N \cup \{\infty\}$, $L\,(n, cont, k, *, \infty) \subsetneq L\,(n, mon, k)$.

Theorem 4. For every $n \in N$ and all $k \in N \cup \{\infty\}$, $L\,(n, cont, k, t, \infty) \subsetneq L\,(n, mon, k)$.

The following is the main result of our paper:

Theorem 5. Every context-free array language in $L\,(n, cf, k)$ is the coding of a contextual array language in $L\,(n, cont, k, t, \infty)$, hence every context-free array language in $L\,(n, cf, k)$ is shape–equivalent with a contextual array language in $L\,(n, cont, k, t, \infty)$.

References

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

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

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

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

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

Minimierung von Muller-Automaten

Norbert Gutleben
Theoretische Informatik
BTU Cottbus
Postfach 101 344
03013 Cottbus

Für deterministische Muller–Automaten ist bislang noch kein effizienter Algorithmus bekannt, der einen äquivalenten zustandsminimalen Automaten erzeugt. Im allgemeinen gibt es keinen eindeutig bestimmten äquivalenten zustandsminimalen Automaten. Für reguläre Sprachen, die in der Borel–Hierarchie in $F_\sigma \cap G\delta$ liegen, wies Staiger(1993) jedoch nach, daßder Minimalautomat der eindeutig bestimmte zustandsminimalen Automaten ist. Wagner zeigte 1979, daßjeder deterministische Muller–Automat, der eine Sprache aus $F_\sigma \cap G\delta$ erkennt, eine recht einfache Struktur hat. Es werden zwei weitere Eigenschaften vorgestellt, die es dann erlauben, für deterministische Muller–Automaten, deren Sprache in $F_\sigma \cap G\delta$ liegen effizient zu minimieren:
Eine Menge $F \subseteq Q$, die

- Vereinigung von Zusammenhangskomponenten ist,
- Vereinigung von Klassen äquivalenter Zustände ist und
- mit q auch $\delta(q, a)$ und $\delta(q, b)$ enthält,

kann, falls sie von Q verschieden ist, so vergrößert werden, daßauch die Vergrößerung diese Eigenschaften hat und man mit Hilfe der Klassen äquivalenter Zustände, die Teilmenge von F sind, effizient die Klassen der Vergrößerung bestimmen kann.
Ausgehend von der leeren Menge kann man dann sukzessive die Äquivalenzklassen bestimmen und dann zu einem Zustand verschmelzen.

On the Complexities of Deterministic Indexed Grammars*

(Extended Abstract)

Markus Holzer and Klaus-Jörn Lange

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

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

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

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

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

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

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

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

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

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

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

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

References

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

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

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

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

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

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

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

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

Effiziente Speicherstrukturen für Turingmaschinen mit *einem* Kopf

Martin Hühne

Lehrstuhl Informatik II
Universität Dortmund
44221 Dortmund

huehne@ls2.informatik.uni-dortmund.de

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

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

Bekannte und vielfach untersuchte Speicherstrukturen sind das Band, das Gitter, und der Baum. Ein-Kopf Turingmaschinen mit der Speicherstruktur "Band" sind sehr ineffizient, da für die Simulation von t Schritten einer Mehrband Turingmaschine Zeit $O(t^2)$ benötigt wird [HS65]. Auf Ein-Kopf Turingmaschinen mit der Speicherstruktur "d-dimensionales Gitter", $d \geq 2$, kann die Simulation in Zeit $O(t^{1+1/d} \,/\, \log t)$ erfolgen [DH94]. Ein-Kopf Turingmaschinen mit der Speicherstruktur "binärer Baum" können Mehrband Turingmaschinen sogar in Zeit $O(t \log t)$ simulieren [PR81]. Diese Zeitschranken sind scharf. Schnellere deterministische Simulationen von Mehrband Turingmaschinen sind nur auf Modellen bekannt, die mehr als einen Kopf besitzen.

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

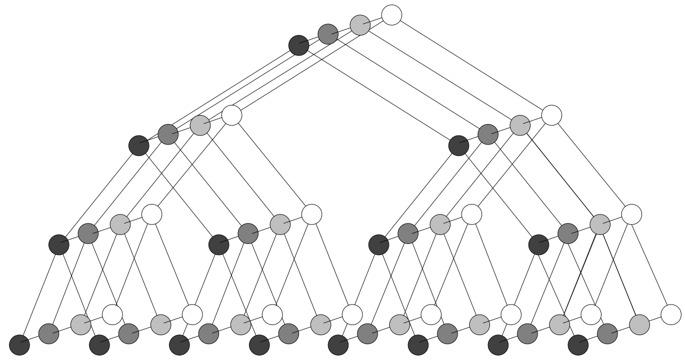
Man kann leicht eine Speicherstruktur definieren, in deren Kanten alle möglichen Berechnungen von Turingmaschinen kodiert sind. Sei $\mathcal{C} = \{L_1, L_2, L_3, \ldots\}$ eine abzählbare Menge von Sprachen über dem Alphabet $\{0,1\}$. Betrachte die baumartige Speicherstruktur mit Knotenmenge $\{0,1\}^*$,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

entscheiden — und zwar selbst dann, wenn einzelne oder gar alle Sprachen in $\mathcal{C}$ nicht rekursiv sind.

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

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

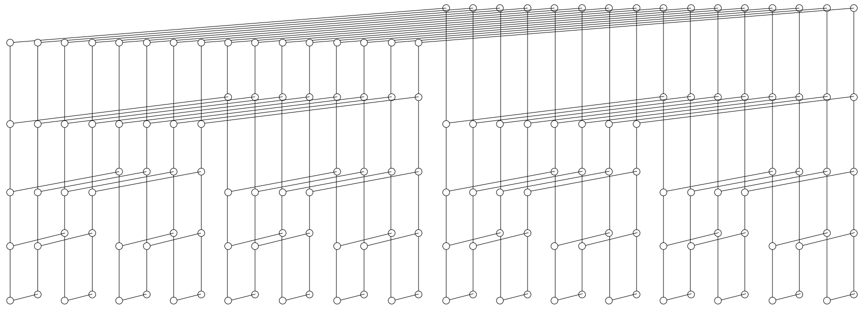
Die erste solche Speicherstruktur ist die "Hecke". Die Hecke läßt sich als das kartesische Produkt von Baum und Band beschreiben. Auf einer Ein-Kopf Turingmaschine mit Speicherstruktur "Hecke" können t Schritte von Mehrband Turingmaschinen in Zeit $O(t\,\log t \,/\, \log\log t)$ simuliert werden. Grundlage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



Ein Ausschnitt aus der Speicherstruktur "Hecke".

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

Auf der Hecke und auf dem CCC können auch Turingmaschinen mit mehreren anderen Speicherstrukturen effizient simuliert werden. So können t Schritte von Turingmaschinen mit mehreren d-dimensionalen Gittern auf einer Turingmaschine mit einer einzelnen Hecke in Zeit $O(t\,\log t \,/\, \sqrt[d]{\log\log t})$ und auf einer



Ein Ausschnitt aus der Speicherstruktur “Cube Connected Cycles”.

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

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

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

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

Literatur

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

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

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

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

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

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

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

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

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

Introducing a Two-Level Grammar Concept for Design*

(extended abstract)

Klaus P. Jantke

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

jantke@informatik.th-leipzig.de
www.informatik.th-leipzig.de

1 Introduction

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

1.1 Case-Based Reasoning

Case-based reasoning is deemed an important technology to alleviate the bottleneck of knowledge acquisition in Artificial Intelligence. Publications are mushrooming. Hence, it is almost impossible to give a reasonable overview. [RS89] and [Kol93] are widely accepted textbooks. There are several introductory papers like [Kol92], e.g. In case-based reasoning, knowledge is represented in the form of particular cases or episodes with an appropriate similarity measure relating them to each other rather than any form of generalized or abstracted

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

rules. One of the underlying key motivations is to avoid unjustified generalization and abstraction during knowledge acquisition. Instead, emphasis is put on adequately representing cases in a so-called case base. This is taken as a basis for further automated reasoning. When new problems are to be solved, a base-based reasoning system may search its case base for similar problems considered in the past. If more or less similar cases are found, the system may try to adapt them to the needs of the recent problem on hand. Thus, similarity based retrieval and adaptation are essential steps of case-based reasoning.

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

Recently, there is an enormous amount of approaches and applications. The majority of publications adopts an attribute-value-pair descrition of cases. In a sense, there is assumed some n-dimensional space for case representation. Every particular case is characterized by some possibly incomplete vector of attribute values. This is appropriate for a huge amount of application domains. But in many interesting domains posing exciting problems to Artificial Intelligence, representing cases in an attribute-value-pair style means already substantial abstraction and generalization (cf. [EOSW94] and [Sch95], e.g.). This essentially misses the original motivation of case-based reasoning approaches. Episodic knowledge should be kept in a form as close as possible to its appearence. Knowledge elicitation is widely underestimated, and investigations like [Gor92], [PAG92], and [RH92] are frequently disregarded in computation-oriented Artificial Intelligence.

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

There is a common motivation for approaches like in [Jan94] and [Bör94]) drawn from the authors' work within the BMFT joint project FABEL (cf. [FC93]).

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

1.2 Case-Based Learning

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

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

From the overwhelming scenarios of embedding inductive learning into case-based reasoning, a prototypical example should do for illustration. Assume an interactive design support system possessing its case base together with some similarity measure. Whenever a user is designing a new case, (s)he may ask the system for formerly solved problems similar to a specific problem presented. Usually, the system will respond to the request with one or more cases of maximal similarity. Let us assume that there is always some case among the most similar cases presented which fits the users needs, but which is usually not the most similar one. The user may whish to correct the underlying similarity concept in such a way that his preferred cases would be presented as proposals of highest similarity. This means to change this similarity concept by inductive learning. The input to the assumed learning device is quite clear. First of all, there is a current similarity concept to be modified incrementally. Second, there is a permanently changing case base collecting the system's experience. Third, there is the user's input consisting of pairs containing a recent problem together with a desired output case to be of highest similarity. A sequence of user inputs should result in a sequence of concept changes. In the best case, this may converge to an a priori unknown target similarity concept.

The scenario sketched is a typical instance of the general inductive inference scenario (cf. [AS83], [KW80], [Jan89]) of learning in the limit from usually incomplete information. In the present paper, we are more particularly faced to problems of language learning (cf. [Vid94]). The approach is even more particular, as we are interested in two-level grammar concepts for generating graphs (see below). There are several further motivations of investigating two-level grammars, especially in computational linguistics (cf. [Kos83], e.g.). Inductive learning of two-level grammars is restricted two word languages, so far, like in [Ita94].

The present paper does not deal with the development of inductive learning for advanced case-based reasoning. Its intention is to introduce an appropriate graph-theoretic concept flexible and expressive enough to reflect essential cases of the FABEL application domain. The precision of the introduced concepts and their relation to other formal language concepts should provide a firm background for the intended development and application of inductive learning methods.

2 Structural Case Representation in Design

The present papers addresses case-based reasoning applications where the main objects of reasoning are highly structured. In those domains, exploiting the internal structures of objects is usually crucial.

Many computer science applications are substantially suffering from what the author calls the *paradoxon of levelling*. Although objects in the real world are usually very complex, have a certain hierarchical structure or internal topology, knowledge representation on computers is initially levelling this structure down, in most cases. In later phases of knowledge processing, enormous efforts are necessary to reconstruct the structural information levelled before. This seems to be somehow paradox.

Our underlying FABEL project is an excellent example in this regard. Typical target objects considered within the FABEL application are fresh air supply nets of buildings, for instance. In reality, these nets are highly structured, consist of certain levels both due to stages of architectural design and due to types of constructive parts, have an obvious topology, and are reasonably understood as graphs. However, as an effect of using a particular CAD system during design, these objects are levelled down to lists of tupels representing a bunch of attribute/value pairs. This leads to the somehow crazy necessity to rebuild structured objects from flat representations. In [Bör94] there is sketched the problem of transforming tupel representations back into structured objects for an appropriate case-based knowledge processing.

The present paper focusses on structural concepts for appropriate case-based reasoning, particularly in architectural design. It is an implementation decision whether or not to make a detour via flat case representations when invoking the concepts developed. This may essentially depend on the available hard- and software tools. However, those considerations are beyond the present investigation.

2.1 Case Structures in FABEL

In the FABEL approach (cf. [FK93] and [FC93]), we are dealt with certain architectural design problems. Typical applications are designing complete air supply systems of buildings. The net structures to be created interactively may also represent electricity or water supply, for instance. Characteristically, those nets are huge structures of a high regularity which may be reasonably understood as graphs.

