GI-Fachgruppe 0.1.5

Automaten und Formale Sprachen

7. Theorietag

mit Workshop

Perspektiven der Automaten und Formalen Sprachen

Barnstorf, 29.9.97 bis 1.10.97

Vortragszusammenfassungen

zusammengestellt von F. Drewes

Vorträge

Workshop

Theorietag

Monadic second order logic on strings and trees

Joost Engelfriet
Department of Computer Science
Leiden University
P.O. Box 9512
NL-2300 RA Leiden
The Netherlands
engelfri@wi.leidenuniv.nl

A closed formula of monadic second order logic (MSO) can be used to define a graph language (i.e., a set of graphs): all graphs that satisfy the formula. If the graphs are strings (trees), it is well known that the MSO definable languages are exactly the regular languages (the regular tree languages, respectively).

Graph transductions, i.e., translations from one graph into another, can be defined by MSO formulas in the following way. If g is a source graph, then the nodes of the target graph h are defined by means of MSO formulas with one free variable to be interpreted in g. The edges of h are defined by MSO formulas with two free variables, defining binary relations on the nodes of g (and then restricting them to the nodes of h).

If the source graphs are trees, the transduction can be viewed as a context-free graph grammar generating its range (analogous to the fact that a context-free string grammar generates the set of yields of its derivation trees). If the source and target graphs are trees, the transductions can be computed by macro tree transducers which are a formal model for denotational semantics closely related to attribute grammars. If the source and target graphs are strings, the transductions can be computed by 2-way deterministic finite-state transducers.

Über den modularen hierarchischen Entwurf reaktiver Systeme

Ursula Goltz
Universität Hildesheim
Marienburger Platz 22
D-31141 Hildesheim
goltz@informatik.uni-hildesheim.de

Der Begriff "reaktive Systeme" wurde geprägt, um Programmsysteme zu beschreiben, deren Verhalten während des im allgemeinen zeitlich unbeschränkten Ablaufs im Zusammenhang mit ihrer Umgebung gesehen werden muß. Typisch sind etwa Anwendungen im Bereich der Fertigungssteürung oder der Kommunikationstechnologie. Die groe Komplexität solcher Systeme bei hohen Anforderungen an ihre Zuverlässigkeit erfordern neue Entwurfsmethoden, um die Korrektheit in hohem Maße zu gewährleisten.

Als Bestandteile einer solchen Entwurfsmethodik werden logische Spezikikationsmethoden zur Beschreibung der gewünschten Systemeigenschaften, sowie Sprachen zur modularen hierarchischen Beschreibung des Systementwurfs bis hin zur implementierungsnahen Ebene vorgeschlagen. Eine wichtige Rolle spielt die Wahl eines geeigneten Systemmodells, das unter anderem auch erlauben soll, beide Anteile zu einem Entwurfskalkül zu verbinden.

Das Ziel bei der hierarchischen Modellierung von Systemen ist, die Komplexität von Systemen durch Abstraktionsebenen zu strukturieren. Der Nachweis von Eigenschaften kann dann jeweils auf dem höchstmöglichen Abstraktionsniveau geführt werden. Die Schwiergkeit ist die Wahl eines hinreichend strengen Verfeinerungskonzepts. Erste Ansätze eines Verfeinerungskonzeptes in Prozeßalgebren haben die gewünschten Eigenschaften, erweisen sich jedoch in Fallstudien als zu strikt. Als Verallgemeinerung werden "parametrisierte" Verfeinerungen vorgeschlagen, wo die zu vererbenden Beziehungen zwischen Systemteilen von Benutzer spezifiziert werden. Erste Untersuchungen zeigen, daß ein ähnlicher Ansatz in der logischen Spezifikation von Systemeigenschaften möglich und mit der hierarchischen Modellierung verträglich ist.

Aspekte automatischer Folgen

F. v. Haeseler G. Skordev
CeVis Universität Bremen
Universitätsallee 29
28359 Bremen

Es sei A eine endliche Menge, $\underline{x} : \mathbf{N} \to A$ sei eine Folge mit Werten in A. Ist k eine natürliche Zahl, $k \geq 2$, so heißt die Folge $\underline{x}$ k-automatisch, wenn sie einer der folgenden äquivalenten Bedingungen genügt.

- Die Folge wird durch einen endlich k-Automaten generiert.
- Die Folge ist Bild eines Fixpunktes einer k-Ersetzung.
- Der k-Kern der Folge ist endlich.
- Die Folge kann als Lösung einer Mahlergleichung verstanden werden.

In dem Vortrag werden die obigen Konzepte erläutert. Zusätzlich führen wir den sogenannten Kerngraphen ein werden einige Anwendungen diskutieren, diese sind

- Periodische und schließlich periodische Folgen.
- Existenz verschiedener Folgen mit gleichen Kerngraphen.
- Geometrische Objekte, die durch den Kerngraph bestimmt sind. Die Hausdorff Dimension dieses Objekts und ihre Beziehung zu der Folge, die den Kerngraph bestimmt.

Abschließend werden wir noch Verallgemeinerungen für Doppelfolgen betrachten.

Algorithmische Probleme für baumartige Strukturen: Automatisch/automatentheoretisch vom Problem zur Lösung

Detlef Seese
AIFB, Universitt Karlsruhe
D-76128 Karlsruhe
seese@aifb.uni-karlsruhe.de

Der Vortrag gibt einen Überblick über algorithmische Probleme für Graphen mit baumartiger Struktur (universell beschränkter Baumweite) und zeigt die Einsetzbarkeit von Baumautomaten für deren Lösung.

Theorie der Tree Transducer

Heiko Vogler
Technische Universität Dresden, Fakultät Informatik
Institut für Softwaretechnik I, Grundlagen der Programmierung
http://orchid.inf.tu-dresden.de/gdp/
vogler@inf.tu-dresden.de

Tree Transducers - also: Baumübersetzer - sind spezielle Termersetzungssysteme, mit deren Hilfe syntaxgesteuerte Semantik spezifiziert werden kann [Eng81]. Das Konzept der syntaxgesteuerten Semantik basiert auf der Idee, daß, für ein Wort w einer kontextfreien Sprache, die Semantik eines Teilworts v von w durch die Semantik der syntaktischen Teilstrukturen von v und den Kontext von v definiert wird.

