Technical Reports Mathematics/Computer Science
FB IV - Department of Computer Science
University of Trier
54296 Trier

Theorietage 2015

Algorithmen und Komplexität & Automaten und Formale Sprachen

(Speyer, 29.09. - 02.10.2015)

Herausgeber: Henning Fernau

Technical Report No. 15-1 September 2015

Vorwort

Theorietage haben eine lange Tradition bei verschiedenen Fachgruppen der Gesellschaft für Informatik (GI). Die Fachgruppe *Algorithmen* (ALGO), vor fünf Jahren entstanden aus einer Fusion der beiden bis dahin existierenden Fachgruppen Algorithmen und Datenstrukturen (ADS) und Parallele und Verteilte Algorithmen (PARVA), und die Fachgruppe *Komplexität* (KP) veranstalten üblicherweise zwei- oder dreimal jährlich gemeinsam einen in der Regel ein- bis zweitägigen Workshop über Algorithmen und Komplexität. In dieser Reihe ist das Treffen in Speyer die 70. Auflage. Die Fachgruppe *Automaten und Formale Sprachen* (AFS) trifft sich einmal jährlich zu ihrem zweitägigen Theorietag. In diesem Jahr feiert die Reihe ihr Silberjubiläum in Speyer. Seit zwanzig Jahren gibt es auch die Tradition, einen Workshop mit eingeladenen Vorträgen zu einem übergreifenden Themenbereich dem AFS-Theorietag voranzustellen. Beide Fachgruppen gehören zum Fachbereich *Grundlagen der Informatik* (GInf) innerhalb der GI.

Beide Workshopserien haben zum Ziel, insbesondere dem wissenschaftlichen Nachwuchs ein Forum zu bieten, auf dem sie ihre Ergebnisse der nationalen Fachöffentlichkeit vorstellen können. Näheres zur Geschichte dieser Serien findet man bei den Internetauftritten der beiden Fachgruppen. Leider führt die starke Zergliederung der Informatik heutzutage dazu, dass es sehr viele spezialisierte Treffen der unterschiedlichsten (oft doch recht kleinen) internationalen Fachöffentlichkeiten (scientific communities) gibt, wobei dann die verbindenden Elemente verloren gehen. Wir möchten mit der bewusst gemeinsamen Veranstaltung der Theorietage zweier GI-Fachgruppen dem etwas entgegenwirken. Als Brücke soll ein "Zwischenworkshop" mit fünf eingeladenen Vorträgen dienen. Diese beleuchten verschiedene Aspekte, die Algorithmen und Komplexität mit Automaten und Formalen Sprachen verknüpfen. Als Vortragende konnten wir gewinnen:

- Till Tantau, Lübeck;
- Ralf Treinen, Paris;
- Christian Komusiewicz, Berlin;
- Henrik Björklund, Umeå;
- Laura Kallmeyer, Düsseldorf.

Dadurch konnten wir ein insgesamt dreieinhalbtägiges wissenschaftliches Programm zusammenstellen.

1. 70. Workshop über Algorithmen und Komplexität mit elf Beiträgen;
2. Zwischenworkshop mit fünf eingeladenen Vorträgen;

3. 25. Theorietag AFS (Teil 1) mit zehn Beiträgen;

4. 25. Theorietag AFS (Teil 2) mit sieben Beiträgen.

Außerdem haben wir im Programm Zeiten zur Sammlung offener Fragen sowie für das alljährliche Treffen der Fachgruppe AFS vorgesehen und überdies einen Besuch im Technikmuseum in Speyer eingeplant. Nähere Einzelheiten entnehme man bitte dem Tagungsprogramm. In dem hier vorgelegten Tagungsband finden Sie ein- bis vierseitige Kurzfassungen der "nichteingeladenen" Vorträge. Dieses folgt der Tradition der AFS-Theorietage. Allerdings werden wir auf der Tagung selbst nur drittelseitige Zusammenfassungen austeilen, der Tagungsband wird elektronisch zur Verfügung gestellt, soll aber bei Bedarf ein Nachschlagen der vorgestellten Resultate ermöglichen.

Um dem Charakter einer Nachwuchsveranstaltung deutlicher zu entsprechen, sind wir diesmal bewusst in eine Jugendherberge als Veranstaltungsort gegangen. Speyer haben wir ausgewählt, weil dieser Ort doch merklich einfacher für die meisten Teilnehmer zu erreichen ist als Trier (oder ein Ort in der Umgebung von Trier), insbesondere, da dort mittlerweile keine Fernverkehrszüge mehr halten. Die "Rheinschiene" ist doch deutlich besser angeschlossen.

Wir danken der Gesellschaft für Informatik und dem MDPI-Verlag für die freundliche Unterstützung. Dadurch wurde es möglich, die Reisekosten der eingeladenen Vortragenden abzudecken. Überdies ist geplant, Fassungen der eingeladenen Vortragenden in einem Sonderband der Zeitschrift *Algorithms* des MDPI-Verlages erscheinen zu lassen. Das ermöglicht eine noch deutlich ausführlichere Darstellung der vorgestellten Konzepte und Ergebnisse im Vergleich zu diesem Tagungsband und auch eine bessere Sichtbarkeit der Beiträge.