2.2 Graph Objects in Architectural Design

The recent scientific literature yields an extremely inhomogeneous impression of the area. There are a score or more approaches like the one in [Bro94] which sound quite promising, but do not provide any precision. An extreme example

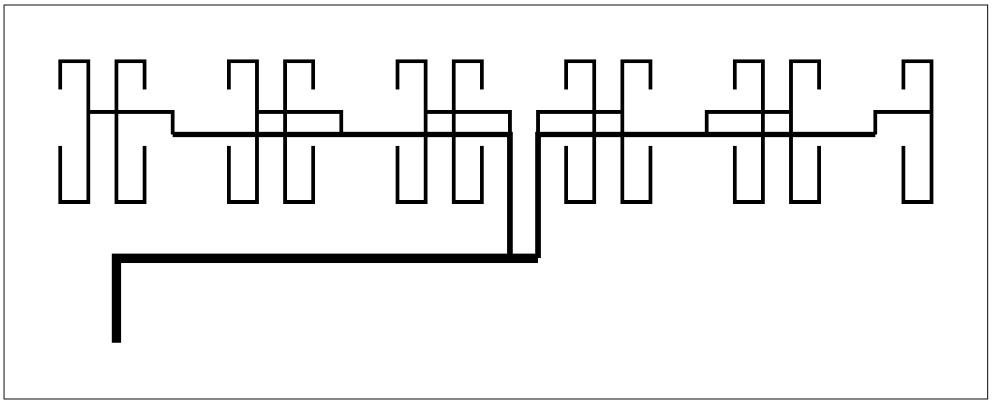


Figure 1: Part of a Typical Fresh-Air Supply Structure

is [Lee94] which announces "the formal basis ...", but does not contain any formalism.

In a minority of papers, one may find well-explained concepts of a certain precision. A valuable introduction leading to grammars for defining topological changes may be found in [HSF+92]. There, the authors have developed a clearly motivated two-level concepts of grammars for architectural design. The languages in use are essentially regular. It is one of the key intentions of the present paper to generalize this approach considerably.

3 Two-Level Design Grammars

The present chapter is devoted to the introduction of two-level graph grammar concepts. On the upper level, classical concepts of the Chomsky hierarchy are invoked. On the lower level, the ideas of string patterns is adopted. For the classical formal language concepts, there is a large amount of topical publications. [HU79] is an appropriate reference. The basic concepts of pattern languages have been introduced by Dana Angluin in [Ang80b].

The following figure illustrates the skeleton of some target graph to be described by conventional grammar rules.

In dependence on the underlying application, those rules may be regular or context free. The resulting graph usually possesses some clearly marked nodes which are to be completed by attaching certain subgraphs taken from a class of more or less well-defined prototypes. This process of completion is somehow similar to the process of variable substitution for pattern languages (cf. [Ang80b]). The next figure is intended to display the "substitution" of "variables" implicitly contained in the figure above.

The ultimate goal of the overall approach behind the present paper is to learn appropriate grammatical concepts from actual design processes. This needs clear formalisms to be outlined here.

First, our basic ideas are developed for string languages, for readability. It

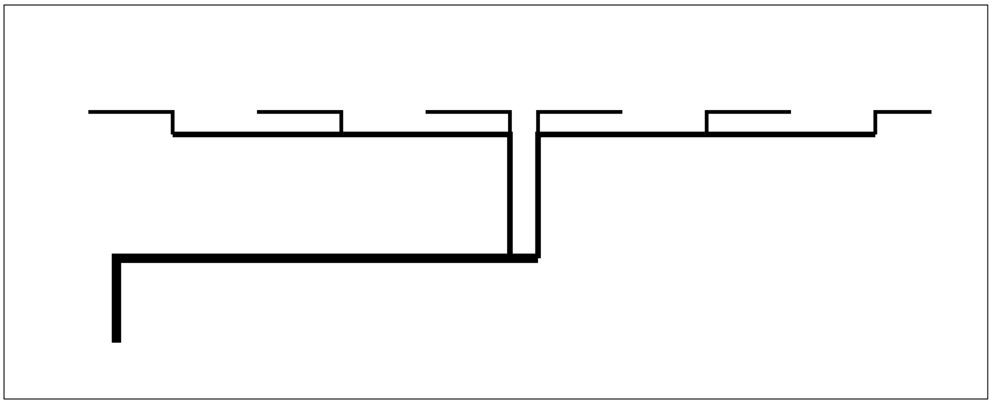


Figure 2: The CHOMSKY Part of a Target Graph

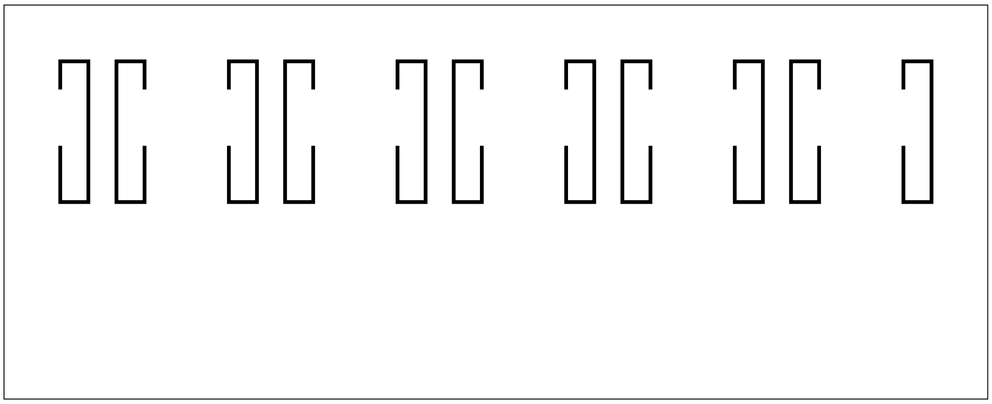


Figure 3: The Pattern Part of a Target Graph

is assumed that most readers are familiar with the fundamentals of classical formal language theory. Thus, the introduction of our key concepts may look like an easy tour beyond the boundaries of classical theory. Second, they will be straightforwardly generalized to cover graph concepts

3.1 String Languages

The presentation follows classical concepts as tight as possible. When introducing graph grammars below, it seems reasonable to restrict our attention to only context-free substitutions. Therefore, we will apply the same restriction to the string language case discussed first.

Definition 1

A two-level context-free design grammar $\mathcal{DG} = [N, T_1, V, T_2, R, s]$ is given by the following parameters:

- N is a non-empty set (the so-called nonterminals).
- T_1 is a (possibly empty) set (the so-called first-level terminals).

- V is a non-empty set (the so-called pattern variables).
- T_2 is a non-empty set (the so-called second-level terminals).
- For R it holds $R \subseteq N \times (N \cup V \cup T_1)^+$ (the rule system of the grammar).
- For s it holds $s \in N$ (the start symbol of the grammar).

A design grammar $\mathcal{DG} = [N, T_1, V, T_2, R, s]$ is said to be finite, if all sets N, T_1, V, T_2, and R are finite. $\square$

Usually, grammars are used to define languages. We will introduce a formal semantics specifing for every grammar $\mathcal{DG}$ its formal language $L(\mathcal{DG})$. The standard concepts of the CHOMSKY hierarchy are assumed. Additionally, we invoke the basic concepts of pattern languages (cf. [Ang80b]).

In particular, the following notions and notations are in use:

- For every CHOMSKY grammar $\mathcal{G}$ on any alphabet A, $L(\mathcal{G})$ denotes the formal language generated by $\mathcal{G}$ over A.
- For every pattern p on any alphabet A and for any set of variables X disjoint with the alphabet A, $L(p)$ denotes the pattern language generated by p via non-erasing substitutions over A.

Definition 2

The formal semantics of any context-free design grammar $\mathcal{DG} = [N, T_1, V, T_2, R, s]$ is some formal language $L(\mathcal{DG})$ on the alphabet $T_1 \cup T_2$ defined by

$$L(\mathcal{DG}) \;=\; \bigcup_{p \in L(\mathcal{G})} L(p)$$

where

- $\mathcal{G} = [N, V \cup T_1, R, s]$ is a CHOMSKY grammar and
- every string $p \in L(\mathcal{G})$ is a pattern with variables in V varying over T_2 .

For any given alphabet A, the notation $\mathcal{L}_{DG}$ means the class of all formal languages defined by any context-free design grammar over $A = T_1 \cup T_2$. $\square$

One could easily introduce a couple of requirements of naturalness. For instance, if $V \neq \emptyset$ and $L(\mathcal{G}) \setminus T_1^* \neq \emptyset$, one should reasonably assume $T_2 \neq \emptyset$. Vice versa, if V is empty, one does not need any letter in T_2. The investigation of dependencies of this type is left to the reader as an introductory excercise.

For extremely simple special cases, one gets well-known concepts. For instance, $V = \emptyset$ implies that $L(\mathcal{DG})$ is a usual CHOMSKY language. If $L(\mathcal{G})$ is a singleton set, $L(\mathcal{DG})$ is a standard pattern language. Already in the simple case where $L(\mathcal{G})$ is finite, we are faced to a type of formal languages not investigated so far.

There is a couple of reasonable generalizations and refinements of the basic approach introduced above. For example, one may generalize the approach to multi-layer grammars. Furthermore, one may refine the approach by assigning particular alphabets to different variables in V, or by guarding substitutions. In the author's opinion, one should stick with the basic approach for some time. First, this approach seems to meet the needs of the driving application described above quite well. There is no obvious need to develop more than two levels. Second, one should try to solve some of the open problems in the simplest setting before attacking more general cases. Hence, the approach will not be generalized throughout this paper.

A few investigation may first illustrate the differences of our new concepts from known hierarchies.

Even if $L(\mathcal{G})$ is regular, $L(\mathcal{DG})$ may be not context-free.

There is no need for a proof as it is folklore that pattern languages are usually not context-free.

We present a few further trivialities to warm up:

In case $|V| = 1$, $L(\mathcal{DG})$ is regular if and only if $L(\mathcal{G})$ is regular.

If T_1 and T_2 are disjoint, it holds:

1. Membership for $L(\mathcal{DG})$ is uniformly decidable.
2. Emptyness for $L(\mathcal{DG})$ is uniformly decidable.
3. Finiteness for $L(\mathcal{DG})$ is uniformly decidable.

There is no need for an explicit proof. The results above are immediately inherited from classical results in formal language theory (cf. [HU79]).

Further properties seem to require particular assumptions. For the intended application domain, languages which satisfy $T_1 \setminus T_2 \neq \emptyset$ are of a particular importance. It may also be of a special interest to consider languages with a certain rate of letters from $T_1 \setminus T_2$ in terminal words.

3.2 Graph Grammars

The area of graph grammar is quite rich. First, one needs to specify whether the generation process should be based on node replacements or edge replacement (or even hyperedge replacements). The FABEL application does no govern this decision in any way. Throughout this paper, we deal with node substitutions, exclusively.

A key question is how to embed graphs when inserted for a node of a given host graph. The embedding rule essentially determines the peculiarities of the whole approach. In the case of undirected graphs, there is an enormous number of concurring ideas.

In the present paper, we assume directed acyclic graphs. This approach has two basic roots. First, it is a direct generalization of approaches for two-level word grammars as sketched in [HSF+92]. Second, it is based on graph grammar concepts mainly proposed by Oksana Arnold in [AJ94b], [AJ94c] and [JA95]. Although these grammatical concepts are originally designed to meet the necessities of therapy plan generation for complex dynamic processes, they fit our present needs quite well.

The non-terminal symbols of the above definitions collected in N correspond to compound vertices in C_i below. Analoguously to the above set of variables V, there will be sets of variables X_i in every graph.

For the following definitions, there are two basic alternatives for representing what corresponds to the start symbol of Chomsky grammars: One may either choose a node or a graph. The choice of a graph would directly correspond to the underlying approach in [AJ94b], [AJ94a] and [JA95]. However, some reader may find it more intuitive to have a start object which is properly atomic. As this decision is not essential, we have chosen the syntactically simpler second alternative.

Definition 3

A two-level context-free graph design grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$ is given as follows:

1. $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$ is a set of so-called pin graphs.
2. $\mathcal{H} = \{\mathcal{G}_{m+1}, \ldots, \mathcal{G}_n\}$ is another set of pin graphs.
3. Every pin graph in $\mathcal{F}$ has the form $\mathcal{G}_i = [V_i, E_i, P_i^{in}, P_i^{out}, C_i, sub_i, X_i]$.
4. Every pin graph in $\mathcal{H}$ has the form $\mathcal{G}_i = [V_i, E_i, P_i^{in}, P_i^{out}]$.
5. Every pin graph $\mathcal{G}_i$ in $\mathcal{F} \cup \mathcal{H}$ with $i \in \{1, \ldots, n\}$ meets the following conditions:

   5.1 $\mathcal{G}_i' = [V_i, E_i]$ is a finite, directed, acyclic graph with the set V_i of vertices and the set E_i of edges.