Ein tree transducer M spezifiziert eine Baum-zu-Baum Übersetzung τ_M; das Argument t von τ_M repräsentiert den Ableitungsbaum des Worts w (z.B. w ist ein Programm einer Programmiersprache). Vermöge der üblichen Termersetzungssemantik berechnet M einen Term $\tau_M(t)$ über einer Signatur (oder: Rangalphabet) Δ. Die Signatur bestimmt den initialen Homomorphismus h der initialen Δ-Algebra auf den vorliegenden semantischen Bereich. Durch Anwendung von h auf $\tau_M(t)$ (d.h. Ausrechnen der semantischen Operationen) wird schließlich die Semantik von w (d.h. $h(\tau_M(t))$) berechnet. Im Rahmen des Vortrags werde ich anhand eines Beispiels die Beziehung zwischen syntaxgesteuerter Semantik und tree transducer erläutern.

Inzwischen gibt es eine Fülle von Klassen von tree transducers, die durch die Notwendigkeit der Erweiterung oder Einschränkung des ursprünglichen Konzepts motiviert sind. In der Theorie der tree transducer werden deren Ausdrucksstärke, (De-)Komponierbarkeit, Kompositionsmonoide, Charakterisierbarkeit durch iterative Maschinenmodelle, Transformierbarkeit in effizientere Auswertungsalgorithmen und formalsprachlichen Eigenschaften untersucht (vgl. auch den Übersichtsartikel [GS97]).

In diesem Vortrag möchte ich zwei Charakterisierungsresultate der Klasse MAC vorstellen – MAC ist die Klasse der von macro tree transducers berechneten Baum-zu-Baum Übersetzungen. Die Charaktersierung erfolgt durch sogenannte *top-down tree-to-graph transducer* [EV94] und *bottom-up tree-to-graph transducer* [EV96].

Literatur

[Eng81] J. Engelfriet. Tree transducers and syntax-directed semantics. Technical Report Memorandum 363, Technische Hogeschool Twente, 1981. also in: Proceedings of 7th Colloquium on Trees in Algebra and Programming (CAAP 1992), Lille, France, March 4-6, 1992.

[EV94] J. Engelfriet and H. Vogler. The translation power of top-down tree-to-graph transducers. *Journal of Computer and System Sciences*, 49:258–305, 1994.

[EV96] J. Engelfriet and H. Vogler. The equivalence of bottom-up and top-down tree-to-graph transducers. Technical report, Dresden University of Technology, 1996. TUD/FI/96-17.

[GS97] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol.3*, pages 1–68. Springer-Verlag, 1997.

Ergebnisse zu Komplexitätsklassen und Bemerkungen zur Zeitreduktion in zellularen Räumen

Thomas Buchholz Martin Kutrib
Institut für Informatik
Universität Gießen
Arndtstr. 2, 35392 Gießen
{buchholz, kutrib}@informatik.uni-giessen.de

Zellulare Räume sind seit ihrer Einführung durch J. von Neumann (1966) [8] hinsichtlich verschiedener Aspekte untersucht worden. Standen in den frühen Jahren strukturelle Fragen (z.B. Standardisierungen, Fähigkeit zur Selbstreproduktion) im Vordergrund, so konzentrierte sich das Interesse in der Folgezeit mehr und mehr auf Probleme im Zusammenhang mit Berechnungen (z.B. Sprachverarbeitung, Mustertransformation, Simulation, Modellierung).

Wir betrachten neben den klassischen *zellularen Räumen* (CS) deren *iterative* Varianten (IA) [2] sowie die linear raumbeschränkten *zellularen Automaten* (CA) und eine Verallgemeinerung, bei der jede Zelle mit zusätzlichem Kellerspeicher versehen ist (PDCA) [6]. Es werden zwei naheliegende Verbindungsstrukturen unterschieden. Im eindimensionalen Euklidischen Raum ist zum einen jede Zelle mit ihren beiden unmittelbaren Nachbarn verbunden, woraus sich ein bidirektionaler Informationsfluß ergibt, zum anderen ist jede Zelle lediglich mit ihrem unmittelbaren rechten Nachbarn verbunden, woraus sich ein unidirektionaler Informationsfluß von rechts nach links ergibt (Abkürzungen OCA, OPDCA) [3, 7].

Ein erstaunlich hartnäckiges Problem ist die Existenz einer Sprache, deren Erkennung in bidirektionalen zellularen Automaten mehr als Realzeit benötigt (es ist z.B. nicht bekannt, ob ein CA in Exponentialzeit mehr zu leisten vermag als in Realzeit).
Ein weiteres offenes Problem ist die Frage, ob bidirektionale zellulare Automaten ohne Zeitbeschränkung mehr zu leisten vermögen, als unidirektionale [4].

Wir werden dieses Problem für PDCAs positiv beantworten.

Weitere Ergebnisse bzgl. Komplexitätsklassen betreffen den Bereich zwischen Real- und Linearzeit in OCAs. Dank bekannter Beschleunigungssätze [5] läßt sich jede Linearzeit-Sprache bereits in $(1+\varepsilon)$ mal Realzeit erkennen (für ein rationales $\varepsilon > 0$). Üblicherweise gibt man sich mit dieser „fast Realzeit“ zufrieden.

Es wird sich aber herausstellen, daß Sprachen in Komplexitätsklassen unterhalb von Linearzeit existieren, die nicht in Realzeit erkannt werden können. Im unären Fall läßt sich sogar eine Lücke zwischen Realzeit und Realzeit + Logarithmus nachweisen.

Anhand dieser Problematik läßt sich verdeutlichen, daß die üblicherweise verwendete Definition von Zeit-Komplexitätsklassen nur dann das Gewünschte leistet, wenn die Komplexitätsfunktion selbst im zellularen Automaten zeitberechenbar ist [1].

In einem zweiten Teil wird die Möglichkeit, gleichzeitig die Nachbarschaft und die Zeit in einem zellularen Raum zu reduzieren [9], neu beleuchtet. Die hierfür notwendige Abbildung der Anfangskonfiguration des simulierten Raumes in den simulierenden ist eng verknüpft mit dem zugrundeliegenden Simulationsbegriff.

Anhand verschiedener Simulationsbegriffe kann gezeigt werden, daß die bekannten Ergebnisse zur Zeitreduktion in ihrer Allgemeinheit eher fragwürdig sind. Entsprechende Aussagen mit leicht modifizierten Voraussetzungen können aber bewiesen werden.

Literatur

[1] Buchholz, Th. and Kutrib, M. *On time computability of functions in one-way cellular automata*. Erscheint in Acta Informatica (1998).

[2] Cole, S. N. *Real-time computation by n-dimensional iterative arrays of finite-state machines*. IEEE Transactions on Computers C-18 (1969), 349–365.

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

[4] Ibarra, O. H. and Jiang, T. *On some open problems concerning the complexity of cellular arrays*. Results and Trends in Theoretical Computer Science, LNCS 812, 1994, pp. 183–196.

[5] Ibarra, O. H. and Palis, M. A. *Some results concerning linear iterative (systolic) arrays*. Journal of Parallel and Distributed Computing 2 (1985), 182–218.