Unser besonderer Dank gilt Heike Beewen für die organisatorische und Katrin Casel und Markus Schmid für die programmatische Begleitung der Vorbereitung und Durchführung der Theorietage. Wir wünschen allen Teilnehmern eine interessante und anregende Tagung sowie einen angenehmen Aufenthalt in Speyer. Hoffentlich finden sie auch Zeit, diesen geschichtsträchtigen Ort zu erkunden.

Trier, im September 2015 Henning Fernau

Inhaltsverzeichnis

Vorträge am Dienstag, 29. September 2015

Theorietage der GI in Speyer: Algorithmen und Komplexität

1. Sitzung: 9:00 bis 10:30

- H. Fernau: Begrüßung
- M. Laudahn: Platzeffiziente Euler-Partitionierung und bipartite Kantenfärbung
- M. Gobbert: Edge Hop
- F. Kammer: Exploration von Temporalen Graphen

2. Sitzung: 11:00 bis 12:15

- K. Casel: Clustering mit unteren Schranken
- A. Nichterlein: On Finding Points in General Position
- M. Blumenstock: Pseudoarboricity or Lowest Maximum Indegree Orientations

3. Sitzung: 14:00 bis 15:15

- M. L. Schmid: A New Parameterisation for Consensus Strings Problems
- R. Freund: Antimaterie in Komplexitätsklassen von Membransystemen
- A. Jaax: Der GapSet Algorithmus

4. Sitzung: 15:45 bis 17:00

- R. Reitzig: A Simple and Fast Linear-Time Algorithm for Proportional Apportionment
- S. Wild: News on Quicksort
- Sammlung offener Fragen

Vorträge am Mittwoch, 30. September 2015

Theorietage der GI in Speyer: Zwischen den Welten

1. Sitzung: 8:30 bis 9:45

- H. Fernau: Begrüßung
- T. Tantau: Algorithmische Metatheoreme 2.0

2. Sitzung: 10:15 bis 12:15

- R. Treinen: Towards reasoning about file trees using tree automata
- C. Komusiewicz: Multivariate Algorithmics for NP-hard String Problems

3. Sitzung: 13:45 bis 15:45

- H. Björklund: On the complexity of mildly context-sensitive formalisms
- L. Kallmeyer: LCFRS: Parsing und Anwendungen der Computerlinguistik

Besuch im Technikmuseum ab 16:00

Vorträge am Donnerstag, 01. Oktober 2015

Theorietage der GI in Speyer: Automaten und Formale Sprachen

1. Sitzung: 9:00 bis 10:30

- H. Fernau: Begrüßung
- M. L. Schmid: Revisiting Shinohara's Algorithm for Computing Descriptive Patterns
- C. Albrecht: Die Lamplighter-Gruppe $\mathbb{Z}_3 \wr \mathbb{Z}$ als Automatengruppe
- A. Maletti: Linking Theorems for Tree Transducers

2. Sitzung: 11:00 bis 12:15

- D. D. Freydenberger: RegEx und Wortgleichungen: Ein Vergleich
- F. Manea: Longest Gapped Repeats and Palindromes
- G. Zetzsche: An Approach to Computing Downward Closures

3. Sitzung: 14:00 bis 16:00

- B. Truthe: Zu Beziehungen zwischen subregulären Sprachfamilien
- D. Kuske: Infinite and Bi-infinite Words with Decidable Monadic Theories
- K. Meckel: Parameterized Prefix Distance Between Regular Languages
- P. Babari: Characterization of $+\omega$-picture languages
- Sammlung offener Fragen

GI-Fachgruppensitzung: ab 16:30

Traditionelles GI-Fachgruppenkegeln: ab 19:45

Vorträge am Freitag, 02. Oktober 2015

Theorietage der GI in Speyer: Automaten und Formale Sprachen

Aus-Checken bis 9:00

1. Sitzung: 9:00 bis 10:15

- R. Freund: Kontrollierte kontextuelle Array-Grammatiken
- M. Paramasivan: Scanning Pictures The Boustrophedon Way
- K. Reinhardt: Emptiness for Context-free Picture Languages is undecidable

2. Sitzung: 10:35 bis 12:15

- Q. Wang: Weighted Restarting Automata and Pushdown Relations
- K. Kwee: On Nondeterminism on Ordered Restarting Automata
- M. Wendlandt: Tinput-Driven Pushdown Automat
- M. Hahn: Visibly-Counter-Sprachen und die Struktur von NC^1

Mittagessen und Heimfahrt

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 9.