   5.2 $P_i^{in} \cup P_i^{out}$ contains the so-called input pins and output pins of $\mathcal{G}_i$ defined as follows:

   5.3 $P_i^{in} = \{ v \mid v \in V_i \wedge \neg\exists u \in V_i ((u, v) \in E_i) \}$

   5.4 $P_i^{out} = \{ v \mid v \in V_i \wedge \neg\exists u \in V_i ((v, u) \in E_i) \}$

6. Every pin graph $\mathcal{G}_i$ in $\mathcal{F}$ with $i \in \{1, \ldots, m\}$ meets the following conditions:

   6.1 The vertices in $C_i \subseteq V_i$ are called compound.

   6.2 $sub_i : C_i \rightarrow 2^{\{1,\ldots,m\}} \setminus \emptyset$ defined the admissible substitutions of graphs $\mathcal{G}_j$ for compound vertices in C_i.

   6.3 The vertices in $X_i \subseteq V_i$ are called variables. For all $i \in \{1, \ldots, m\}$ it holds $C_i \cap X_i = \emptyset$.

7. $c_0 \in \bigcup_{i=1\ldots m} C_i$ is some start vertex. $\square$

Obviously, the specification of P_i^{in} and P_i^{out} is redundant. It is given exclusively for clarity.

The pin graphs in $\mathcal{H}$ can be understood as special cases of graphs in $\mathcal{F}$ with $C_i = \emptyset$ and $X_i = \emptyset$. The property $C_i = \emptyset$ implies that the mapping sub_i (understood as a relation) must be empty. Intuitively, the graphs in $\mathcal{F}$ are to be substituted one into another according to $\{sub_i\}_{i=1\ldots m}$, whereas the graphs in $\mathcal{H}$ are used for replacing the varaibles in $\{X_i\}_{i=1\ldots m}$.

For illustration, recall figure 3 above. It depicts 2 pin graphs (one occurs in 5 copies) of $\mathcal{H}$. The graph of figure 2 belonging to $\mathcal{F}$ contains variables in 6 places. A possible substitution yields the graph originally displayed in figure 1.

Decisions about the substitution mechanism, i.e. embedding, define the underlying formal semantics. As introduced before, there isa two-level mechanism. For understandability, it is split into the following concepts:

- a rewrite relation,
- a language of pattern graphs,
- graph substitution, and
- a language of ground graphs.

There are some useful notations. For example, there is some desire to avoid name conflicts during substitution. Therefore, the following concentions will be adopted:

- If c and d are names of vertices (perhaps, in different graphs), then $c.d$ is an acceptable name of a vertex as well.
- Similarly, this notation is applied to pairs of vertices: $c.(d_1, d_2) = (c.d_1, c.d_2)$
- If c is a vertex and C a set of vertices, then holds $c.C = \{c.d \mid d \in C\}$.

Names of vertices of the form "c.d" are said to be structured. Originally, there are no structured names.

Definition 4

Assume any two-level context-free graph design grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$. For any graph $\mathcal{G}_i \in \mathcal{F}$, any vertex $c \in C_i$ and any index $j \in sub_i(c)$, the *Substitution of* $\mathcal{G}_j$ *in* $\mathcal{G}_i$ *at position* $c \in C_i$ yields a graph $\mathcal{G}_i[c \hookleftarrow \mathcal{G}_j]$ where $\mathcal{G}_i[c \hookleftarrow \mathcal{G}_j] = [V, E, P^{in}, P^{out}, C, sub, X]$ is defined by:

1. $V \;=\; (\,V_i \setminus \{c\}\,) \cup c.V_j$

2. $E \;=\; (\,(E_i \cup c.E_j) \setminus (V_i \times \{c\} \cup \{c\} \times V_i)\,)$
$\qquad \cup\,((\{v \mid (v,c) \in E_i\} \times c.P_j^{in}) \cup (c.P_j^{out} \times \{v \mid (c,v) \in E_i\}))$

3.1 $P^{in} \;=\; P_i^{in} \qquad\qquad$ if $c \notin P_i^{in}$

3.2 $P^{in} \ = \ (\, P_i \setminus \{c\}\,) \ \cup \ c.P_j^{in}$ if $c \in P_i^{in}$

4.1 $P^{out} \ = \ P_i^{out}$ if $c \notin P_i^{out}$

4.2 $P^{out} \ = \ (\, P_i \setminus \{c\}\,) \ \cup \ c.P_j^{out}$ if $c \in P_i^{out}$

5. $C \ = \ (\, C_i \setminus \{c\}\,) \cup c.C_j$

6. $sub \ = \ sub_{i/C_i \setminus \{c\}} \ \cup \ c.sub_j$

7. $X \ = \ X_i \ \cup \ c.X_j$ □

The resulting pin graph $\mathcal{G} = \mathcal{G}_i[c \hookleftarrow \mathcal{G}_j]$ is of the same structure as the graphs in $\mathcal{F}$. Therefore, the definition above applies to those generated host graphs $\mathcal{G}$, for arbitrary of their vertices c, and for all graphs $\mathcal{G}_j$ in $\mathcal{F}$ as well. This allows to write $\mathcal{G}[c \hookleftarrow \mathcal{G}_j]$ even if $\mathcal{G}$ is the result of any preceeding substitution process.

Definition 5

Among pin graphs of the structure $[V, E, P^{in}, P^{out}, C, sub]$, one defines a rewrite relation $\Rightarrow_{\mathcal{F}}$ by

1. $\mathcal{G} \ \Rightarrow_{\mathcal{F}} \ \mathcal{G}'$ if and only is $\exists c \ \exists j \ (\, \mathcal{G}' = \mathcal{G}[c \hookleftarrow \mathcal{G}_j]\,)$

One may interpret the vertex c_0 as a graph consisting of only the one (compound) vertex c_0 for justifying the notation $c_0 \ \Rightarrow_{\mathcal{F}} \ \mathcal{G}$.

2. $c_0 \ \Rightarrow_{\mathcal{F}} \ \mathcal{G}$ gilt if and only if
   $\exists i \in \{1, \ldots, m\} \ \exists j \in sub_i(c_0) \ (\, c_0 \in C_i \ \wedge \ \mathcal{G} = \mathcal{G}_j\,)$

$\Rightarrow_{\mathcal{F}}^{+}$ denotes the transitive closure of $\Rightarrow_{\mathcal{F}}$ and the irreducibility of any graph $\mathcal{G}$ with respect to $\Rightarrow_{\mathcal{F}}$ is indicated by $\mathcal{G}\!\downarrow_{\mathcal{F}}$. □

This is sufficient for defining the language of patterns generated by any given two-level context-free graph design grammar. The graph depicted in figure 2 is a typical element of such a language, for instance.

Definition 6

Assume any two-level context-free graph design grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$. This defines a language $\mathcal{P}(\mathcal{GDG})$ as follows:

- $\mathcal{P}(\mathcal{GDG}) \ = \ \{\, \mathcal{G} \mid c_0 \ \Rightarrow_{\mathcal{F}}^{+} \ \mathcal{G} \ \wedge \ \mathcal{G}\!\downarrow_{\mathcal{F}} \}$ □

The following technical concepts seem quite appropriate.

Definition 7

Assume any grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$. The set of its variables is

1. $X_{\mathcal{GDG}} \ = \ \bigcup_{i=1\ldots m} X_i$

The final segment relation $\sqsupseteq$ of vertices is introduced as follows.

2. $\forall d, e\ (\ e \sqsupset_1 d \iff \exists c\ (\ e = c.d\ )\ )$

3. $\sqsupseteq_1$ denotes the reflexive closure of $\sqsupset_1$.

4. $\forall d, e\ (\ e \sqsupseteq d \iff e = d\ \vee\ \exists c\ (\ e \sqsupset_1 c\ \wedge\ c \sqsupseteq d\ )\ )$

ν computes the last letter of any vertex name via $\sqsupseteq$.

5. $\forall c, d\ (\ d = \nu(c) \iff c \sqsupseteq d \wedge\ \nexists e\ (\ d \sqsupset e\ )\ ).$ □

Structured vertices c which have been constructed during stepwise substitution are finally identified with its last letter $\nu(c)$. First, the substitution concept is introduced.

Definition 8

Assume any two-level context-free graph design grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$ with the language of patterns $\mathcal{P}(\mathcal{GDG})$. A substitution σ is some mapping

1. $\sigma\ :\ X_{\mathcal{GDG}} \longrightarrow \{m+1, \ldots, n\}$

Every substitution σ specifies a non-deterministic rewrite operator $\Rightarrow_\sigma$ on $\mathcal{P}(\mathcal{GDG})$.

2. $\mathcal{G} \Rightarrow_\sigma \mathcal{G}'$, if $\exists c\ (\ \mathcal{G}' = \mathcal{G}[c \hookleftarrow \mathcal{G}_{\sigma(\nu(c))}]\ )$

3. $\Rightarrow_\sigma^+$ denotes the transitive closure and $\Rightarrow_\sigma^*$ denotes the reflexive and transitive closure of $\Rightarrow_\sigma$.

4. $\mathcal{G}\!\downarrow_\sigma$ means that $\mathcal{G}$ is irreducible w.r.t. $\Rightarrow_\sigma$.

The target concept to be developed in the present paper is the formal graph language generated by any given two-level context-free graph design grammar.

Definition 9

Assume any two-level context-free graph design grammar $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$ with the language of patterns $\mathcal{P}(\mathcal{GDG})$. $\mathcal{GDG} = [\mathcal{F}, \mathcal{H}, c_0]$ defines a formal language $\mathcal{L}(\mathcal{GDG})$ by

$$\mathcal{L}(\mathcal{GDG}) \ = \bigcup_{\substack{\mathcal{G}\in\mathcal{P}(\mathcal{GDG})\\ \sigma\in X_{\mathcal{GDG}}}} \sigma(\mathcal{G})$$

The two levels of the derivation process are easy to see. For generating any graph of $\mathcal{L}(\mathcal{GDG})$, one first generates a pattern graph describing the skeleton of the target graph which belongs to $\mathcal{P}(\mathcal{GDG})$. Second, the final graph is constructed by an appropriate variable substitution.

This process suggests several similarity concepts. The simplest one is to consider two (or even more) graphs as somehow similar, exactly if they have the same skeleton. Another interesting approach is to choose the largest common subgraph of two skeletons as the similarity of two given graphs. A quite opposite approach compares the similarities of final substitutions.

Furthermore, the novel formalism developed may be taken as a firm basis for learning-theoretic investigations. One may try to learn one level of the grammar under the assumption of the other one. And one may try to learn similarity concepts for given two-level context-free graph design grammars.

Despite the attractiveness of these questions, the reader should recall that the aim of the present extended abstract was exclusively to set the stage both for the case-based reasoning investigations and for the learning-theoretic investigations by providing a well-motivated innovative concept of graph grammars and languages.

References

[AJ94a] Oksana Arnold and Klaus P. Jantke. Therapy plan generation as program synthesis. In Setsuo Arikawa and K.P. Jantke, editors, *Algorithmic Learning Theory, AII'94 & ALT'94*, volume 872 of *LNAI*, pages 40–55. Springer–Verlag, 1994.

[AJ94b] Oksana Arnold and Klaus P. Jantke. Therapy plans as hierarchically structured graphs. WISCON Report 02/94, HTWK Leipzig (FH), Fachbereich IMN, April 1994.

[AJ94c] Oksana Arnold and Klaus P. Jantke. Therapy plans as hierarchically structured graphs. In *Fifth International Workshop on Graph Grammars and their Application to Computer Science, Williamsburg, Virginia, USA*, November 1994.

[Ang80a] Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[Ang80b] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.

[AS83] Dana Angluin and Carl H. Smith. A survey of inductive inference: Theory and methods. *Computing Surveys*, 15:237–269, 1983.

[Bör94] Katy Börner. Structural similarity as guidance in case-based reasoning. In Stefan Wess, Klaus-Dieter Althoff, and Michael M. Richter, editors, *1st European Workshop on Case–Based Reasoning (EWCBR'93), November 1-5, 1993, Selected Papers*, volume 837 of *Lecture Notes in Artificial Intelligence*, pages 197–208. Springer-Verlag, 1994.

[Bro94] Ken N. Brown. Shape grammars and semantics: Formalising the link between geometry, features and the design process. In *Third International Conference on Artificial Intelligence in Design, Lausanne, Switzerland, August 15–18, 1994, Workshop on Reasoning with Shapes in Design, Workshop Notes*, pages 1–4, 1994.