[6] Kutrib, M. *Pushdown cellular automata*. Erscheint in Theoretical Computer Science 203 (1997).

[7] Kutrib, M. and Richstein, J. *Real-time one-way pushdown cellular automata languages*. Developments in Language Theory II. At the Crossroads of Mathematics, Computer Science and Biology, 1996, pp. 420–429.

[8] von Neumann, J. *Theory of Self-Reproducing Automata*. edited and completed by Arthur W. Burks. University of Illinois Press, 1966.

[9] Smith III, A. R. *Cellular automata complexity trade-offs*. Information and Control 18 (1971), 466–482.

Linksableitungen bei programmierten Grammatiken

Jürgen Dassow
Otto-von-Guericke-Universität Magdeburg
Postfach 41 20
D-39016 Magdeburg

Henning Fernau
WSI, Universität Tübingen
Sand 13
D-72076 Tübingen

Gheorghe Păun
Institute of Mathematics of the Romanian Academy
PO Box 1 – 764
70700 Bucureşti

Frank Stephan
Universität Heidelberg
Im Neuenheimer Feld 294
D-69120 Heidelberg

Programmierte Grammatiken wurden bereits 1969 von Rosenkrantz eingeführt. Dennoch sind zahlreiche Fragen, insbesondere bei Linksableitungen, in den vergangenen Jahren unbeantwortet geblieben. In diesem Vortrag werden Lösungen zu den meisten der in der Monographie von Dassow und Păun [1] aufgelisteten Probleme nebst anderen neuen Ergebnissen zu Linksableitungen angegeben.

Beispielsweise enthalten programmierte kontextfreie Sprachen ohne Vorkommenstest nicht-rekursive Sprachen, aber nicht die kontextsensitive Sprache $\{ a^{2^n} \mid n \geq 0 \}$ gemäß dem bekannten Ergebnis von Hauschildt und Jantzen [3]. Ferner lassen sich alle aufzählbaren Sprachen durch programmierte Grammatiken unter Links-3-Ableitung erzeugen, obschon es keinen Algorithmus zur Überführung einer Turingmaschine in eine solche Grammatik gibt, siehe [2].

Literatur

[1] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory.* Berlin: Springer, 1989.

[2] H. Fernau. On unconditional transfer. In W. Penczek and A. Szałas, editors, *MFCS'96*, volume 1113 of *LNCS*, pages 348–359, 1996.

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

Bilderzeugung mit Hilfe von Baumgeneratoren

Frank Drewes
Fachbereich 3 – Mathematik/Informatik
Universität Bremen
Postfach 33 04 40
D–28334 Bremen
drewes@informatik.uni-bremen.de

Baumgeneratoren sind Regelsysteme, die Baumsprachen (also Mengen von Bäumen) erzeugen. Ein Baum in diesem Sinne ist ein Term über einer gegebenen Signatur. Typische Beispiele für Baumgeneratoren sind reguläre Baumgrammatiken und „top-down tree transducer". (Bei letzteren wird der Bildbereich der berechneten Transformation als die generierte Baumsprache angesehen.)

Ein Baumgenerator g, der eine Baumsprache $L(G)$ erzeugt, kann als bildgenerierendes System verstanden werden, indem die in den erzeugten Bäumen vorkommenden Symbole als Operationen auf Bildern interpretiert werden. Dann nämlich beschreibt jeder Term t ein Bild $val(t)$, d.h. g liefert die Bildsprache $\mathcal{L}(g) = \{val(t) \mid t \in L(g)\}$.[1]

Jede Wahl einer Klasse von Baumgeneratoren und einer Menge von Operationen definiert also einen bestimmten Typ bilderzeugender Systeme. Es ist nicht schwer zu sehen, bei welcher Wahl sich Äquivalentes zu den klassischen Methoden wie z.B. Kettenkode-Bildgrammatiken, L-Systemen mit Schildkrötengeometrie, iterativen Funktionensystemen und Collagen-Grammatiken ergibt. Die bekannten Methoden der Bilderzeugung in diesen begrifflichen Rahmen zu übersetzen, erscheint aus folgenden Gründen als interessant:

1. Durch den einheitlichen Rahmen ergibt sich automatisch eine Klassifizierung der verschiedenen Typen bilderzeugender Systeme, die deren Beziehungen verdeutlicht. Die beiden Grundkomponenten – Baumgeneratoren und Operationen auf Bildern – können als die syntaktische bzw. semantische Ebene der Erzeugung einer Bildsprache verstanden werden. Bezüglich der vier oben genannten Systeme ergibt dies zwei sich orthogonal zueinander verhaltende Unterteilungen. Auf der semantischen Ebene erweisen sich einerseits Kettenkode und Schildkrötengeometrie sowie andererseits iterative Funktionensysteme und Collagen-Grammmatiken als verwandt. Die syntaktische Einordnung dagegen offenbart eher Ähnlichkeiten zwischen Kettenkode- und Collagen-Grammatiken auf der einen Seite sowie zwischen L-Systemen und iterativen Funktionensystemen auf der anderen.

[1] Selbstverständlich funktioniert dies nicht nur für Bilder, sondern ebenso für jeden andern Datenbereich. Der klassische Fall ist der der Wörter, aber auch für die Untersuchung verschiedener Klassen kontextfreier Graphsprachen wurde von dieser Möglichkeit bereits Gebrauch gemacht (siehe [Eng94]).

2. Vergleiche, Verallgemeinerungen und Kombinationen verschiedener Typen bildgenerierender Systeme werden einfacher.

3. Die reichhaltige Theorie der Baumgrammatiken und „tree transducer“ läßt sich verwenden, um zu neuen Ergebnissen zu gelangen und Beweise zu führen. Außerdem besteht die Möglichkeit, Sätze und Beweise bis zu einem gewissen Grad unabhängig vom betrachteten Typ bilderzeugender Systeme zu formulieren (siehe z.B. [Dre96], wo sowohl IFS als auch Collagen-Grammatiken behandelt wurden).

Literatur

[Dre96] Frank Drewes. Language theoretic and algorithmic properties of d-dimensional collages and patterns in a grid. *Journal of Computer and System Sciences*, 53:33–60, 1996.

[Eng94] Joost Engelfriet. Graph grammars and tree transducers. In S. Tison, editor, *Proc. CAAP 94*, volume 787 of *Lecture Notes in Computer Science*, pages 15–37. Springer, 1994.