Exploration von Temporalen Graphen

Moritz Laudahn[(A)]

[(A)]Universität Augsburg, Lehrstuhl für Theoretische Informatik
moritz.laudahn@informatik.uni-augsburg.de

Zusammenfassung

Der Vortrag behandelt platzeffiziente Algorithmen für zwei Probleme auf ungerichteten Multigraphen, nämlich Euler-Partitionierung (die Partitionierung der Kantenmenge in eine minimale Anzahl von Pfaden) und bipartite Kantenfärbung. Im Fall der Euler-Partitionierung gelingt es, den Speicherbedarf unter Beibehaltung der linearen Laufzeit um einen logarithmischen Faktor zu reduzieren. Für die bipartite Kantenfärbung erreichen wir eine Laufzeit von $O(n+m\min\{\Delta,\log m\})$ unter Nutzung von $O(n+m)$ Bits an Arbeitsspeicher, wobei n, m und Δ die Anzahl der Knoten, die Anzahl der Kanten sowie den maximalen Grad des Multigraphen bezeichnen. Der Vortrag beruht auf gemeinsamer Arbeit mit Torben Hagerup und Frank Kammer.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 10–13.

Edge Hop – Ein Modell zur Komplexitätsanalyse von kombinatorischen Spielen

Moritz Gobbert[(A)]

[(A)]Universität Trier
Behringstraße 21, D-54296 Trier
gobbert@informatik.uni-trier.de

Zusammenfassung

In meinem Vortrag werde ich ein Spiel namens *Edge Hop* vorstellen, bei dem es darum geht einen markierten Spielstein auf einem Graphen von einem Startknoten zu einem Zielknoten zu bewegen. Hierbei hat der Graph bestimmte Eigenschaften. Beispielsweise befinden sich auf ihm zusätzliche Spielsteine und jeder Knoten kann nur eine bestimmte Anzahl an Steinen aufnehmen. Weiter werde ich zeigen, dass EDGEHOP, also die Frage, ob der markierte Spielstein den Zielknoten erreichen kann, *NP-vollständig* ist. Für die Reduktion werde ich verschiedene Gadgets konstruieren, die ein Gerüst für die Komplexitätsanalyse anderer (kombinatorischer) Spiele bzw. ähnlicher Fragestellungen bieten.

1. Einleitung

Spiele sind ein in der Komplexitätsheorie stark untersuchter Bereich. Hierbei werden verschiedene Arten von Spielen, bspw. kombinatorische Geduldsspiele, Zweipersonenspiele, Videospiele oder sogar „Keinpersonen"-Spiele untersucht. Aufgrund dieser Vielfalt ist es wünschenswert, die Komplexität gleichartiger Spiele auf „ähnliche Weise" bestimmen zu können. Hierzu haben Hearn und Demaine 2005 ein Modell entwickelt, welches als Rahmen für Schwere-Beweise fungiert – die sogenannte *Nondeterministic Contraint Logic (NCL)* [1]. Die NCL ist in ihrer ursprünglichen Version ein Modell zum Zeigen von $PSPACE$-Schwere, aber es gibt mittlerweile Varianten für andere Komplexitätsklassen. *Edge Hop* soll nun ein weiteres solches Modell sein, um die NP-Schwere von kombinatorischen Spielen zu zeigen. Im Gegensatz zur NCL orientiert sich Edge Hop näher an Bewegungsplanungs-Problemen, so dass sich solche Problemstellungen natürlicher in Edge Hop einbetten lassen.

Im folgenden Abschnitt werde ich Edge Hop selbst vorstellen und kurz skizzieren, wie man die NP-Vollständigkeit zeigen kann. Danach werde ich darauf eingehen, wie man Edge Hop nutzen kann, um die NP-Schwere anderer Probleme zu zeigen.

2. Edge Hop

2.1. Definitionen

Edge Hop ist ein kombinatorisches Geduldsspiel, bei dem es darum geht auf einem Graphen mit besonderen Eigenschaften einen Spielstein (*aktiver Stein*) zu einem ausgezeichneten Knoten (*Zielknoten*) zu ziehen. Hierbei darf der aktive Stein höchstens einmal zwischen adjazenten Knoten ziehen.

Zusätzlich zum aktiven Stein befinden sich auf dem Graphen noch *passive Steine*. Diese können ebenfalls nur zu adjazenten Knoten gezogen werden. Hier gibt es allerdings noch die weitere Einschränkung, dass passive Steine nur zu einem Knoten gezogen werden dürfen, wenn sich auf diesem der aktive Stein befindet. Passive Steine dürfen allerdings beliebig oft über Kanten ziehen.

Eine letzte Eigenschaft des Graphen ist eine Art maximale Kapazität der Knoten, genannt *maximale Belegtheit*. Auf einem Knoten dürfen höchstens so viele Spielsteine (hierzu zählen der aktive als auch die passiven Steine) liegen, wie seine maximale Belegtheit ist.