[EOSW94] Dieter Ehrenberg, Thomas J. Olbrich, Ralf Schulz, and Herbert Wiesner. Distributed reasoning support systems – a case-based approach to solving ill-structured decision problems. Arbeitsbericht 5, Universität Leipzig, Wirtschaftswissenschaftliche Fakultät, Lehrstuhl Wirtschaftsinformatik, May 1994.

[FC93] FABEL-Consortium. Survey of FABEL. FABEL–Report 2, Gesellschaft für Mathematik und Datenverarbeitung mbH, Forschungsbereich Künstliche Intelligenz, February 1993.

[FK93] Fabel-Konsortium. FABEL im Überblick. FABEL–Report 1, Gesellschaft für Mathematik und Datenverarbeitung mbH, Forschungsbereich Künstliche Intelligenz, February 1993.

[Gor92] Dallie E. Gordon. Implication of cognitive theory for knowledge acquisition. In Robert R. Hoffman, editor, *The Psychology of Expertise. Cognitive Research of Empirical AI*, pages 99–120. Springer-Verlag, 1992.

[HSF+92] K. Hua, I. Smith, B. Faltings, S. Shih, and G. Schmitt. Adaptation of spatial design cases. In John S. Gero, editor, *Artificial Intelligence in Design'92*, pages 559–575. Kluwer Academic Publ., 1992.

[HU79] John E. Hopcroft and Jeffrey. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[Ita94] Alon Itai. Learning morphology – practice makes good. In Rafael C. Carrasco and Jose Oncina, editors, *Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, September 21–23, 1994*, volume 862 of *Lecture Notes in Artificial Intelligence*, pages 5–15. Springer–Verlag, 1994.

[JA95] Klaus P. Jantke and Oksana Arnold. Graphgrammatik-Konzepte in der Therapieplanung für komplexe dynamische Prozesse. In Markus Holzer, editor, *4. GI Theorietag "Automaten und Formale Sprachen", 1994, Proc., Herrsching*, pages 42–47. Universität Tübingen, Wilhelm-Schickard-Institut, 1995.

[Jan89] Klaus P. Jantke. Algorithmic learning from incomplete information: Principles and problems. In J. Dassow and J. Kelemen, editors, *Machines, Languages, and Complexity*, Lecture Notes in Computer Science, pages 188–207. Springer-Verlag, 1989.

[Jan92] Klaus P. Jantke. Case based learning in inductive inference. In *Proc. 5th Annual ACM Workshop on Computational Learning Theory, (COLT'92), July 27-29, 1992, Pittsburgh, PA, USA*, pages 218–223. ACM Press, 1992.

[Jan94] Klaus P. Jantke. Nonstandard concepts of similarity in case-based reasoning. In Hans-Hermann Bock, Wolfgang Lenski, and Michael M. Richter, editors, *Information Systems and Data Analysis:*

Prospects – Foundations – Applications, Proceedings of the 17th Annual Conference of the GfKl, Univ. of Kaiserslautern, 1993, Studies in Classification, Data Analysis, and Knowledge Organization, pages 28–43. Springer-Verlag, 1994.

[JL93] Klaus P. Jantke and Steffen Lange. Case–based representation and learning of pattern languages. In K.P. Jantke, S. Kobayashi, E. Tomita, and T. Yokomori, editors, *4th Intern. Workshop on Algorithmic Learning Theory (ALT'93), Proc., November 1993, Tokyo, Japan*, volume 744 of *Lecture Notes in Artificial Intelligence*, pages 87–100. Springer–Verlag, 1993.

[Kol92] Janet L. Kolodner. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6:3–34, 1992.

[Kol93] Janet L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[Kos83] K. Koskienniemi. Two-level morphology: A general computational model for word-form recognition and production. Technical report, Department of General Linguistics, University of Helsinki, Publication No. 11, 1983.

[KW80] Reinhard Klette and Rolf Wiehagen. Research in the theory of inductive inference by GDR mathematicians - a survey. *Information Sciences*, 22:149–169, 1980.

[Lak87] George Lakoff. *Women, Fire, and Dangerous Things*. The University of Chicago Press, 1987.

[Lee94] John R. Lee. The formal basis of "reasoning with shapes". In *Third International Conference on Artificial Intelligence in Design, Lausanne, Switzerland, August 15–18, 1994, Workshop on Reasoning with Shapes in Design, Workshop Notes*, pages 30–34, 1994.

[PAG92] David S. Prerau, Mark R. Adler, and Alan S. Gunderson. Eliciting and using experiential knowledge and general expertise. In Robert R. Hoffman, editor, *The Psychology of Expertise. Cognitive Research of Empirical AI*, pages 137–148. Springer-Verlag, 1992.

[RH92] Stephen B. Regoczei and Graeme Hirst. Knowledge and knowledge acquisition in the computational context. In Robert R. Hoffman, editor, *The Psychology of Expertise. Cognitive Research of Empirical AI*, pages vii–ix. Springer-Verlag, 1992.

[Ros75] Eleanor Rosch. Cognitive reference points. *Cognitive Psychology*, 7:531–547, 1975.

[RS89] Christopher K. Riesbeck and Roger C. Schank. *Inside Case-Based Reasoning*. Lawrence Erlbaum Assoc., 1989.

[Sch95] Ralf Schulz. Handling similarity in case-based systems for ill-structured problems. Arbeitsbericht 7, Universität Leipzig, Wirtschaftswissenschaftliche Fakultät, Lehrstuhl Wirtschaftsinformatik, January 1995.

[SJL94] Yasubumi Sakakibara, Klaus P. Jantke, and Steffen Lange. Learning languages by collecting cases and tuning parameters. In Setsuo Arikawa and K.P. Jantke, editors, *Algorithmic Learning Theory, AII'94 & ALT'94*, volume 872 of *LNAI*, pages 533–547. Springer-Verlag, 1994.

[Vid94] Enrique Vidal. Grammatical inference: An introductory survey. In Rafael C. Carrasco and Jose Oncina, editors, *Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, September 21–23, 1994*, volume 862 of *Lecture Notes in Artificial Intelligence*, pages 1–4. Springer–Verlag, 1994.

Properties of formal Languages of Therapy Plans created by Graph Grammars (short overview)*

Daniel Kirsten
Technische Hochschule Leipzig
91 AI 2
EMail: kirsten@imn.th-leipzig.de

1 Motivation

The paper deals about a special type of graph grammar used in therapy planning in complex dynamic systems in the Joint Project (BMFT-Verbundprojekt) WISCON .
Plans are represented as finite, directed, acyclic graphs, called pin graphs.
A planner (a program) has to works on plans: creation, evaluation, simulation, revision... That means the planner has to create and to evaluate pin graphs.
Therefore Dipl.-Ing. Oksana Arnold has developed a graph grammar concept suitable for therapy planning. The core is the *rooted family* [1][2].
It is based on vertex rewriting, i.e. single vertices are replaced by complex graphs like deriving in context-free languages.
The basic idea of planning is the following: The planner starts at a pin graph called starting graph. It contains two kinds of vertices: elementary vertices and compound vertices like terminals and nonterminals in CHOMSKY-languages. The planner replaces compound vertices by complex pin graphs wich may also contain compound vertices. Usually there are several pin graphs wich can replace a compound vertex. The planner inserts and evaluates pin graphs. So the search space of the planner is a set of pin graphs created by a graph grammar, or better it is a language of graphs.
The usage of graph grammars leads also to problems. Is the rooted family really suitable for therapy planning? May be there is an optimal plan to solve a problem, but the graph grammar is only able to create worse plans. May be there are several plans solving a problem, but the graph grammar is unable to

*Complete paper will be available under
http://www.informatik.th-leipzig.de/~kirsten/inf/index.html

create one of them. We deal therapy planning, so human life can depend on a graph grammar.
Therefore a basic research on languages creatable by rooted families was necessary! We call these languages context-free.

- How is the *power* of context-fee languages? I.e. are all recursively enumerable languages context-free, or only special cases?
- Are there interesting languages wich are not context-free ?
- Are there plans wich can not be created by a rooted family ?
- Are context-free languages stable under complement, intersection and union?
- How can I proof that a language is not contect-free?

2 Introduction

A short introduction to the basic concepts.

Definition 1.
A *hierarchically structured family of plans* is a finite set of pin graphs where: $\mathcal{G}_i = [V_i, E_i, P_i^{in}, P_i^{out}, C_i, sub_i]$ $(i = 1 \dots k)$, such that for all $i \in \{1 \dots k\}$:

1. $[V_i, E_i]$ is a nonempty, finite, directed, acyclic graph.
2. $P_i^{in} := \{v | v \in V_i \quad \wedge \quad \neg\exists u \in V_i((u,v) \in E_i)\}$
   $P_i^{out} := \{v | v \in V_i \quad \wedge \quad \neg\exists u \in V_i((v,u) \in E_i)\}$
3. $C_i \subseteq V_i$
4. $sub_i : C_i \to (2^{\{1 \dots k\}} \setminus \{\emptyset\})$ □

We assume that the vertices are letters of an alphabet !

Substitutions:
given: $\mathcal{F} := \{\mathcal{G}_1 \dots \mathcal{G}_k\}$
$\mathcal{G} := \mathcal{G}_i[c \hookleftarrow \mathcal{G}_j]$
Presumption: $c \in V_i$
Definition of $\mathcal{G}$:
1. $V := (V_i \setminus \{c\}) \cup c.V_j$
2. $E := ((E_i \cup c.E_j) \setminus (V_i \times \{c\}) \setminus (\{c\} \times V_j) \cup (\{v | (v,c) \in E_i\} \times c.P_j^{in})$
$\cup\, (c.P_j^{out} \times \{v | (c,v) \in E_i\})$
3. P^{in} , P^{out} already defined.
4. $C := (C_i \setminus \{c\}) \cup c.C_j$

6. $sub := sub_{j/C_i\setminus\{c\}} \cup c.sub_j$

$c.V$ simply means set a c in front of each vertex in V . This is necessary to get disjoint sets of vertices. So the vertices grow to words.
Substitution steps are commutativ and associativ. We may denote several substitutions (paths) by $\mathcal{G}[c_1 \hookleftarrow \mathcal{G}_3][c_2 \hookleftarrow \mathcal{G}_6][c_1c_3 \hookleftarrow \mathcal{G}_{42}]$

The deriving relation is defined based on substitution:

$$\mathcal{G}_i \Rightarrow_{\mathcal{F}} \mathcal{G} \quad \Longleftrightarrow \quad \exists c \in \mathcal{G}_i \; \exists j \in sub_i(c) \; : \; \mathcal{G} = \mathcal{G}_i[c \hookleftarrow \mathcal{G}_j]$$

The transitive closure of $\Rightarrow_{\mathcal{F}}$ is $\Rightarrow_{\mathcal{F}}^{+}$.

Definition 2.
A *rooted family* is a pair $[\mathcal{F}, \mathcal{G}_i]$ such that $\mathcal{F}$ is a *hierarchically structured family of plans* and $\mathcal{G}_i$ is a pin graph in $\mathcal{F}$. $\mathcal{G}_i$ is the starting pin graph.

The *language of a rooted family* is defined by:
$L(\mathcal{R}) := \{\mathcal{G} \mid \mathcal{G}_i \Rightarrow_{\mathcal{F}}^{+} \mathcal{G} \quad \wedge \quad C = \emptyset \}$
Remember that the vertices of the created pin graphs are words: The renaming from V to $c.V$ produces longer and longer vertices. So vertices consist of several former compound vertices followed by an elementary vertex. The main point for the planning domain are the last letters. So the same plan can be represented by different graphs, there are infinite many graphs representing a given plan. This leads to a problem, solved by an equivalency concept. ($last(v)$ is simply the last letter of v.)

Definition 3.
The pin graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent iff there is a bijection $r : V_1 \rightarrow V_2$ such that :

- $\forall v \in V_1 : last(v) = last(r(v))$
- $\forall (u, v) \in (V_1 \times V_1) : (u, v) \in E_1 \longleftrightarrow (r(u), r(v)) \in E_2$

We denote it by $\mathcal{G}_1 \cong \mathcal{G}_2$.
Equivalence of pin graphs reflects the intuitive meaning of equality of plans. Equivalence is necessary, but it draws us into a tide of problems. A tasting (You find formal definitions in my paper.) :

- We have to use $\lessapprox$ (equivalent in) instead of $\in$. $\mathcal{G} \lessapprox L$ means L contains a pin graph wich is equivalent to $\mathcal{G}$.
- So we have to define new relations for subset: It is $L_1 \lnapprox L_2$ iff all pin graphs in L_1 are equivalent in L_2 . $\lnapprox$ is understood accordingly.
- Two languages are equivalent ($L_1 \cong L_2$) iff each language is an equivalent subset of the other language.