Interprozedurale Analyse mithilfe von Kellerautomaten

Helmut Seidl
Fachbereich IV – Informatik
Universität Trier
D–54286 Trier
seidl@uni-trier.de

Christian Fecht
Universität des Saarlandes
Postfach 151150
D-66041 Saarbrücken
fecht@cs.uni-sb.de

Wir stellen einen gemeinsamen Ansatz für die automatische Programmanalyse sowohl für imperative wie logische Programmiersprachen vor. Dieser Ansatz basiert auf einer *small-step* operationalen Semantik. Sowohl die konkrete wie die abstrakte operationelle Semantik formalisieren wir mithilfe von (eingabefreien) *Kellerautomaten*. Diese Formalisierung ist nicht nur sehr elegant, sondern gestattet auch den Einsatz von Methoden der Formalen Sprachen, um Constraint Systeme abzuleiten, auf deren Lösungen letztendlich die Programmanalyse beruht.

Complexity of Array Languages[1]

Henning Fernau Markus Holzer
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Sand 13
D-72076 Tübingen
{fernau,holzer}@informatik.uni-tuebingen.de

Rudolf Freund
Institut für Computersprachen
Technische Universität Wien
Resselgasse 3
A-1040 Wien
rudi@logic.tuwien.ac.at

We consider the complexity of (d-dimensional) array Chomsky-type grammars, see [1, 2]. Briefly stated, we observe that there is a huge complexity gap between one-dimensional array languages (whose complexity results can be mostly inferred by the well-known string case) and the case of two and more dimensions.

There, as main results, we obtain that both fixed and general membership for regular and context-free array languages, even in the unary case, is NP-complete, while the nonemptiness problem is undecidable for regular array languages. This result nicely fits into the picture observed in other picture describing formalisms [3, 4].

Moreover, every recursively enumerable string language can be obtained via an at least two-dimensional #-context-free array grammar, so that the membership problem for this grammar and language type is undecidable, while it is PSPACE-complete for monotonic array grammars and languages.

Literatur

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

[2] R. Freund. Control mechanisms on #-context-free array grammars. In: Gh. Păun (ed.), *Mathematical Aspects of Natural and Formal Languages* (World Scientific Publ., Singapore, 1994), pp. 97–137.

[3] D. Giammarresi and A. Restivo. Two-dimensional finite state recognizability. *Fundamenta Informaticae*, 25:399–422, 1996.

[4] I.H. Sudborough and E. Welzl. Complexity and decidability for chain code picture languages. *Theoretical Computer Science*, 36:173–202, 1985.

[1]First author supported by Deutsche Forschungsgemeinschaft grant DFG La 618/3-1.

[5] J. van Leeuwen. The membership problem for ET0L-languages is polynomially complete. *Information Processing Letters*, 3(5):138–143, May 1975.

Graphgrammatiken als Basis von Programmsystemen und -entwicklungsumgebungen

Rudolf Freund Herbert Pötzl Tatjana Svizensky
Technische Universität Wien
Resselg. 3
A-1040 Wien
{rudi,herbert,tatjana}@logic.tuwien.ac.at

Seit mehr als 25 Jahren sind Graphgrammatiken in der Theorie der Formalen Sprachen vertreten, in der praktischen Informatik wurden sie jedoch erst geraume Zeit später als Alternative zu konventionellen Ansätzen entdeckt. Praktische Anwendungen existieren erst sehr vereinzelt (e.g. [5]), obwohl eine Vielzahl von Ideen und Konzepten zur Verwendung von Graphgrammatiken in Programmieranwendungen ausgearbeitet und zur Verfügung gestellt wurde ([1]).

Als Hauptkritikpunkte am praktischen Einsatz von Graphgrammatiken für Programme und Programmentwicklungen werden oft mangelnde Effizienz, geringe Übersichtlichkeit und unzureichende Flexibilität genannt.

Andererseits gibt es eine Vielfalt von Anwendungen für Graphgrammatiken: In vielen Problemstellungen der Informatik existieren bereits Strukturen, welche oftmals im Aufbau ohnehin graphenähnlich sind oder sich leicht in geeignete Form bringen lassen, sodass sich die Beschreibung, Darstellung und Bearbeitung in Form von Graphen geradezu anbietet. In solchen Fällen ist es unbestreitbar sinnvoll, das Prinzip der Graphgrammatiken zur Modellierung und Problemlösung heranzuziehen ([2], [3]).

Bekannte Beispiele für geeignete Bereiche sind etwa Bilderkennung, Datenbanken, Steuerungen oder neuronale Netze. In diesen Bereichen bringt der Ansatz mittels Graphgrammatiken eine erhöhte Übersichlichkeit und verbesserte Möglichkeit zur Visualisierung komplexer Zusammenhänge, welche im konventionellen Programmieransatz nicht in dieser Form gegeben ist.

Die Effizienz kann aus Mangel an praktischen Beispielen schwer bewertet werden, jedoch hängt die Gesamteffizienz eines Programmes oder Algorithmus nicht nur von der Ausführungseffizienz des Programmiersystems ab, welche zweifelsfrei optimiert werden kann, sondern auch von der Entwicklungseffizienz. Eine problemorientierte Entwicklungsumgebung kann daher die gesamte Effizienz erheblich erhöhen, sofern sie die Bedürfnisse des Entwicklers berücksichtigt. Eine Effizienzsteigerung in der Behandlung graphenähnlicher Strukturen eben in Form von Graphen ist sicherlich unbestreitbar.

Für jede Art von Programmentwicklung (ob auf Basis von Graphgrammatiken oder nicht) ist eine geeignete, durchschaubare, klar definierte Entwicklungsumgebung essentiell. Wir wollen nun eine mögliche Entwicklungsumgebung vorgestellen, die für die spezifischen Notwendigkeiten für die Entwicklung von auf Graphgrammatiken basierenden Programmsystemen konzipiert

ist.

UPGraDE ([4]) ist ein Beispiel für eine Entwicklungsumgebung, die den Entwickler durch eine universelle aber einfach zu verstehende Graphgrammatikprogrammiersprache befähigt, komplexe Graphgrammatikprogramme zu entwickeln und zu überprüfen.

Die zugrundeliegenden Graphproduktionen erlauben das Anlegen bzw. Löschen eines Knotens und das Abändern des Labels und der Attribute eines Knotens sowie das Anlegen bzw. Löschen einer Kante zwischen zwei Knoten und das Abändern des Labels und der Attribute einer Kante und der damit verbundenen Knoten.

Das Programm wird in Form eines Kontrollgraphen in das Graphsystem eingebracht und zur Laufzeit von einem Graphgrammatikinterpreter abgearbeitet. Dies ermöglicht unter anderem die Abänderung des Programmes zur Laufzeit durch das Programm selbst. Die Programmerstellung wird durch einen flexiblen Compiler unterstützt, der eine den gängigen höheren Programmiersprachen angepasste Syntax (Pascal, C, etc.) aufweist und damit auf dieser abstrakten Ebene strukturierte Programmierung ermöglicht.