Formal ist ein Edge-Hop-Graph ein 5-Tupel $\mathcal{EH} = (G, k_G, a, z, P_G)$. Hierbei ist $G = (V, E)$ ein einfacher (zusammenhängender) Graph, $k_G : V \to \mathbb{N},\ k_G(v) = k_v$ eine Funktion, die jedem Knoten eine maximale Belegtheit zuordnet, $a \in V$ der Knoten, auf dem der aktive Stein zu Beginn liegt, $z \in V$ der Zielknoten und P_G eine Multimenge über V, welche für jeden passiven Stein den Knoten beinhaltet, auf dem er liegt.

2.2. Komplexität

EDGEHOP
Eingabe: Ein beliebiger Edge-Hop-Graph.
Frage: Ist es möglich, den aktiven Stein von a nach z zu ziehen, ohne die Regeln von Edge Hop zu verletzen?

Um die NP-Mitgliedschaft dieses Problems zu zeigen, argumentiert man, dass es nicht beliebig viele Züge geben kann, da jede Kante nur einmal (vom aktiven Stein) genutzt werden kann.

Um die NP-Schwere zu zeigen führt man eine Reduktion vom *gerichteten Hamilton-Kreis-Problem* durch. Hierbei konstruiert man aus einem gewöhnlichen Graphen $G = (V, E)$ einen Edge-Hop-Graphen $\mathcal{EH}$ in dem man jeden Knoten $v \in V$ durch ein *Knoten-Gadget* (Abbildung 5) ersetzt. Zusätzlich werden noch drei weitere Knoten benötigt (Startknoten a, Zielknoten z und ein Hilfsknoten b). Die Knoten-Gadgets werden so konstruiert, dass jedes genau einmal (auf einer Zugfolge von a nach b) durchquert werden muss, damit es eine gültige Zugfolge von b nach z gibt. Weiter wird der Graph so aufgebaut, dass jede Zugfolge von a nach z über b führt. Damit folgt, dass es nur dann eine Zugfolge von a nach z gibt, wenn es einen Hamilton-Kreis im (gerichteten) Graphen G gibt.

Das Knoten-Gadget selbst wird aus vier verschiedenen „Grund-Gadgets“ konstruiert, welche direkt als Edge-Hop-Graphen-Teilgraphen konstruiert werden. Diese vier Gadgets sind: Das *Dioden-Gadget* (Abbildung 3) , dessen Aufgabe es ist, den aktiven Stein nur in einer vorgegebenen Richtung passieren zu lassen. Das *Verzweigungs-Gadget* (Abbildung 1), welches dem aktiven Stein die Möglichkeit gibt, einen von zwei verschiedenen Pfaden zu wählen. Das

Zusammenführ-Gadget (Abbildung 2), welches zwei verschiedene Pfade zu einem einzelnen zusammenführt. Und das *Entsperr-Gadget* (Abbildung 4), welches der aktive Stein zuerst über einen „entsperrenden Eingang“ betreten muss, damit er danach das Gadget durchqueren kann.

Zusätzlich zu den vier Grund-Gadgets müssen noch kleine Gadgets konstruiert werden, um bspw. die Kanten zu modellieren. Ebenfalls muss ein *Kreuzungs-Gadget* konstruiert werden, da Edge-Hop-Graphen nicht planar sind.

3. Fazit

Edge Hop ist ein Modell um NP-Schwere zu zeigen. Hierzu muss man nur die entsprechenden Gadgets (Dioden-, Verzweigungs-, Zusammenführ-, Entsperr-, Kanten-, Kreuzungs-Gadget) als Instanzen des zu untersuchenden Problems konstruieren und hat damit automatisch die NP-Schwere gezeigt.

Eine offene Frage bzgl. Edge Hop ist, ob man das Kreuzungs-Gadget umgehen kann. Also ob es entweder möglich ist, die Edge-Hop-Graphen planar zu machen oder ob es möglich ist, aus den vorhandenen Gadgets Kreuzungs-Gadgets zu konstruieren. Weiter stellt sich die Frage, wie sich die Komplexität von EDGEHOP verhält, wenn man die Regeln modifiziert. Hier könnte man bspw. untersuchen, ob man $PSPACE$-Schwere zeigen kann, wenn man erlaubt, dass Kanten beliebig oft genutzt werden dürfen.

Literatur

[1] R. A. HEARN, E. D. DEMAINE, PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *TCS* **343** (2005) 1-2, 72–96.

Anhang

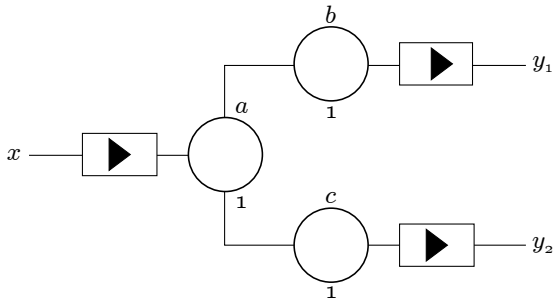


Abbildung 1: Ein Verzweigungs-Gadget konstruiert als EDGEHOP-Teilgraph. Der aktive Stein kann das Gadget über einen von zwei möglichen Ausgängen verlassen. Die Rechtecke mit dem Dreiecks-Symbol kennzeichnen Dioden-Gadgets.