- $L_1 \setminus\!\!\sim L_2$ means we remove all pin graphs from L_1 wich are equivalent in L_2.

Another problem caused by equivalence is that there are infinite languages wich contain only equivalent copies of a finite number of pin graphs. An infinite language wich can be reduced to a finite language by removing equivalent copies is called weakly infinite.

Definition 4.
A language L is called *strongly infinite* iff $\quad \nexists \hat{L} \ : \ \hat{L}$ is finite $\ \wedge \hat{L} \cong L_1$ $\qquad \square$

Now we are able to define the familie of context-free languages:

Definition 5. $\qquad \mathcal{PG}_{cf} := \{ \ L \mid \exists \mathcal{R} \ : \ L(\mathcal{R}) \cong L \ \}$

The set of all pin graphs over a nonempty finite alphabet Σ is defined as follows:

Definition 6.
$\mathcal{PG}^+(\Sigma) := \{\mathcal{G} \mid \quad \mathcal{G}$ is a pin graph $\quad \wedge \quad C = \emptyset \quad \wedge \quad \forall v \in V : last(v) \in \Sigma\}$ $\quad \square$

3 A Pumping Theorem

Context-free languages allow pumping in the following way:
(You find two stronger versions in my paper.)
c^1 and c^2 are c with upper indices, so $c^2 \neq cc$!
Theorem 1.
$\forall L \in \mathcal{PG}_{cf} : \quad (L$ is strongly infinite$) \longrightarrow$

$$\exists n \in I\!\!N \quad \forall \mathcal{G} \underset{\sim}{\in} L : \quad |\mathcal{G}| \geq n \rightarrow$$

$$\begin{aligned} \exists \mathcal{G}^1, \mathcal{G}^2, \mathcal{G}^3 : \qquad & C^1 = \{c^1\}, |V^1| > 1, \\ & C^2 = \{c^2\}, |V^2| > 1, \\ & C^3 = \emptyset, \\ & \mathcal{G} \cong \mathcal{G}^1[c^1 \hookleftarrow \mathcal{G}^2][c^1c^2 \hookleftarrow \mathcal{G}^3], \\ & \forall j \in I\!\!N : \mathcal{G}^1[c^1 \hookleftarrow \mathcal{G}^2]^{j,c2}[c^1(c^2)^j \hookleftarrow \mathcal{G}^3] \underset{\sim}{\in} L \end{aligned}$$

$\square$

We can not simply pump like $[c^1c^2 \hookleftarrow \mathcal{G}^2][c^1c^2 \hookleftarrow \mathcal{G}^2][c^1c^2 \hookleftarrow \mathcal{G}^2] \ldots$, because the vertex c^1c^2 will be overwritten to $c^1c^2c^2$ and so on...
We have to pump like $[c^1c^2 \hookleftarrow \mathcal{G}^2][c^1c^2c^2 \hookleftarrow \mathcal{G}^2][c^1c^2c^2c^2 \hookleftarrow \mathcal{G}^2]$.

Often it is not necessary to use the last property (the pumping) to proof that a language is not context-free .

4 Other main Results

The language of all pin graphs over an alphabet is not context-free.

Theorem 2.
$\forall\,\Sigma : \qquad \mathcal{PG}^{+}(\Sigma) \notin \mathcal{PG}_{cf}$ □

This is life.
A remarkable fakt is that the pumping theorem can be used to proof theorem 2. You may protest, pumping can never exceed $\mathcal{PG}^{+}$ because it contains each pin graph. The problem is that many pin graphs are not creatable via $\mathcal{G}^1[c^1 \hookleftarrow \mathcal{G}^2][c^1c^2 \hookleftarrow \mathcal{G}^3]$ (c.f. pumping theorem). I leave the proof because the next theorem is stronger.

L_{cd} is the language of all complete, directed graphs over the alphabet $\{a\}$. This language is not context-free. It is even worse: Each context-free language contains only a finite count of pin graphs from L_{cd} or it contains infinite many pin graphs from L_{cd} wich are only equivalent copies of finite many pin graphs from L_{cd}.

Theorem 3.
$\nexists L \in PG_{cf} \;:\quad (L_{cd} \mathrel{\widetilde{\cap}} L)$ is strongly infinite □

This theorem can be proofed using the pumping theorem.
The main lack of the rooted family is the precise rule of drawing new edges in the definition of the substitution.
Context-free languages are closed under union. The idea of the proof is easy: "Merge" the rooted families, create a new single-vertex starting graph from wich you can derive the two "old" starting graphs. This fakt and theorem 2 show that the complement of a context-free language is never context-free. However, there are languages wich are not context-free and their complements are also not context-free. To show it, you may use "the half of" L_{cd} and theorem 3. Context-free languages are not stable under intersection. An easy example is the standard example from the CHOMSKY-hierarchy, $a^n b^n c^n$.

Theorem 4.

- $\forall L_1, L_2 \in \mathcal{PG}_{cf} \;:\qquad (L_1 \cup L_2) \in \mathcal{PG}_{cf}$
- $\forall L \lnsim \mathcal{PG}^{+}(\Sigma) \;:\quad L$ is finite $\rightarrow L \in \mathcal{PG}_{cf}$
- $\forall L \lnsim \mathcal{PG}^{+}(\Sigma) \;:\quad L$ is weakly infinite $\rightarrow L \in \mathcal{PG}_{cf}$

- $\forall L \in \mathcal{PG}_{cf} \ : \quad (\mathcal{PG}^+(\Sigma) \mathbin{\tilde{\setminus}} L) \notin \mathcal{PG}_{cf}$
- $\exists L \subseteq \mathcal{PG}^+(\Sigma) \ : \quad L \notin \mathcal{PG}_{cf} \quad \wedge \quad (\mathcal{PG}^+(\Sigma) \mathbin{\tilde{\setminus}} L) \notin \mathcal{PG}_{cf}$
- $\exists \bar{L}, \hat{L} \in \mathcal{PG}_{cf} \ : \quad (\bar{L} \mathbin{\underset{\sim}{\cup}} \hat{L}) \notin \mathcal{PG}_{cf}$ $\square$

5 Conclusions and next Steps

The main questions are answered. But I am not able to evaluate the suitability of the rooted family for therapy planning. I figured out and proofed several properties wich allow people working in therapy planning a better understanding of the graph grammar concept they created themself. Theorem 3 looks terrible to theorists, may be practists smile about it, because they hate complete directed graphs.
Interesting problems to investigate are the following:

- The concept of the rooted family has been extended such that vertices carry logical constraints wich are inherited in substitutions. The constraints allow a shrinking of the substitution mapping...
- How important is connectity? I believe to know that it is useful to allow disconnected graphs in rooted families, even if you wish to create only connected graphs.
- (How) can we decide membership, finitness...

References

[1] Oksana Arnold and Klaus P. Jantke. Therapy plans as hierarchically structured graphs. WISCON Report 02/94, HTKW Leipzig (FH), Fachbereich IMN, April 1994

[2] Oksana Arnold and Klaus P. Jantke. Therapy Plan Generation as Program Synthesis. in LNAI Vol. 872. pages 40-55, Springer Verlag 1994

[3] John E. Hopcroft and Jeffrey D. Ullmann. Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie. Addison-Wesley, 1979

A Pumping Lemma for Output Languages of Attributed Tree Transducers

Armin Kühnemann and Heiko Vogler

Grundlagen der Programmierung, Institut für Softwaretechnik I,
Fakultät Informatik, Technische Universität Dresden, D–01062 Dresden, Germany,
e–mail: {ak15,vogler} @ inf.tu–dresden.de

Abstract

An attributed tree transducer is a formal model for studying properties of attribute grammars. We introduce a pumping lemma for output languages of noncircular, producing, and visiting attributed tree transducers. The pumping lemma can be applied to gain two results: (1) there is no noncircular, producing, and visiting attributed tree transducer which computes the set of all monadic trees with exponential height as output and (2) there is a hierarchy of noncircular, producing, and visiting attributed tree transducers with respect to their number of attributes.

Keywords: Attribute Grammars, Tree Languages, Pumping Lemma

1 Introduction

First Aho and Ullman have inspected pumping lemmata for output languages of translation schemes in [1], namely for generalized syntax directed translations. Perrault and Ésik have introduced in [8] and [4], respectively, pumping lemmata for (nondeterministic) top–down tree transducers (cf. [9, 10, 2]). Engelfriet, Rozenberg, and Slutzki have presented in [3] a pumping lemma for output languages of deterministic top–down tree–to–string transducers which has a structure that is closely related to the pumping lemma for context–free string languages. The proof of this lemma had a big influence on the development of the pumping lemma for attributed tree transducers, which was introduced in [7].

Attributed tree transducers have been investigated by Fülöp in [5]. They are abstractions of attribute grammars (cf. [6]) in the sense that they take trees over an arbitrary ranked alphabet of input symbols rather than derivation trees as argument, and that the values of the attributes are also trees over a ranked alphabet of output symbols.

Like in attribute grammars, the set of attributes is partitioned into the set of synthesized and inherited attributes which are associated to the input symbols and which compute their values in a bottom–up manner and in a top–down manner, respectively. In contrast to attribute grammars, to every input symbol the whole set of attributes is associated. Roughly speaking, computing the value of a synthesized attribute occurrence of a node x of an input tree, the values of the inherited attribute occurrences of x and of the synthesized attribute occurrences of its sons (if they exist) may be used and, computing the value of an inherited attribute occurrence of x, the values of the inherited attribute occurrences of its father (if it exists) and of the synthesized attribute occurrences of x and of its brothers may be used.

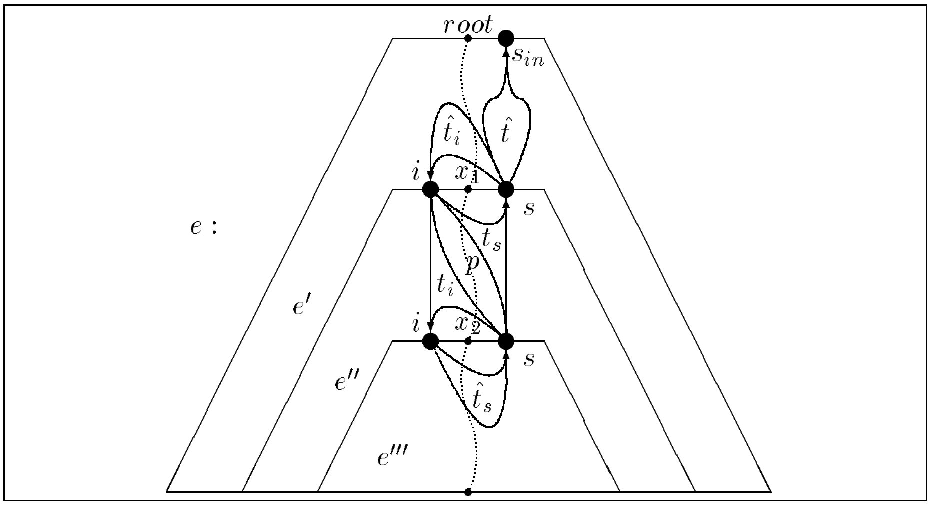
We consider only total deterministic attributed tree transducers: For every node x of an input tree which is labeled by a particular input symbol and for every synthesized attribute s, the computation of the attribute occurrence of s at x is fixed by exactly one rule. Similarly, for every node x which is labeled by a particular input symbol and for every inherited attribute i, the computation of the attribute occurrence of i at the j–th son of x is fixed by exactly one rule.

As in attribute grammars, these dependencies can induce circularities among the attribute occurrences. We restrict the attributed tree transducers to be noncircular and we designate a synthesized attribute s_{in} as initial attribute. Thus we designate an initial attribute occurrence at the root of every input tree of which the value will be the output tree. Then every attributed tree transducer M computes a total function from input trees to output trees. The output language of an attributed tree transducer M is defined as the range of this function.

Additionally, we restrict our pumping lemma to producing and visiting attributed tree transducers. An attributed tree transducer is producing, if every rule application delivers at least one new output symbol. An attributed tree transducer is visiting, if for every input tree and for every node x of it, the value of at least one attribute occurrence of x is needed to compute the value of s_{in} at the root.

The main idea of our pumping lemma is adopted from the pumping lemma for context–free string languages. We consider input trees belonging to a sufficiently large output tree to obtain new pumped output trees: For every producing and visiting attributed tree transducer M, a natural number n_M, called the pumping index of M, can be constructed. If we choose an output tree t from the output language of M which has at least n_M nodes, then every input tree e which can be transformed into t has the following property: e is high enough, such that it has a path p, on which two different nodes x_1 and x_2 can be found, which have the same attribute–set, i.e. the same set of attributes, for which the attribute occurrence of x_1 (and of x_2, respectively) is needed to calculate s_{in} at the root of e. Assuming that x_1 is closer to the root than x_2, we can define the following decomposition of e into input patterns. Roughly speaking,

- e' is the tree e without the subtree which has x_1 as root.
- e'' is the tree which has x_1 as root without the subtree which has x_2 as root.