Attributfunktionen werden in Form von Funktionsmodulen zur Verfügung gestellt, die bei Bedarf einfach erweitert oder mit beliebiger Komplexität kombiniert werden können.

Ein- und Ausgabekomponenten können einfach gestaltet und erweitert werden, sodass die Entwicklung einer geeigneten graphischen Benutzerschnittstelle sehr vereinfacht wird.

Ein spezielles Modul unterstützt durch Einzelschrittsimulation und ausführliche Darstellung des Graphsystems sowie der an einer Produktion beteiligten Komponenten das Auffinden von Programmierfehlern. Auch ist durch dieses Modul eine Präsentation und Erläuterung eines Programmes zu Lehrzwecken leicht möglich.

Alle Komponenten, selbst Compiler und Graphgrammatikinterpreter, sind modular aufgebaut und können daher bei Bedarf erweitert oder ausgetauscht werden.

Die Entwicklungsumgebung selbst basiert auf einem äusserst stabilen aber flexiblen Betriebsystem (OPENSTEP) und ist in einer objektorientierten Programmiersprache gestaltet (Objective-C). In Planung befinden sich folgende Erweiterungen:

1. Eine Makrosprache, die es dem Entwickler ermöglicht, sich seine eigenen Befehle selbst zu gestalten (eigene Programmiersprache).

2. Ein flexibles Konzept für Unterprogramme mit Parameterübergabe.

3. Ein erweitertes System zur raschen Erstellung von Attributfunktionen, sodass die Entwicklung von Funktionsmodulen weitgehendst überflüssig wird.

4. Ein graphisches Darstellungsmodul zur besseren Visualisierung der Graphstrukturen.

Literatur

[1] G. Engels, C. Lewerentz, and W. Schäfer: *Graph Grammar Engineering - A Software Specification Method.* H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld (editors): Graph Grammars and Their Applications to Computer Science. LNCS 291, Springer, 1987.

[2] R. Freund, B. Haberstroh, and C. Stary: *Applying graph grammars for task-oriented user interface development.* Proceedings IEEE Conference on Computing and Information ICCI'92, 361-365, 1992.

[3] R. Freund, *Software spezification based on graph grammars,* Fifth International Workshop on Graph Grammars, Williamsburg, USA, November 1994

[4] H. Pötzl, UPGraDE, Diplomarbeit, September 1997.

[5] A. Schürr: *A VHL-Language Based on Graph Grammars.* H. Ehrig, H.-J. Kreowski, and G. Rozenberg (editors): *Graph Grammars and Their Application to Computer Science.* LNCS 532, Springer, 1991.

Substitutionsbasierte Graphersetzung

Annegret Habel
Universität Hildesheim
Marienburger Platz 22
D-31141 Hildesheim
habel@informatik.uni-hildesheim.de

In diesem Vortrag wird ein substitutionsbasierter Ansatz zur Graphersetzung vorgestellt. Der in Plump und Habel [PH96] eingeführte Ansatz ist sehr einfach zu beschreiben und braucht weder Pushouts noch Einbettungsinstruktionen. Er umfaßt sowohl den Doppel-Pushout-Ansatz (DPO) zur Graphersetzung als auch klassische Termersetzung. Dabei korrespondiert jede direkte Ableitung in diesen Modellen zu einer direkten Ableitung im substitutionsbasierten Ansatz (SB-Ansatz). Bei Einschränkung des Ansatzes auf *elementare* Ersetzung erhält man sogar eine bijektive Korresondenz zwischen DPO-Ersetzung and elementarer SB-Ersetzung. Als Konsequenz hieraus lassen sich Resultate wie die Kommutativitäts- und Parallelismustheoreme des DPO-Ansatzes auf den elementaren SB-Ansatz übertragen. Für allgemeine SB-Ersetzung wird das "schwache Kommutativitätsproblem" untersucht. Für schwach nicht-überlappende Ableitungen $G \Rightarrow H$ durch r und $G \Rightarrow H'$ durch r', wobei r eine Kontext-löschende, -bewahrende oder -kopierende Regel und r' eine Kontext-bewahrende Regel ist, wird die Existenz von Ableitungen $H \Rightarrow^* X$ durch r' und $H' \Rightarrow^* X$ durch r gezeigt.

Literatur

[PH96] Detlef Plump, Annegret Habel. Graph unification and matching. In J. Cuny, H. Ehrig, G. Engels, G. Rozenberg, eds., Graph Grammars and Their Application to Computer Science, Lecture Notes in Computer Science 1073, 75–89, 1996.

Elementare Ersetzung in Hypergraphen

Renate Klempien-Hinrichs
Universität Bremen

Graphersetzungssysteme sind ein mächtiges Modell zur Formalisierung von Berechnungen auf Graphen und zur Spezifikation von Graphsprachen. Eine verbreitete Einschränkung besteht darin, die Ersetzung von atomaren Einheiten eines Graphen (oder Hypergraphen) – d.h. von Knoten bzw. von (Hyper)Kanten – zu betrachten.

Knotenersetzungssysteme erlauben es, Knoten eines Graphen durch einen Graphen zu ersetzen und diesen neuen Graphen durch Kanten mit dem Rest des alten Graphen zu verbinden und somit einzubetten. Durch die Einbettung des neuen Graphen in den Rest des alten Graphen kann es zu einer Vervielfältigung von Kanten kommen. Eine typische Sprache, die sich durch eine Knotenersetzungsgrammatik, jedoch nicht durch eine Hyperkantenersetzungsgrammatik erzeugen läßt, ist die Menge aller vollständigen Graphen.

Hyperkantenersetzungssysteme erlauben es, Hyperkanten eines Hypergraphen durch einen Hypergraphen zu ersetzen. Hierbei werden ausgezeichnete Knoten des neuen Hypergraphen mit den Anknüpfungspunkten der Hyperkante im Restgraphen verklebt. Insbesondere können selektiv Hyperkanten gelöscht werden. Eine typische Sprache, die sich durch eine Hyperkantenersetzungsgrammatik erzeugen läßt, ist die Menge aller partiellen k-Bäume (für ein festes k).

In diesem Vortrag wird eine Kombination von Knoten- und Hyperkantenersetzung in Hypergraphen, die sogenannte *Atomersetzung*, vorgestellt. Dabei wird eine atomare Einheit durch einen Hypergraphen ersetzt, indem dieser mit dem Resthypergraphen sowohl durch Ziehen von Einbettungshyperkanten als auch durch Verkleben von Knoten verbunden wird. Durch die Kombination der Ersetzungstechniken in einen Formalismus läßt sich z.B. die Sprache aller Graphen, die vollständig oder ein partieller k-Baum sind, durch eine Atomersetzungsgrammatik auf einfache Art und Weise erzeugen.