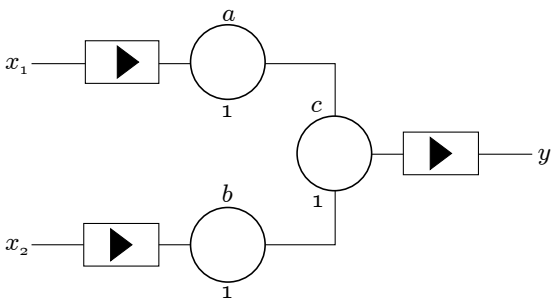


Abbildung 2: Ein Zusammenführungs-Gadget konstruiert als EDGEHOP-Teilgraph. Der aktive Stein kann das Gadget nur über y verlassen, unabhängig davon, über welchen Eingangs-Knoten er das Gadget betritt. Die Rechtecke mit dem Dreiecks-Symbol kennzeichnen Dioden-Gadgets.

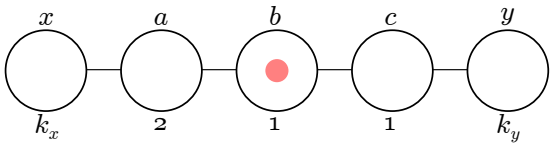


Abbildung 3: Das Dioden-Gadget als EDGE-HOP-Teilgraph. Dem aktiven Stein ist es nur möglich, dieses Gadget in einer bestimmten Richtung zu passieren.

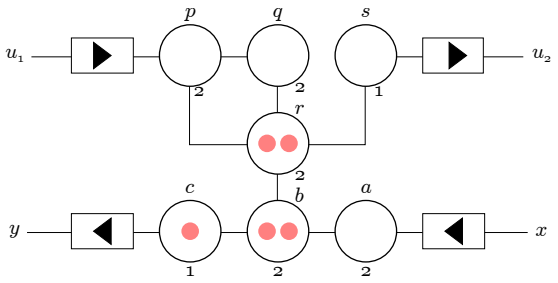


Abbildung 4: Ein Entsperr-Gadget. Der aktive Stein kann nur von x nach y ziehen, wenn er zu einem früheren Zeitpunkt das Gadget über u_1 betreten hat. Die Rechtecke mit dem Dreiecks-Symbol kennzeichnen Dioden-Gadgets.

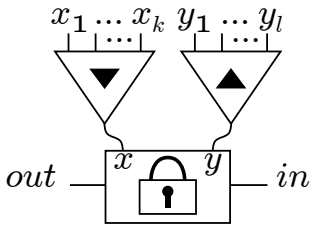


Abbildung 5: Ein Knoten-Gadget. Damit der aktive Stein im Gadget von in nach out ziehen kann, muss er es zuvor über einen der Knoten $x_1,...,x_k$ betreten haben. Die Knoten $x_1,...,x_k$ entsprechen den eingehenden und die Knoten $y_1,...,y_l$ den ausgehenden Kanten des korrespondieren Knotens. Das Rechteck mit dem Schloss-Symbol kennzeichnet ein Entsperr-Gadget und die Dreicke mit den Dreiecks-Symbolen kennzeichnen Zusammenführ- und Verzweigungs-Gadgets.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 14.

Exploration von Temporalen Graphen

F. Kammer[(A)]

[(A)]Universität Augsburg, Lehrstuhl für Theoretische Informatik
kammer@informatik.uni-augsburg.de

Zusammenfassung

Ein temporaler Graph ist ein Graph, dessen Kantenmenge sich in jedem Zeitschritt verändern kann. Eine Exploration in einem Graphen G ist ein Pfad, der alle Knoten von G mindestens einmal enthält. Die Suche nach einer Exploration in einem temporalen Graphen ist die Suche nach einem Zeitplan, der für jeden Zeitschritt vorschreibt, ob man am aktuellen Knoten verbleibt oder ihn über eine momentan vorhandene Kante verläßt. TEXP ist das Problem, einen solchen Zeitplan zu finden, der möglichst wenige Zeitschritte hat. Der Vortrag zeigt, dass das Finden einer $O(n^{1-\varepsilon})$-approximativen Lösung für TEXP NP-hart ist, und skizziert Ideen für spezielle temporale Graphen.
Die Ergebnisse sind gemeinsame Arbeit mit T. Erlebach und M. Hoffmann.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 15–18.

Clustering with Size Constraints

Faisal Abu-Khzam[(A)] Cristina Bazgan[(B,C)] Katrin Casel[(D)]
Henning Fernau[(D)]

[(A)]Lebanese American University, Beirut, Lebanon
faisal.abukhzam@lau.edu.lb

[(B)]PSL, Université Paris-Dauphine, LAMSADE UMR CNRS 7243, France
bazgan@lamsade.dauphine.fr

[(C)]Institut Universitaire de France

[(D)]Trier University, Fachbereich IV – Abteilung Informatikwissenschaften,
D-54286 Trier, Germany, {casel,fernau}@uni-trier.de

Abstract

Some clustering tasks do not fix the number of resulting clusters but rather ask for a partitioning which produces subsets of fixed minimum cardinality k. We consider this type of clustering with different measures for optimality, discuss how solutions for $k = 2$ can be computed and used to construct approximations for larger values of k. The comparison of different measures under a unified model reveals useful similarities.