- e''' is the tree which has x_2 as root.

This decomposition of e induces a decomposition of the output tree t into output patterns. Roughly speaking,

- The pattern $\hat{t}$ corresponds to the normal form of the attribute occurrence s_{in} at the root of e, which is calculated only on the nodes of e'.
- For every synthesized attribute s in the attribute–set of the two nodes x_1 and x_2, the pattern t_s (and $\hat{t}_s$) corresponds to the normal form of the attribute occurrence s at x_1 (and at x_2, respectively), which is calculated only on the nodes of e'' (and e''', respectively).
- For every inherited attribute i in the attribute–set of the two nodes x_2 and x_1, the pattern t_i (and $\hat{t}_i$) corresponds to the normal form of the attribute occurrence i at x_2 (and at x_1, respectively), which is calculated only on the nodes of e'' (and e', respectively).

Since the two nodes x_1 and x_2 are compatible with respect to their attribute–sets, we can construct new input trees by repeating e'' arbitrarily many times. The output patterns t_s and t_i must be used for every repetition of e'' to obtain the corresponding new output trees. Thus the pumping process itself can be described by using only the output patterns.

References

[1] A.V. Aho and J.D. Ullman. Translations on a context free grammar. *Inform. and Control*, 19:439–475, 1971.

[2] J. Engelfriet. Bottom–up and top–down tree transformations — a comparison. *Math. Syst. Theory*, 9:198–231, 1975.

[3] J. Engelfriet, G. Rozenberg, and G. Slutzki. Tree transducers, L systems, and two–way machines. *J. Comput. Syst. Sci.*, 20:150–202, 1980.

[4] Z. Ésik. Decidability results concerning tree transducers. *Acta Cybernetica*, 5:1–20, 1980.

[5] Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.

[6] D.E. Knuth. Semantics of context–free languages. *Math. Syst. Theory*, 2:127–145, 1968.

[7] A. Kühnemann and H. Vogler. A pumping lemma for output languages of attributed tree transducers. *Acta Cybernetica*, 11:261–305, 1994.

[8] C.R. Perrault. Intercalation lemmas for tree transducer languages. *J. Comput. Syst. Sci.*, 13:246–277, 1976.

[9] W.C. Rounds. Mappings and grammars on trees. *Math. Syst. Theory*, 4:257–287, 1970.

[10] J.W. Thatcher. Generalized^2 sequential machine maps. *J. Comput. Syst. Sci.*, 4:339–367, 1970.

On language families of interacting automata

Martin Kutrib

AG Informatik, University Giessen
D-35392 Giessen, Arndtstr. 2, Germany
kutrib@informatik.uni-giessen.de

The language recognition capabilities of some classes of interacting automata are studied. Classical one-dimensional *cellular spaces* and various modifications and generalizations have been investigated for a long time (e.g. [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13]). We restrict our considerations on some of such classes Smith [12] collectively called *polyautomata*, their iterative variants and a real generalization obtained by stack augmentation. Since in the theory of polyautomata there are open problems, as are the question whether linear-time *cellular automata* are more powerful than real-time cellular automata or not, we designate real-time acceptors and acceptors without any time restrictions.

At first we summarize the known relationships between some of the language families. Subsequently the languages definable by the iterative variant of *pushdown cellular automata* are investigated.

Definition.
An *iterative pushdown array* (IPDA) is a system $(S, \Sigma, \Gamma, \sigma_0, \sigma, q_0, g_0, \#)$, where

a) S is the finite, nonempty set of *states*,
b) Σ is the finite, nonempty set of *input symbols*,
c) Γ is the finite, nonempty set of *stack symbols*,
d) $g_0 \in \Gamma$ is the *bottom of stack symbol*,
e) $\# \in \Sigma$ is the *end of input symbol*,
f) $\sigma_0 : S^3 \times \Gamma \times \Sigma \longrightarrow S \times \Gamma^*$ and
$\sigma : S^3 \times \Gamma \longrightarrow S \times \Gamma^*$
are the *local transformations* of the cell at the origin and of all other cells respectively, satisfying
$\forall s_1, s_2, s_3, s_4 \in S, g \in \Gamma, a \in \Sigma : \ \sigma(s_1, s_2, s_3, g) = (s_4, \gamma) \Longrightarrow$
(resp. $\sigma_0(s_1, s_2, s_3, a, g) = (s_4, \gamma) \Longrightarrow$)
$(\gamma \in (\Gamma \setminus \{g_0\})^* \ \wedge \ g \neq g_0) \ \vee \ (\gamma = \gamma' g_0 \ \wedge \ \gamma' \in (\Gamma \setminus \{g_0\})^* \ \wedge \ g = g_0)$
g) $q_0 \in S$ is the *quiescent state*, such that for all $g \in \Gamma \quad \sigma(q_0, q_0, q_0, g) = (q_0, g)$ holds.

The local transition functions induce a length preserving mapping (*global transformation*) $\mathcal{T} : (S \times \Gamma^+)^+ \times \Sigma^\omega \longrightarrow (S \times \Gamma^+)^+ \times \Sigma^\omega$ according to the following.

Let $\bar{\sigma} : S^3 \times \Gamma^+ \longrightarrow S \times \Gamma^+$ be defined as
$\bar{\sigma}(s_1, s_2, s_3, g_m \cdots g_0) :=$

$$\Big(\pi_1(\sigma(s_1, s_2, s_3, g_m)), \pi_2(\sigma(s_1, s_2, s_3, g_m))g_{m-1} \cdots g_0\Big),$$

and $\bar{\sigma}_0 : S^3 \times \Gamma^+ \times \Sigma \longrightarrow S \times \Gamma^+$ be defined as
$\bar{\sigma}_0(s_1, s_2, s_3, g_m \cdots g_0, a) :=$

$$\Big(\pi_1(\sigma_0(s_1, s_2, s_3, g_m, a)), \pi_2(\sigma_0(s_1, s_2, s_3, g_m, a))g_{m-1} \cdots g_0\Big),$$

then
$\forall m \in \mathbb{N}, i \in \{-m, \ldots, -1, 0, 1, \ldots, m\} : \forall s_i \in S, \ \gamma_i \in \Gamma^+, \ a \in \Sigma :$

$$\mathcal{T}((s_{-m}, \gamma_{-m}) \ldots (s_0, \gamma_0) \ldots (s_m, \gamma_m), aa_1a_2 \cdots) :=$$
$$\begin{cases} (\bar{\sigma}(q_0, q_0, s_0, g_0)\bar{\sigma}_0(q_0, s_0, q_0, \gamma_0, a)\bar{\sigma}(s_0, q_0, q_0, g_0), a_1a_2 \cdots) & \text{if } m = 0 \\ (\bar{\sigma}(q_0, q_0, s_{-m}, g_0) \cdots \bar{\sigma}_0(s_{-1}, s_0, s_1, \gamma_0, a) \cdots \bar{\sigma}(s_m, q_0, q_0, g_0), a_1a_2 \cdots) & \\ & \text{if } m > 0 \end{cases}$$

Definition.
Let $M = (S, \Sigma, \Gamma, \sigma_0, \sigma, q_0, g_0, \texttt{\#})$ be an IPDA, $L \subseteq (\Sigma \setminus \{\texttt{\#}\})^+$ a formal language and $t : \mathbb{N} \longrightarrow \mathbb{N}$ a function.
L is *accepted* by M in time t (with respect to F) if and only if $F \subseteq S$ and $L = \{w | \ \pi_1(\pi_0(\mathcal{T}^{t(|w|)}((q_0, g_0), w\texttt{\#}^\omega))) \in F \wedge \forall t' < t(n) : \mathcal{T}^{t'}((q_0, g_0), w\texttt{\#}^\omega))) \notin F\}$ holds.

The families of languages acceptable by IPDAs in time t are denoted by $\mathscr{L}_t$(IPDA). Of special interest are the real-time languages which can be accepted within time $rt(n) := n$.

Trivially, $\mathscr{L}_t(\mathrm{IA}) \subseteq \mathscr{L}_t(\mathrm{IPDA})$ holds true.
We can prove the following results:

Theorem. There exists a context-free language not acceptable by IPDAs in real-time.

Theorem. There exists a context-sensitive not context-free language which belongs to $\mathscr{L}_{rt}$(IPDA).

We conclude that the families $\mathscr{L}_2$ and $\mathscr{L}_{rt}$(IPDA) are incomparable.

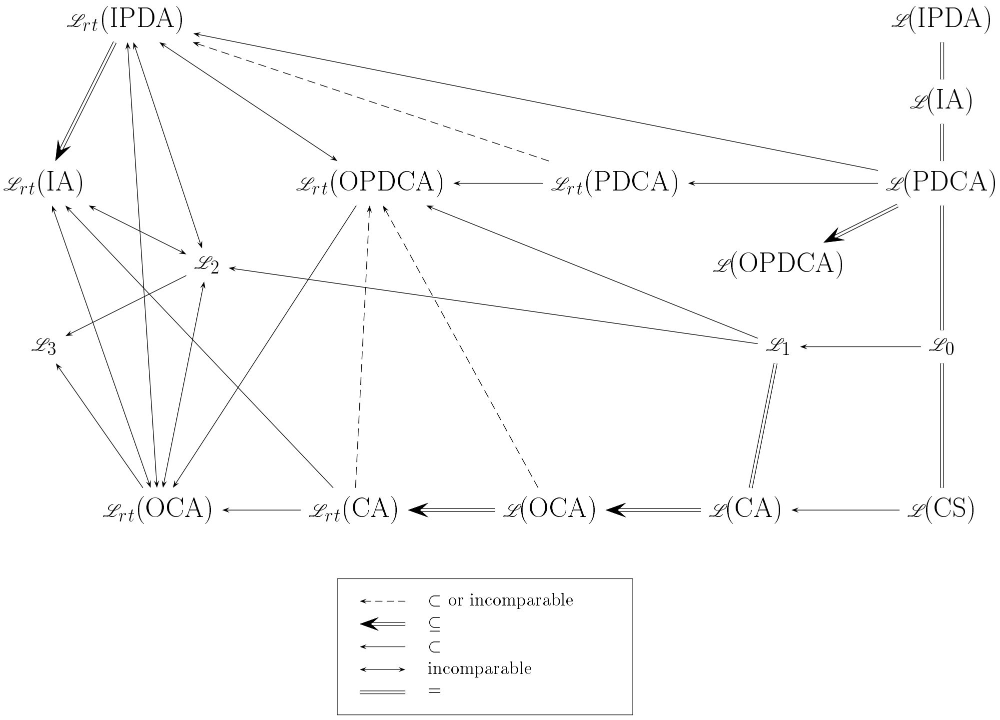
Theorem. $\exists L \notin \mathscr{L}_{rt}(\mathrm{OPDCA}) : \ L \in \mathscr{L}_{rt}(\mathrm{IPDA})$.

Theorem. $\exists L \in \mathscr{L}_{rt}(\mathrm{OPDCA}) : \ L \notin \mathscr{L}_{rt}(\mathrm{IPDA})$.

We conclude that the families $\mathscr{L}_{rt}$(OPDCA) and $\mathscr{L}_{rt}$(IPDA) are incomparable.

Theorem. The time t online TM languages are a subset of $\mathscr{L}_t$(IPDA).

In the following figure some relationships are depicted.



References

[1] Beyer, W. T. *Recognition of topological invariants by iterative arrays*. Technical Report MAC TR-66, Massachusetts Institute of Technology, 1969.

[2] Bucher, W. and Culik II, K. *On real time and linear time cellular automata.* RAIRO Informatique Théorique et Applications. 18 (1984), 307–325.

[3] Choffrut, C. and Culik II, K. *On real-time cellular automata and trellis automata*. Acta Informatica 21 (1984), 393–407.

[4] Cole, S. N. *Real-time computation by n-dimensional iterative arrays of finite-state machines*. IEEE Conference Record of 7^{th} Annual Symposium on Switching and Automata Theory, 1966, pp. 53–77.

[5] Dyer, C. R. *One-way bounded cellular automata*. Information and Control 44 (1980), 261–281.

[6] Kosaraju, S. R. *On some open problems in the theory of cellular automata.* IEEE Transactions on Computers C-23 (1974), 561–565.

[7] Kutrib, M. *Real-time language recognition by pushdown cellular automata.* Proc. Workshop on Parallel Processing, Lessach, 1994, pp. 126–140.

[8] Kutrib, M. and Richstein, J. *Real-time one-way pushdown cellular automata languages*. Bericht 9403, Arbeitsgruppe Informatik, Universität Gießen, Gießen, 1994.