Es zeigt sich, daß sowohl Hyperkantenersetzung (siehe [Hab92]) als auch Knotenersetzung in Hypergraphen (siehe [Kle96]) Spezialfälle der Atomersetzung sind und daß Atomersetzungssprachen gegenüber Vereinigung abgeschlossen sind. Dies erlaubt es, die Vorteile der Knotenersetzung und die der Hyperkantenersetzung zu nutzen: Wenn viele Kanten gezogen werden müssen, kann Knotenersetzung verwendet werden und wenn selektiv Kanten gelöscht werden müssen, läßt sich dies mit Hyperkantenersetzung bewerkstelligen. Da konfluente Knotenersetzung in Hypergraphen die *separierten Hantelgrammatiken* aus [CER93] verallgemeinert, gilt dies auch für Atomersetzung; der Zusammenhang zwischen allgemeiner Hantelersetzung und Atomersetzung ist dagegen noch ungeklärt.

Literatur

[CER93] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46:218–270, 1993.

[Hab92] Annegret Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1992.

[Kle96] Renate Klempien-Hinrichs. Node replacement in hypergraphs: Simulation of hyperedge replacement, and decidability of confluence. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Proc. Fifth Intl. Workshop on Graph Grammars and Their Application to Comp. Sci.*, volume 1073 of *Lecture Notes in Computer Science*, pages 397–411. Springer, 1996.

[KR90] Hans-Jörg Kreowski and Grzegorz Rozenberg. On structured graph grammars, parts I and II. *Information Sciences*, 52:185–210 and 221–246, 1990.

Kontextfreie und kontextsensitive Collagen-Grammatiken

Hans-Jörg Kreowski
Fachbereich 3 – Mathematik/Informatik
Universität Bremen
Postfach 33 04 40
D–28334 Bremen
kreo@informatik.uni-bremen.de

Collagen-Grammatiken sind Regelsysteme zur Erzeugung d-dimensionaler Bildsprachen. Eine Collage besteht aus einer Menge von Teilen, d.h. Teilmengen des $\mathbb{R}^d$, deren Vereinigung das zugehörige Bild liefert. Der in Collagen-Grammatiken benutzte Generierungsmechanismus basiert auf der Ersetzung von Hyperkanten – atomaren nichtterminalen Objekten, die in ähnlicher Form auch auf dem Gebiet der Graphgrammatiken Verwendung finden. Die linke Seite einer Produktion besteht im kontextfreien Fall aus einer einzelnen Hyperkante, während die linke Seite einer kontextsensitiven Produktion zusätzlich einen Kontext enthalten kann, der zur Anwendung einer Regel in der Collage vorhanden sein muß (aber selbst durch die Produktion nicht verändert wird).

Kontextfreie Collagen-Grammatiken wurden bereits in einer Reihe von Arbeiten untersucht (siehe z.B. [HKT93, DHKT95, DK96, Dre96]). Überraschend schwierig gestaltet sich selbst bei dieser Variante der Nachweis, daß bestimmte Collagensprachen nicht generierbar sind. Der Grund dafür ist, daß für Collagensprachen bisher kein als Kriterium für Kontextfreiheit nutzbares Pumpinglemma bekannt ist. Inzwischen konnten allerdings zwei andere Kriterien solcher Art gefunden werden [DKL97]. Diese betreffen die Anzahl der Teile generierter Collagen, welche höchstens linear schnell wachsen kann, und das Volumen der in einer Sprache vorkommenden Teile. Einfach ausgedrückt, wächst das Volumen (sofern es überhaupt wächst) stets exponentiell.

Mit Hilfe des ersten Kriteriums kann man jetzt auch die Klasse der kontextsensitiven von der der kontextfreien Collagensprachen trennen, was vorher wegen fehlender trennender Beispiele nicht möglich war.

Literatur

[DHKT95] Frank Drewes, Annegret Habel, Hans-Jörg Kreowski, and Stefan Taubenberger. Generating self-affine fractals by collage grammars. *Theoretical Computer Science*, 145:159–187, 1995.

[DK96] Frank Drewes and Hans-Jörg Kreowski. (Un-)decidability of geometric properties of pictures generated by collage grammars. *Fundamenta Informaticae*, 25:295–325, 1996.

[DKL97] Frank Drewes, Hans-Jörg Kreowski, and Denis Lapoire. Criteria to disprove context-freeness of collage languages. In B.S. Chlebus and L. Czaja, editors, *Proc. Fundamentals of Computation Theory XI*, volume 1279 of *Lecture Notes in Computer Science*, pages 169–178, 1997.

[Dre96] Frank Drewes. Language theoretic and algorithmic properties of d-dimensional collages and patterns in a grid. *Journal of Computer and System Sciences*, 53:33–60, 1996.

[HKT93] Annegret Habel, Hans-Jörg Kreowski, and Stefan Taubenberger. Collages and patterns generated by hyperedge replacement. *Languages of Design*, 1:125–145, 1993.

Acceptance modes for asynchronous cellular automata for pomsets and first order logic[1]

Dietrich Kuske[2]
Institut für Algebra
Technische Universität Dresden
D-01062 Dresden
kuske@math.tu-dresden.de

In a distributed system, some events may occur concurrently, meaning that they may occur in any order or simultaneously or even that their executions may overlap. This is the case in particular when two events use independent resources. On the other hand, some events may causally depend on each other. Therefore, a distributed behaviour may be abstracted as a pomset, that is a set together with a partial order which describes causal dependencies of events and with a labeling function. In this paper, we consider pomsets without autoconcurrency, i.e. events with the same label are linearly ordered. These pomsets are called semi-words in [St81, Di94]. For studies how general pomsets can be used to represent parallel processes and how they can be composed, we refer the reader e.g. to [Pr87, Gi88].

There are several ways to describe the behaviour of a system. For instance, logic formulas are suited for specification issues. Depending on the properties one has to express, we can use various logics such as temporal logics, the first order logic or the (monadic) second order logic.

When dealing with distributed systems, it is natural to look for transition systems that faithfully reflect the concurrency. For instance, Petri nets are a widely studied class of such transition systems. Asynchronous cellular automata (ACA) form another fundamental class of transition systems with built-in concurrency. They were introduced for Mazurkiewicz traces by Zielonka [Zi87, Zi89]. A trace ([CF69, Ma77, Ma87]) is a pomset where the partial order is dictated by a static dependence relation over the actions of the system.