1. Introduction

Classical, well-studied clustering-problems like k-MEANS or k-MEDIAN search for an "optimal" partition of n elements of a metric space into exactly k "small" subsets. There are several options to define the size of a cluster, for example radius or diameter, and different ways to define the optimality of the whole partition.

We discuss different clustering-problems which consider partitioning without a fixed number of subsets but with a lower bound on the number of elements in each subset. These types of clustering occur in different areas, for example as a version of the facility location problem where each facility is profitable because it has at least k customers, or as generalisation methods (clustering of personal data for privacy-reasons) with different measures for information-loss. We will use the following fixed abstract graph model for all these problem versions which will allow for an easier comparison:

k-CLUSTER
Input: Graph $G = (V, E)$ with edge-weight function $w_E : E \to \mathbb{R}_+$, $k \in \mathbb{N}$.
Output: A partitioning $P_1, \ldots, P_s$ of V, for some $s \in \mathbb{N}$, with $|P_i| \geq k$ for all $i = 1, \ldots, s$.

Of cause, one does not search for just any solution for k-CLUSTER but for a partitioning which preferably only combines objects with similar properties. The distance $d\colon V \times V \to \mathbb{R}_+$ induced by the shortest path with respect to the given edge-weight function w_E, reflects the dissimilarity of two vertices in the sense that objects with small distance are considered similar.

The overall cost of a partitioning $P_1, \ldots, P_s$ is always in some sense proportional to the dissimilarities within each cluster and is hence calculated in two steps: First by computing the local cost of each cluster P_i and second by combining all this individual information. We will discuss three different measures for the local cost caused by a cluster P_i:

- Radius, formally defined by $\text{rad}(P_i) := \min_{x \in P_i} \max_{y \in P_i} d(x,y)$.
- Diameter: $\text{diam}(P_i) := \max_{x \in P_i} \max_{y \in P_i} d(x,y)$.
- Average distortion: $\text{avg}(P_i) := |P_i|^{-1} \cdot \min_{x \in P_i} \sum_{y \in P_i} d(x,y)$.

These measures are naturally related as follows:

$$\text{avg}(P_i) \leq \text{rad}(P_i) \leq \text{diam}(P_i) \leq 2\text{rad}(P_i) \leq \tfrac{|P_i|}{2}\text{avg}(P_i) \tag{1}$$

The overall cost of a k-cluster $P_1, \ldots, P_s$ is then given by a certain combination of the local cost $f(P_1), \ldots, f(P_s)$ with $f \in \{\text{rad}, \text{diam}, \text{avg}\}$. In order to model the most common problem-versions we consider the following three possibilities to do so:

- The maximum individual cost: $\max_{i=1,\ldots,s} f(P_i)$. Because of its structure with respect to the values $f(P_1), \ldots, f(P_s)$, this overall-measure will be denoted with $\|\cdot\|_\infty$.
- The maximum individual cost, weighted by the number of vertices: $\max_{i=1,\ldots,s} |P_i| f(P_i)$, denoted by $\|\cdot\|_\infty^w$.
- The sum of individual cost, weighted by the number of vertices: $\sum_{i=1}^{s} |P_i| f(P_i)$, denoted by $\|\cdot\|_1^w$.

Any combination of $f \in \{\text{rad}, \text{diam}, \text{avg}\}$ with $\|\cdot\|_1^w$ or $\|\cdot\|_\infty^w$ or $\|\cdot\|_\infty$ yields a different problem. For a fixed $k \in \mathbb{N}$, the general optimization-problem is given by:

$(\|\cdot\|, f)$-k-CLUSTER
Input: Graph $G = (V, E)$ with edge-weight function $w_E : E \to \mathbb{R}_+$, $k \in \mathbb{N}$.
Output: k-clustering $P_1, \ldots, P_s$ of V for some $s \in \mathbb{N}$, which minimizes $\|(f(P_1), \ldots, f(P_s))\|$.

With these definitions, $(\|\cdot\|_\infty^w, \text{avg})$-$k$-CLUSTER models LOAD BALANCED FACILITY LOCATION without facility costs [6], $(\|\cdot\|_\infty, \text{rad})$-$k$-CLUSTER with the Euclidean distance is the so-called r-GATHER problem [1] and $(\|\cdot\|_1^w, \text{diam})$-$k$-CLUSTER is also known as k-MEMBER CLUSTERING [3].