[9] Kutrib, M. and Worsch, Th. *Investigation of different input modes for cellular automata*. Parcella '94, Akademie Verlag, Berlin, 1994, pp. 141–150.

[10] Seidel, S. R. *Language recognition and the synchronization of cellular automata*. Technical Report 79-02, Department of Computer Science, Uni-

versity of Iowa, Iowa City, 1979.

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

[12] Smith III, A. R. *Introduction to and survey of polyautomata theory*. In Lindenmayer, A. and Rozenberg, G. (eds.), *Automata, Languages, Development*. North-Holland, Amsterdam, 1976, pp. 405–422.

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

Automaten mit der Datenstruktur Menge

Klaus-Jörn Lange und Klaus Reinhardt

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

Das Problem *Element Distinctness* besteht darin zu einer gegebenen Menge von Wörten zu entscheiden, ob diese alle paarweise verschieden sind. Während einerseits die Komplexität dieses Problems sehr niedrig ist, scheint andererseits die Beschreibung mit formalsprachlichen Methoden eher aufwendig zu sein.

Zu diesem Zweck betrachten wir Automaten mit dem Speichertyp Menge. Elementare Oprationen sind

> Verlängern eines Abfragewortes auf einem Abfrageband und
> Testen ob ein Abfragewort in der Menge enthalten ist. Wenn nicht, wird das Wort in die Menge aufgenommen.

Zu Beginn einer Berechnung ist die Menge leer. Ausserdem wird nach jeder Abfrage das Abfrageband gelöscht. Es stellt sich heraus, daß das Leerheitsproblem für diesen Automatentyp entscheidbar ist. Weiterhin ist das Wortproblem NP-hart.

Offen sind derzeit noch die Fragen nach der Komplexität des Leerheitsproblems, einer grammatikalischen Charakterisierung und, ob das Wortproblem in NP liegt.

Turing Maschinen mit sublogarithmischen Platzschranken

—

The Complexity World below Logarithmic Space

Maciej Liśkiewicz and **Rüdiger Reischuk**

Med. Universität zu Lübeck*

Rauischholzhausen, September 1995

We investigate space complexity classes defined by Turing machines that use less than logarithmic space. Because of the limited counting ability of such machines most of the standard simulation techniques do not work for sublogarithmic space classes. However, machines with such little space may still be quite powerful. Therefore, it was not obvious how to obtain analogs for inclusion and separation results known for classes above logspace.

We review known facts about the sublogarithmic space world and present several new results, which show that these classes really behave differently. For example, certain closure properties do not hold. The restricted power of these machines makes it possible to prove explicit separations – even for alternating complexity classes – by combinatorial arguments and to obtain a hierarchy of non-relativized complexity classes without any unproven assumption. We will also discuss upward and downward translation issues.

These complexity classes are related to other classes within $\mathcal{P}$, in particular to context-free languages. Finally, we briefly mention extensions to machines that, in addition, can make use of randomness, in other words to interactive proof systems with small space bounds.

*Institut für Theoretische Informatik, Wallstraße 40, 23560 Lübeck
e-mail: liskiewi, reischuk @ informatik.mu-luebeck.de

Bildkompression mit endlichen Automaten

A. Schrader
Universität-Gesamthochschule Siegen
Fachbereich 12 - Elektrotechnik und Informatik
Programmiersprachen
57068 Siegen
e–mail: schrader@informatik.uni-siegen.de

Das 'multimediale Zeitalter' hat zu einer ständig wachsenden Integration von Bilddaten bei der Übertragung und Speicherung von Informationen geführt. Die Begrenzungen der Kapazitäten in Übertragungskanälen und Speichermedien erforderte dabei schon früh die Entwicklung geeigneter Kompressionsalgorithmen.

In den klassischen Verfahren der ersten Generation werden Kenntnisse oder Vermutungen über die statistische Verteilung der Daten zur Konstruktion eines redundanzreduzierenden Codes verwendet. Der zweidimensionale Charakter der Bilddaten bleibt dabei im wesentlichen unberücksichtigt.

In diesem Artikel wird die Codierung von endlichen diskreten Bilddaten mit endlichen Automaten beschrieben. Ausgangspunkt unserer Überlegungen ist eine zweidimensional in einem kartesischen Raster mit äquidistanten Rasterpunkten angeordnete Menge P von $n \times n$ Bildpunkten (Pixel), denen als Intensitätswert ein Element aus der Menge $\{0, 1\}$ zugeordnet wird (Binärbild, oder Bitmap mit 2 Graustufen):

$$P := [0 \ldots n-1] \times [0 \ldots n-1] \rightarrow \{0, 1\}.$$

Die Beschreibung des Bildes P wird mit Hilfe einer endlichen Sprache $L \subset \Sigma^*$ über einem festgelegten Alphabet Σ vorgenommen. Endliche Sprache lassen sich durch reguläre Ausdrücke beschreiben und sind somit Teilmenge der (regulären) Chomsky-Typ3-Sprachklasse, so daß die Beschreibung der Sprache auch durch die Definition eines entsprechenden endlichen Automaten (FA) erfolgen kann.

Zur Erzeugung einer solchen Bildsprache L wird eine bijektive Abbildung

$$\phi := [0 \ldots n-1] \times [0 \ldots n-1] \rightarrow L$$

verwendet, die jedem Pixel des Bildes ein eindeutiges Codewort $\omega \in L$ zuordnet. Eine Möglichkeit besteht in der Verwendung der Quadtree-Zerlegung (Abb. 1), die mit Hilfe des Alphabets $\Sigma = \{0, 1, 2, 3\}$ für jeden Pixel ein Codewort $\omega \in \Sigma^k$ der Länge $|\omega| = k$ erzeugt.

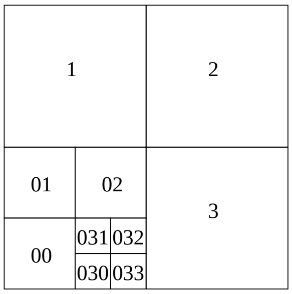


Abbildung 1: Einige Quadtree-Zuordnungen bis $k = 3$

Die Sprache L_P enthält nun alle Codewörter $\omega \in L$, deren Pixel in der Bitmap den Wert 1 besitzen:

$$L_P := \{\omega \in L | P(\phi^{-1}(\omega)) = 1\}.$$

Für diese Sprache L_P wird ein deterministischer endlicher Automat (DFA) $A = (Q, \Sigma, \delta, q_0, F)$ erzeugt, für dessen akzeptierte Sprache gilt:

$$L(A) := \{\omega \in \Sigma^* | \delta^*(q_0, \omega) \in F\} = L_P.$$

Das Bild P läßt sich aus diesem Automaten rekonstruieren, indem der Automat für alle $\omega \in L_P$ in einen akzeptierenden Endzustand übergeht und für alle $\omega \in L \setminus L_P$ in einem nichtakzeptierenden Zustand verbleibt.

Jeder Zustand im Automaten repräsentiert einen bestimmten Subquadranten des Bildes. Die Ähnlichkeit zweier Subquadranten kann auf die Äquivalenz der entsprechenden Zustände übertragen werden und führt so zu einem reduzierten minimalen Automaten mit einer geringen Zustandszahl.

Der Artikel beschreibt zwei heuristische Ansätze zur effektiven Codierung der Zustandsübergangstabelle des Automaten und vergleicht die Ergebnisse mit Werten herkömmlicher Kompressionsverfahren anhand von ausgesuchten Bildbeispielen.

A Modal μ–Calculus for Durational Transition Systems

Helmut Seidl
FB IV – Informatik
Universität Trier
D–54286 Trier
Germany
email: seidl@ti.uni-trier.de

Abstract

Durational transition systems are finite transition systems where every transition is additionally equipped with a duration. We consider the problem of interpreting μ–formulas over durational transition systems. In case the formula contains only operations minimum, maximum, addition, and sequencing, we show that the interpretation ist not only computable but (up to a linear factor) as efficiently computable as the interpretation of ordinary μ–formulas over finite transition systems.

Keywords: μ–Calculus, durational transition systems, model checking.

1 Introduction

The μ–calculus, a propositional modal logic with fixpoint operators, was introduced by Dexter Kozen as a tool for the description and analysis of the behavior of possibly infinite computations [Ko83]. The expressive power of modal μ–calculus is intimately connected to finite–state automata on infinite trees [VW86, EJ91, EJS93, BVW94]. Classically, a μ–formula denotes a predicate on states. Typical properties to be expressed and analyzed are *safety* and *lifeness* assertions. The formula $\nu x.\langle a\rangle x$, for example, denotes the set of all states allowing for an infinite sequence of a–actions. Especially, the modal operator $\langle a\rangle$ (for action a) constructs a property of the actual state from a property of a next state. Thus, it relates presence to a (possibly infinite) future. For detailed explanation of various modal (and temporal) logics consult, e.g., Colin Stirling in [St92].

The problem with which we are concerned here, however, is not only to determine properties of states but also quantitative information. We are interested in questions like: how long does it take a process to reach a state with a certain property. Technically, this means that closed formulas are no longer interpreted as predicates or, equivalently, as mappings $Q \to \{0,1\}$ for some base set Q. Instead, we employ mappings $t : Q \to D$ for some time domain D. In this paper we consider D as a subinterval of

$$-\infty < 0 < 1 < 2 < \ldots < n < \ldots < \infty$$

i.e., the non–negative integers extended by $-\infty$ (to express unaccessibility) and ∞. The "logical" connectives "$\vee$ and "$\wedge$" are now replaced with maximum and minimum, respectively. Additionally, we consider addition "$+$" (appropriately extended) and the sequencing operator ";" which returns the second argument only provided the first one is different from $-\infty$. Otherwise, the result is $-\infty$. While addition is useful for modelling the time behavior of two processes being executed one after the other, the ";" may serve for a primitive form of conditional: the second argument is only counted provided the event corresponding to the first will ever occur.

A corresponding theory of processes with durational actions was developed by Gorrieri, Roccetti and Stancampiano [GRS95]. For the analysis of complexity-like properties of programs, e.g., the number of accesses to some variable, least solutions of (non–hierarchical) systems of equations over non–negative integers (extended by ∞) have been considered in [Se94]. This corresponds to the interpretation of μ–formulas without occurrences of the greatest fixpoint operator.

In this paper we are going to investigate whether or not the interpretation of arbitrary μ–formulas can be effectively computed – even when the chosen time interval is infinite. It turns out that this is in deed the case. Moreover, we find that it is not only computable but, even more, as efficiently (at least up to a linear factor) computable as the interpretation over a two point interval.

2 References

[ACD93] R. Alur, C. Courcoubetis, D.L. Dill: Model Checking in Dense Real Time. Inform. and Comput. 104, 2–34, 1993

[AD94] R. Alur, D.L. Dill: A Theory of Timed Automata. TCS 126, 183–235, 1994

[An92] H.R. Andersen: Model checking and Boolean Graphs. Proc. ESOP'92, LNCS 582, 1–19, 1992

[BVW94] O. Bernholtz, M.V. Vardi, P. Wolper: An Automata–Theoretic Approach to Branching–Time Model Checking (Extended Abstract). Proc. Conf. Computer Aided Verification, LNCS 818, 142–155, 1994

[BCM92] J.R. Burch, E.M. Clarke, K.L. McMillan: Symbolic Model Checking 10^{20} States and Beyond. Inf. and Comput. 98, 142–170, 1992

[EJ91] E.A. Emerson, C.S. Jutla: Tree Automata, Mu–Calculus and Determinacy (Extended Abstract). Proc. 32nd IEEE Symp. on FOCS, 368–377, 1991

[EJS93] E.A. Emerson, C.S. Jutla, A.P. Sistla: On Model–Checking for Fragments of μ–Calculus. Proc. Conf. on Computer Aided Verification, LNCS 697, 385–396, 1993

[GRS95] R. Gorrieri, M. Roccetti, E. Stancampiano: A Theory of Processes with Durational Actions. TCS 140, 73–94, 1995

[HNS94] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine: Symbolic Model Checking for Real–Time Systems. Inform. and Comput. 111, 193–244, 1994

[Ko83] D. Kozen: Results on the Propositional μ–Calculus. TCS 27, 333–354, 1983

[LS94] F. Laroussinie, Ph. Schoebelen: A Hierarchy of Temporal Logics with Past. Proc. 11th STACS, LNCS 775, 47–58, 1994; long version submitted to TCS

[LPS95] F. Laroussinie, S. Pinchinat, Ph. Schoebelen: Translations between Modal Logics of Reactice Systems. Special Issue on AMAST'93, TCS 140, 53–71, 1995

[LBC94] D.E. Long, A. Browne, E.M. Clarke, S. Jha, W.R. Marrero: An Improved Algorithm for the Evaluation of Fixpoint Expressions. Proc. Conf. on Computer Aided Verification, LNCS 818, 338–350, 1994

[Se94] H. Seidl: Least Solution of Equations over $\mathcal{N}$. Proc. ICALP'94, LNCS 820, 400–411, 1994

[St92] C. Stirling: Modal and Temporal Logics. In: S. Abramsky, D. Gabbay, and T. Maibaum (eds.): Handbook of Logic in Computer Science, 477–563, 1992

[Va88] M. Vardi: A Temporal Fixpoint Calculus. Proc. 15th POPL, 250–259, 1988

[VW86] M.V. Vardi, P. Wolper: Automata–Theoretic Techniques for Modal Logics of Programs. JCSS 32, 183–221, 1986

Uniformly optimal prediction of infinite words

Ludwig Staiger
Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
Weinbergweg 17
D-06120 Halle (Saale)
Germany
email: staiger@informatik.uni-halle.de

Abstract

The present paper links the concepts of Kolmogorov complexity (in Complexity theory) and Hausdorff dimension (in Fractal geometry) for a class of recursive (computable) ω-languages.