In [DG96], Droste & Gastin generalize the notion of ACA's so that they can work on pomsets without autoconcurrency. They consider the relation between monadic second order logic on pomsets, ACA's and deterministic ACA's. Their first result states that any set of pomsets recognized by an ACA can be defined by an existential sentence of the monadic second order (MSO) logic. The question, whether the converse holds in general is left open. It is proved for a large class of pomsets, called CROW-pomsets. Actually, Droste & Gastin were able to construct from an arbitrary MSO-sentence a *deterministic* ACA which accepts precisely the CROW-pomsets satisfying the

[1]For the complete paper see [Ku97]

[2]Supported by the German Research Foundation (DFG).

formula. Thus, they proved the equivalence between MSO logic, existential MSO logic, deterministic ACA's and nondeterministic ACA's.

In [DG96, Remark 3.3] the authors shortly discuss several alternative definitions of acceptance by ACA's and state: Ïn fact, these differences are not crucial and we will see ... that under some assumptions they are equivalent.Therefore, they mainly restrict their attention to only one of the four suggested definitions.

While, as remarked above, the question whether any MSO-sentence can be translated into an equivalent ACA is left open in [DG96], we answer it negatively for all possible acceptance modes of ACA's in this paper. More precisely, we show that there exists a first order sentence such that the set of all pomsets satisfying this sentence is not recognizable by an ACA. Since, as we also show, the complement of this set is recognizable in any mode, we obtain as a corollary that for any acceptance mode the class of recognizable pomset languages is not closed under complement. Hence not every nondeterministic ACA can be transformed into an equivalent deterministic one. A similar result has been obtained by Gastin for one of the acceptance modes [Ga].

Furthermore, we show that the differences between the acceptance modes become crucial if we consider the set of all pomsets. More precisely, we show that they yield mutually different classes of recognizable pomset languages and we completely determine the inclusion structure between these classes.

Literatur

[CF69] P. Cartier and D. Foata: *Problemes combinatoires de commutation et rearrangements.* Lecture Notes in Mathematics 85, 1969.

[Di94] V. Diekert: *A partial trace semantics for petri nets.* Theoretical Computer Science **113** (1994), pp. 87-105.

[DG96] D. Droste, P. Gastin: *Asynchronous cellular automata for pomsets without auto-concurrency.* In: CONCUR 96. Lecture Notes in Computer Science 1119, pp. 627-638, 1996.

[Gi88] J.L. Gischer: *The equational theory of pomsets.* Theoretical Computer Science **61** (1988), pp. 199-224.

[Ga] P. Gastin: private communication.

[Ku97] D. Kuske: *Comparing asynchronous cellular automata for pomsets and first order logic.* Technical Report, TU Dresden, `http://www.math.tu-dresden.de/~kuske/abstracts/aca.html`, 1997.

[Ma77] A. Mazurkiewicz: *Concurrent program schemes and their interpretation.* Tech. rep. DAIMI PB 78, Aarhus University, 1977.

[Ma87] A. Mazurkiewicz: *Trace Theory.* In: Advances in Petri Nets 86 (W. Brauer et al., eds.), Lecture Notes in Computer Science 255, pp. 279-324, 1987.

[Pr87] V.R. Pratt: *Modelling concurrency with partial orders.* J. of Parallel Programming **15** (1987), pp. 33-71.

[St81] P.H. Starke: *Processes in Petri Nets.* EIK **17** (1981), pp. 389-416.

[Zi87] W. Zielonka: *Notes on finite asynchronous automata.* R.A.I.R.O. – Informatique Théorique et Applications **21** (1987), pp. 99-135.

[Zi89] W. Zielonka: *Safe executions of recognizable trace languages by asynchronous automata.* In: Logical Foundations of Computer Science (A.R. Meyer et al., eds.), Lecture Notes in Computer Science 363, pp. 278-289, 1989.

Dichte Vollständigkeit

Klaus-Jörn Lange
Universität Tübingen

Die Theorie der formalen Sprachen und die Strukturelle Komplexitätstheorie weisen enge Beziehungen zueinander auf. Obwohl beide Theorien vergleichbare Fragestellungen verfolgen wie beispielsweise den Vergleich von Determinismus und Nichtdeterminismus, weisen beide sehr unterschiedliche Erfolge in der Beantwortung dieser Fragestellungen auf. Während die Komplexitätstheorie zu großen Teilen eine Ansammlung offener Fragen ist und die Mehrzahl ihrer Ergebnisse offene Fragen in Verbindung setzt, ist das Phänomen offener Fragen in der Theorie formaler Sprachen nahezu unbekannt. Es stellt sich also die Frage nach der Enge ihrer wechselseitigen Beziehungen.

Nahezu alle formalen Sprachen, die in irgendeiner Weise kontextfrei sind, weisen Beziehungen zu den Klassen NP, $NAuxPDA_{pt}$, $NSPACE(\log n)$, oder NC^1 auf. Entsprechende Beziehungen gibt es zwischen den deterministischen Versionen dieser Familien und den Klassen P, $DAuxPDA_{pt}$, $DSPACE(\log n)$, und NC^1. Die Art dieser Beziehungen ist dabei derart, daß eine Familie formaler Sprachen $\mathcal{A}$ in einer Komplexitätsklasse $\mathcal{B}$ enthalten ist und daß $\mathcal{A}$ eine $\mathcal{B}$–vollständige Sprache enthält. Durch eine Beobachtung von Richard Beigel ist es nun möglich, systematisch zu einer formalen Sprachfamilie $\mathcal{A}$, die durch sequentielle einweg–Automaten definiert ist und für die die Klasse $\mathcal{B}$ unter logspace–Reduktionen abgeschlossen ist, einen nichtdeterministischen einweg–Automatentyp derart zu konstruieren, daß für die entsprechende Sprachfamilie $\mathcal{A}'$ gilt $\mathcal{A} \subseteq \mathcal{A}' \subset \mathcal{B}$ sowie

$$\forall_{B \in \mathcal{B}} \exists_{A \in \mathcal{A}'} : LOG(A) = LOG(B).$$

Die Familie $\mathcal{A}'$ liegt also *dicht* in der Klasse $\mathcal{B}$. Die Konstruktion und einige ihrer Eigenschaften werden im Vortrag vorgestellt. Es ist offen, ob eine vergleichbare Konstruktion auch für deterministische Familien oder für Klassen unterhalb von $DSPACE(\log n)$ gefunden werden kann.