The diverse behaviour for different choices of f and $\|\cdot\|$ is already evident in the cluster-cardinalities of optimal solutions. With the above definition of minimizing radius or average distortion, clusters in an optimal solution can be of arbitrarily large cardinality: for a star-graph, one cluster containing all vertices is *the* optimal solution with $\|\cdot\|_\infty$ or $\|\cdot\|_1^w$. For the diameter-measure, clusters can always be picked to have a cardinality between k and $2k-1$. Still, in the following we will see that there are very useful similarities between the different measure-choices.

2. *k=2*

At first glance, 2-CLUSTER looks very similar to the classical matching problem [5]. For some choices of f and $\|\cdot\|$, this connection is indeed helpful:

- $(\|\cdot\|_1^w, \text{avg})$-2-CLUSTER searches for a 2-cluster $P_1, \ldots, P_s$ for a graph $G = (V, E)$ which minimizes $\sum_{i=1}^{s} \min\{\sum_{w \in P_i} d(v, w) \colon v \in P_i\}$. This is equivalent to finding a minimum weight edge cover for the complete graph over the vertices V with edge-weights d which can be computed in polynomial time [8].

- $(\|\cdot\|_\infty, \text{rad})$-2-CLUSTER can be solved even simpler with unweighted edge-cover, by again creating the complete graph with induced weights and then deleting all edges of weight larger than $\max_{v \in V} \min_{w \in V} d(v, w)$.

- The diameter-measure is a bit more complicated but $(\|\cdot\|_1^w, \text{diam})$-2-CLUSTER can be modelled as an instance of SIMPLEX MATCHING, a more general version of matching which can be solved in polynomial time with an augmenting path strategy [2].

- $(\|\cdot\|_\infty, \text{diam})$-2-CLUSTER and $(\|\cdot\|_\infty^w, \text{diam})$-2-CLUSTER can be modelled as an instance of SIMPLEX COVER with binary search on the possible minimal values, which are among the set of all pairwise distances $d(u, v)$ with $u, v \in V$.

However, not all versions of 2-CLUSTER have these nice properties that relate to some sort of matching, in fact, $(\|\cdot\|_1^w, \text{rad})$-2-CLUSTER, $(\|\cdot\|_\infty^w, \text{rad})$-2-CLUSTER, $(\|\cdot\|_\infty^w, \text{avg})$-2-CLUSTER and $(\|\cdot\|_\infty, \text{avg})$-2-CLUSTER are already NP-hard. This hardness partially lies in the difficulty of choosing optimal cluster-center.

3. $k \geq 3$

If k is larger than two, all versions of k-cluster considered here are NP-hard. The reductions used to prove this hardness is very similar for all combinations of f and $\|\cdot\|$ and also show that for $k \geq 3$ there is no polynomial time $(2-\varepsilon)$-approximation algorithm for $(\|\cdot\|_\infty, \text{rad})$-$k$-CLUSTER or $(\|\cdot\|_\infty, \text{diam})$-$k$-CLUSTER unless $P = NP$.

Similarities between the different measure-choices provide ways to develop approximation-algorithms. Equation 1 yields a general guide to transfer results from one local cost to another. An α-approximation for the radius measure and some norm, for example, is also a 2α-approximation for the diameter measure with the same norm. For algorithms which only produce clusters of a certain maximum cardinality, like the approximations in [4], the relation between average distortion and radius or diameter can also be used to transfer results. The solutions for $k = 2$ from the previous section and other known results provide more ways to compute approximate solutions:

- In [1], the problem $(\|\cdot\|_\infty, \text{rad})$-$k$-CLUSTER is discussed under the name r-GATHER, where r takes the role of k. The 2-approximation presented there can be altered to yield a 2-approximation for $(\|\cdot\|_\infty, \text{diam})$-$k$-CLUSTER.

- The SIMPLEX MATCHING approach for $(\|\cdot\|_1^w, \text{diam})$-2-CLUSTER and the EDGE COVER approach for $(\|\cdot\|_1, \text{avg})$-2-CLUSTER can be nested in a particular way to compute a $\frac{35}{6}$-approximation for $(\|\cdot\|_1^w, \text{diam})$-4-CLUSTER. This combination of the two different approaches for the different measures exploits the properties of both problem-versions and can hopefully be extended for larger values of k.

- MICROAGGREGATION is essentially $(\|\cdot\|_1^w, \text{avg})$-$k$-CLUSTER where d is defined as the Euclidean distances between the given input-vertices with the difference, that the radius is not measured with respect to some existing vertex but geometrically to the centroid. Some adjustment to the $8(k-1)$-approximation algorithm for from [7] can be made to compute a $(k-1)$-approximation for $(\|\cdot\|_1^w, \text{avg})$-$k$-CLUSTER.

References

[1] G. AGGARWAL, S. KHULLER, T. FEDER, Achieving Anonymity via Clustering. In: V. VIANU, G. GOTTLOB (eds.), *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'00*. ACM, 2006, 153–162.

[2] E. ANSHELEVICH, A. KARAGIOZOVA, Terminal Backup, 3D Matching, and Covering Cubic Graphs. *SIAM J. Comput.* **40** (2011) 3, 678–708.