It is shown that the complexity of an infinite string contained in a Σ_2-definable set of strings is upper bounded by the Hausdorff dimension of this set and that this upper bound is tight. Moreover, we show that there are computable gambling strategies guaranteeing a uniform prediction quality arbitrarily close to the optimal one estimated by Hausdorff dimension and Kolmogorov complexity provided the gambler's adversary plays according to a sequence chosen from a Σ_2-definable set of strings.

We provide also examples which give evidence that our results do not extend further in the Arithmetical hierarchy.

Teilnehmerverzeichnis

- **Oliver Boldt**
  Fakultät für Informatik
  Otto-von-Guericke-Universität Magdeburg
  Postfach 41 20
  39016 Magdeburg

  Tel.: 0391/671-2851
  Fax:
  Email: boldt@fma2.math.uni-magdeburg.de
  WWW:

- **Henning Bordihn**
  Fakultät für Informatik
  Otto-von-Guericke-Universität Magdeburg
  Postfach 41 20
  39016 Magdeburg

  Tel.: 0391/671-2851
  Fax: 0391/67-12810
  Email: bordihn@cs.uni-magdeburg.de
  WWW: http://irb.cs.uni-magdeburg.de/theo/bordihn.html

- **Thomas Buchholz**
  AG Informatik
  Universität Gießen
  Arndtstr. 2
  35392 Gießen

  Tel.: 06003/1289
  Fax: 0641/702-2547
  Email: buchholz@informatik.uni-giessen.de
  WWW: http://www.informatik.uni-giessen.de/mitglieder/buchholz/

- **Gerhard Buntrock**
  Lehrstuhl für Theoretische Informatik
  Institut für Informatik
  Universität Würzburg
  Am Exerzierplatz 3
  97072 Würzburg

  Tel.: 0931/79621.12
  Fax: 0931/79621.20
  Email: bunt@informatik.uni-wuerzburg.de
  WWW: http://haegar.informatik.uni-wuerzburg.de/person/mitarbeiter/bunt

- **David Damanik**
  FB Mathematatik, AG 8.1
  Johann Wolfgang Goethe Universität
  Robert-Mayer-Str. 10
  60054 Frankfurt/Main

  Tel.: 069/79823752
  Fax:
  Email: damanik@math.uni-frankfurt.de
  WWW:

- **Kai Differt**
  Marienstr. 124
  63069 Offenbach

  Tel.: 069/835438
  Fax:
  Email: differt@psc.informatik.uni-frankfurt.de
  WWW:

- **Manfred Droste**
  Institut für Algebra
  TU Dresden
  01062 Dresden

  Tel.: 0351/463-3908
  Fax: 0351/463-4235
  Email: droste@math.tu-dresden.de
  WWW:

- **Henning Fernau**
  Universtität Tübingen
  Wilhelm-Schickard-Institut für Informatik
  Sand 13
  72076 Tübingen

  Tel.: 07071/29-7569
  Fax: 07071/68142
  Email: fernau@informatik.uni-tuebingen.de
  WWW: http://wsiserv.informatik.uni-tuebingen.de/~holzer/mitarbeiter/fernau.html

- **Rudolf Freund**
  Technische Universität Wien
  Resselg. 3
  A-1040 Wien

  Tel.: 00431588014084
  Fax: 004315041589
  Email: freund@csdec1.tuwien.ac.at
  WWW: http://logic.tuwien.ac.at/staff/rudi/home.html

- **H. Peter Gumm**
  Universität Marburg
  Fachbereich Mathematik
  Hans-Meerwein Str.
  35032 Marburg

  Tel.: 06421/285447
  Fax: 06421/285419
  Email: gumm@mathematik.uni-marburg.de
  WWW: http://www.mathematik.uni-marburg.de/~gumm

- **Norbert Gutleben**
  Theoretische Informatik
  BTU Cottbus
  Postfach 101 344
  03013 Cottbus

  Tel.: 0355/692187
  Fax: 0355/692236
  Email: gutleben@informatik.tu-cottbus.de
  WWW:

- **Markus Holzer**
  Universtität Tübingen
  Wilhelm-Schickard-Institut für Informatik
  Sand 13
  72076 Tübingen

  Tel.: 07071/29 7568
  Fax: 07071/ 68142
  Email: holzer@informatik.uni-tuebingen.de
  WWW: http://wsiserv.informatik.uni-tuebingen.de/~holzer/mitarbeiter/holzer.html

- **Martin Hühne**
  Lehrstuhl II
  Fachbereich Informatik
  Universität Dortmund
  44221 Dortmund

  Tel.: 0231/755-4808
  Fax: 0231/755-2047
  Email: huehne@ls2.informatik.uni-dortmund.de
  WWW: http://ls2-www.informatik.uni-dortmund.de/~huehne

- **Klaus P. Jantke**
  Hochschule für Technik, Wirtschaft
  und Kultur Leipzig
  Fachbereich IMN
  Postfach 30066
  04251 Leipzig

  Tel.: 0341/350-6426
  Fax: 0341/350-6426
  Email: jantke@informatik.th-leipzig.de
  WWW: http://www.informatik.th-leipzig.de

- **Matthias Jantzen**
  FB-Informatik
  Universität Hamburg
  Vogt-Kölln-Str. 30
  22527 Hamburg

  Tel.: 040/54715-409
  Fax: 040/54715-246
  Email: jantzen@informatik.uni-hamburg.de
  WWW:

- **Daniel Kirsten**
  Hochschule für Technik, Wirtschaft
  und Kultur Leipzig
  Fachbereich IMN
  Postfach 30066
  04251 Leipzig

  Tel.:
  Fax:
  Email: kirsten@imn.th-leipzig.de
  WWW:

- **Henner Kröger**
  AG Informatik
  Universität Gießen
  Arndtstr. 2
  35392 Gießen

  Tel.: 0641/702-2547
  Fax: 0641/702-2547
  Email: kroeger@informatik.uni-giessen.de
  WWW: http://www.informatik.uni-giessen.de/mitglieder/kroeger.html

- **Armin Kühnemann**
  Institut für Softwaretechnik
  Fakultät Informatik
  TU Dresden
  01062 Dresden

  Tel.: 0351/4575374
  Fax: 0351/4575504
  Email: kuehne@orchid.inf.tu-dresden.de
  WWW:

- **Martin Kutrib**
  AG Informatik
  Universität Gießen
  Arndtstr. 2
  35392 Gießen

  Tel.: 0641/702-2540
  Fax: 0641/702-2547
  Email: kutrib@informatik.uni-giessen.de
  WWW:

- **Klaus-Jörn Lange**
  Universtität Tübingen
  Wilhelm-Schickard-Institut für Informatik
  Sand 13
  72076 Tübingen

  Tel.: 07071/29-7567
  Fax: 07071/68142
  Email: lange@informatik.uni-tuebingen.de
  WWW: http://wsiserv.informatik.uni-tuebingen.de/~lange

- **Friedrich Otto**
  Fachbereich Mathematik/Informatik
  Universität-GH Kassel
  34109 Kassel

  Tel.: 0561/804-4573
  Fax: 0561/804-4008
  Email: otto@theory.informatik.uni-kassel.de
  WWW:

- **Bernd Reichel**
  Fakultät für Informatik
  Otto-von-Guericke-Universität Magdeburg
  Postfach 41 20
  39016 Magdeburg

  Tel.: 0391/67-12851
  Fax: 0391/67-12810
  Email: reichel@cs.uni-magdeburg.de
  WWW: http://irb.cs.uni-magdeburg.de/theo/reichel.html

- **Klaus Reinhardt**
  Universtität Tübingen
  Wilhelm-Schickard-Institut für Informatik
  Sand 13
  72076 Tübingen

  Tel.: 07071/29-7569
  Fax: 07071/68142
  Email: reinhard@informatik.uni-tuebingen.de
  WWW: http://wsiserv.informatik.uni-tuebingen.de/~holzer/mitarbeiter/reinhardt.html

- **Rüdiger Reischuk**
  Institut für Theoretische Informatik
  Universität Lübeck
  Wallstraße 40
  23560 Lübeck

  Tel.: 0451/7030-416
  Fax: 0451/7030-438
  Email: reischuk@informatik.mu-luebeck.de
  WWW: http://www.informatik.mu-luebeck.de

- **Jörg Richstein**
  AG Informatik
  Universität Gießen
  Arndtstr. 2
  35392 Gießen

  Tel.: 0641/702-2540
  Fax: 0641/702-2547
  Email: richstein@informatik.uni-giessen.de
  WWW:

- **Arnd Rußmann**
  FB 20 (Informatik)
  Johann Wolfgang Goethe Universität
  Postfach 11 19 32
  60054 Frankfurt/Main

  Tel.: 069/798-28419
  Fax:
  Email: russmann@psc.informatik.uni-frankfurt.de
  WWW: http://jaguar.psc.informatik.uni-frankfurt.de:8000/~russmann

- **Andreas Schrader**
  Universität-GH Siegen
  FB 12 - Elektrotechnik und Informatik
  Fachgruppe Programmiersprachen
  Hölderlinstr. 3
  57068 Siegen

  Tel.: 0271/7402314
  Fax: 0271/7402532
  Email: schrader@informatik.uni-siegen.de
  WWW: http://www.informatik.uni-siegen.de/~schrader/index.html

- **Helmut Seidl**
  FB IV - Abteilung Informatik
  Universität Trier
  54286 Trier

  Tel.: 0651/201-2835
  Fax: 0651/201-3822
  Email: seidl@ti.uni-trier.de
  WWW: http://www.informatik.uni-trier.de/~seidl/

- **Ludwig Staiger**
  Martin-Luther-Universität Halle-Wittenberg
  Institut für Informatik
  Weinbergweg 17
  06120 Halle (Saale)

  Tel.: 0345/5524714
  Fax: 0345/5527009
  Email: staiger@informatik.uni-halle.de
  WWW: http://www.informatik.uni-halle.de/~staiger/

- **Ralf Stiebe**
  Fakultät für Informatik
  Otto-von-Guericke-Universität Magdeburg
  Postfach 41 20
  39016 Magdeburg

  Tel.: 0391/671-2851
  Fax: 0391/671-2810
  Email: stiebe@cs.uni-magdeburg.de
  WWW: http://irb.cs.uni-magdeburg.de/theo/stiebe.html

- **Wolfgang Thomas**
  Institut für Informatik
  und Praktische Mathematik
  Universität Kiel
  Olshausenstr. 40
  24098 Kiel

  Tel.: 0431/880-4480
  Fax: 0431/880-4054
  Email: wt@jupiter.informatik.uni-kiel.d400.de
  WWW: http://www.informatik.uni-kiel.de/~wt/

- **Klaus Wich**
  Spessartstr. 3
  63477 Maintal-Bischhofsheim

  Tel.: 06109/68608
  Fax:
  Email: wich@informatik.uni-frankfurt.de
  WWW:

- **Renate Winter**
  Martin-Luther-Universität Halle
  Institut für Informatik
  Weinbergweg 17
  06120 Halle/Saale

  Tel.: 0345/5524 719
  Fax:
  Email: winter@informatik.uni-halle.de
  WWW: http://www.informatik.uni-halle.de/~winter/

- **Thomas Worsch**
  Lehrstuhl Informatik für Ingenieure
  und Naturwissenschaftler
  Universität Karlsruhe
  Am Fasanengarten 5
  76128 Karlsruhe

  Tel.: 0721/608-4311
  Fax: 0721/698675
  Email: worsch@ira.uka.de
  WWW: http://www.ira.uka.de/I3V_HTML/MITARBEITER/75