Dieses Dokument wurde nach alten Rechtschreibregeln verfaßt. Zu Risiken und Nebenwirkungen

On the class of Church-Rosser languages

Gundula Niemann Friedrich Otto
Fachbereich Mathematik/Informatik
Universität Kassel
D–34109 Kassel
{niemann,otto}@theory.informatik.uni-kassel.de

For a finite, length-reducing, and confluent string-rewriting system the word problem can be solved in linear time by the so-called normal form algorithm [Boo82]. Motivated by this fact McNaughton et al [MNO88] introduced the *Church-Rosser languages*. A Church-Rosser language $L \subset \Sigma^*$ is given through a finite, length-reducing, and confluent string-rewriting system R on some alphabet Γ properly containing Σ, two irreducible strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^*$, and an irreducible letter $Y \in \Gamma \setminus \Sigma$ satisfying the following condition for all strings $w \in \Sigma^*$:

$$w \in L \text{ if and only if } t_1 w t_2 \to_R^* Y.$$

Hence, the membership problem for a Church-Rosser language is decidable in linear time. It follows immediately that the class CRL of Church-Rosser languages is contained in the class CSL of context-sensitive languages.

It remained open, however, whether the class CRL is closed under complement. Accordingly, McNaughton et al introduced the class of *Church-Rosser decidable languages* CRDL. This subclass of the class CRL contains the class DCFL of deterministic context-free languages, and it is closed under complement. Also it remained open at the time whether or not every context-free language is a Church-Rosser language, although it was conjectured that not even the linear language $L_0 := \{ww^\sim | w \in \{a,b\}^*\}$ is a Church-Rosser language. Here $w^\sim$ denotes the reversal of the string w.

These questions remained open until another, seemingly unrelated development had taken place. Dahlhaus and Warmuth [DaWa86] considered the class GCSL of *growing context-sensitive languages*. These languages are generated by context-sensitive grammars each production rule of which is strictly length-increasing. They proved that these languages have membership problems that are decidable in polynomial time. Although it might appear from the definition that GCSL is not an interesting class of languages, Buntrock and Loryś showed that GCSL is an *abstract family of languages* [BuLo92]. Exploiting these closure properties they characterized the class GCSL through various other classes of grammars that are less restricted [BuLo92, BuLo94].

Using these grammars Buntrock and Otto [BuOt95] obtained a characterization of the class GCSL by a nondeterministic machine model, the so-called *shrinking pushdown automaton with two pushdown stores* (sTPDA). The input for such a machine is provided as the intial contents of one of the pushdown stores, and it accepts either by final state or (equivalently) by empty

pushdown stores. A positive weight is assigned to each tape symbol and each internal state symbol of the machine. By adding up the weights this gives a weight for each configuration. Now it is required that the weight of the actual configuration decreases with each step of the machine. It is with respect to these weights that the two-pushdown automaton is called *shrinking*.

Since the sTPDA is a nondeterministic device, it was only natural to consider the class of languages that are accepted by the deterministic variant of it. As it turned out the deterministic sTPDA accept exactly the so-called *generalized Church-Rosser languages*, which are obtained from the Church-Rosser languages by admitting finite, *weight-reducing*, and confluent string-rewriting systems in the definition [BuOt95]. Thus, the class GCRL of generalized Church-Rosser languages coincides with the class of 'deterministic growing context-sensitive languages.' In particular, it follows that this class is closed under complement. Further, Buntrock and Otto concluded from this result that the language classes CFL and GCRL, and therewith the classes CFL and CRL, are indeed incomparable under set inclusion. Finally, this yields the following chain of inclusions:

$$\mathsf{DCFL} \subset \mathsf{CRDL} \subseteq \mathsf{CRL} \subseteq \mathsf{GCRL} \subset \mathsf{GCSL} \subset \mathsf{CSL},$$

where it was left open whether or not the two inclusions $\mathsf{CRDL} \subseteq \mathsf{CRL} \subseteq \mathsf{GCRL}$ are proper or not.

Here we show that none of these two inclusions is actually a proper one, that is, the three language classes CRDL, CRL, and GCRL all coincide. Our proof makes use of the above-mentioned characterization of the generalized Church-Rosser languages through the deterministic sTPDA. We will prove that each language that is accepted by some deterministic sTPDA is actually a Church-Rosser decidable language. Hence, $\mathsf{GCRL} \subseteq \mathsf{CRDL}$ implying that the three classes above actually coincide. Hence, the class of Church-Rosser languages can be interpreted as the class of deterministic growing context-sensitive languages.

Literatur

[Boo82] R.V. Book. Confluent and other types of Thue systems. *J. Association Computing Machinery*, 29:171–182, 1982.

[BuLo92] G. Buntrock and K. Loryś. On growing context-sensitive languages. In W. Kuich, editor, *Proc. of 19th ICALP*, Lecture Notes in Computer Science 623, pages 77–88. Springer-Verlag, Berlin, 1992.

[BuLo94] G. Buntrock and K. Loryś. The variable membership problem: Succinctness versus complexity. In P. Enjalbert, E.W. Mayr, and K.W. Wagner, editors, *Proc. of STACS 94*, Lecture Notes in Computer Science 775, pages 595–606. Springer-Verlag, Berlin, 1994.

[BuOt95] G. Buntrock and F. Otto. Growing context-sensitive languages and Church-Rosser languages. In E.W. Mayr and C. Puech, editors, *Proc. of STACS 95*, Lecture Notes in Computer Science 900, pages 313–324. Springer-Verlag, Berlin, 1995.

[DaWa86] E. Dahlhaus and M. Warmuth. Membership for growing context-sensitive grammars is polynomial. *J. Computer System Sciences*, 33:456–472, 1986.

[MNO88] R. McNaughton, P. Narendran, and F. Otto. Church-Rosser Thue systems and formal languages. *Journal Association Computing Machinery*, 35:324–344, 1988.

Rich ω-words and monadic second-order arithmetic

Ludwig Staiger
Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
Kurt-Mothes-Str. 1
D-06120 Halle (Saale)

Rich ω-words are one-sided infinite strings which have every finite word as a subword (infix).

Regular ω-words are one-sided infinite strings for which the infix-set of a suffix is a regular language. This implies that the set of infinitely often occurring infixes equals the infix-set of a suitably chosen suffix of the regular ω-word.

We show that for a regular ω-language F (a set of predicates definable in monadic second order arithmetic) the following conditions are equivalent:

1. F contains a rich ω-word.
2. F is of second baire category in the Cantor space of ω-words.
3. F is a nonnull set for a class of measures (including the natural Lebesgue measure on Cantor space).
4. F has maximum Hausdorff dimension.

This shows that, although we cannot translate Compton's result on rich Z-words (in the MSO theory of the integers) to MSO arithmetic, a set definable in MSO arithmetic and containing a dijunctive ω-word is large in several respect simultaneously.

Moreover, for we show under the assumption of an exchanging property for 'distinguishing' prefixes that two regular ω-words not necessarily being rich but having the same sets of infixes occurring infinitely often are indistinguishable by MSO formulas or, equivalently, by finite automata.