[3] J.-W. BYUN, A. KAMRA, E. BERTINO, N. LI, Efficient k-Anonymization Using Clustering Techniques. In: R. KOTAGIRI, P. R. KRISHNA, M. MOHANIA, E. NANTAJEEWARAWAT (eds.), *Advances in Databases: Concepts, Systems and Applications*. LNCS 4443, Springer, 2007, 188–200.

[4] J. DOMINGO-FERRER, F. SEBÉ, A. SOLANAS, A polynomial-time approximation to optimal multivariate microaggregation. *Computers & Mathematics with Applications* **55** (2008) 4, 714–732.

[5] J. EDMONDS, E. L. JOHNSON, Matching, Euler tours and the Chinese postman. *Mathematical Programming* **5** (1973), 88–124.

[6] S. GUHA, A. MEYERSON, K. MUNAGALA, Hierarchical Placement and Network Design Problems. In: *In Proceedings of the 41th Annual IEEE Symposium on Foundations of Computer Science, FOCS'00*. IEEE Computer Society, 2000, 603–612.

[7] M. LASZLO, S. MUKHERJEE, Approximation Bounds for Minimum Information Loss Microaggregation. *IEEE Trans. Knowl. Data Eng.* **21** (2009) 11, 1643–1647.

[8] A. SCHRIJVER, *Combinatorial Optimization*. Springer, 2003.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 19–21.

On Finding Points in General Position

Vincent Froese[(A)] Iyad Kanj[(B)] André Nichterlein[(A)]
Rolf Niedermeier[(A)]

[(A)]Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
{vincent.froese,andre.nichterlein,rolf.niedermeier}@tu-berlin.de

[(B)]School of Computing, DePaul University, Chicago, USA
ikanj@cdm.depaul.edu

Abstract

We study computational aspects of the General Position Subset Selection problem defined as follows: Given a set of points in the plane, find a maximum-cardinality subset of points in general position. We prove that General Position Subset Selection is NP-hard, APX-hard, and give several fixed-parameter tractability results.

1. Introduction

For a set $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{Q}^2$ of n points in the plane, a subset $S \subseteq P$ is in *general position* if no three points in S are *collinear* (that is, lie on the same line). Whereas a frequent assumption for point set problems in computational geometry is that the given point set is in general position, surprisingly, the problem of choosing a maximum-cardinality subset of points in general position, from a given set of points, has received little attention from the computational complexity perspective, although not from the combinatorial geometry perspective. In particular, up to our knowledge, the classical complexity of the aforementioned problem is unresolved. Formally, the decision version of the problem is (see Figure 1 for an example):

GENERAL POSITION SUBSET SELECTION
Input: $P \subseteq \mathbb{Q}^2$ and $k \in \mathbb{N}$.
Question: Is there a subset $S \subseteq P$ in general position of cardinality at least k?

The special case of GENERAL POSITION SUBSET SELECTION, referred to as the no-three-in-line problem, which asks to place a maximum number of points in general position on an $n \times n$-grid, received considerable attention in discrete geometry. Since at most two points can be placed on any grid-line, the maximum number of points in general position that can be placed on an $n \times n$ grid is at most $2n$. Indeed, only for small n it is known that $2n$ points can always be placed on the $n \times n$ grid. Erdős [8] observed that, for sufficiently large n, one can place $(1-\epsilon)n$ points in general position on the $n \times n$ grid, for any $\epsilon > 0$. This lower bound was improved by Hall et al. [5] to $(\frac{3}{2}-\epsilon)n$. It was conjectured by Guy and Kelly [4] that, for sufficiently large n, one can place more than $\frac{\pi}{\sqrt{3}}n$ many points in general position on the $n \times n$

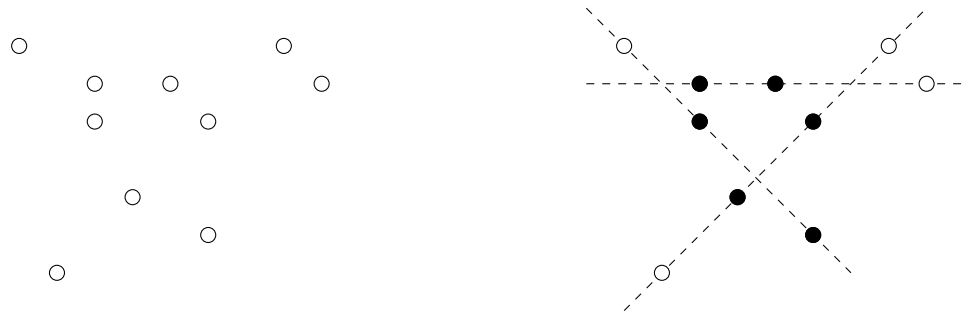


Figure 1: On the left side a set of ten points is displayed where a largest subset S of points in general position has cardinality six. This can be seen as follows: The ten points can be covered by three lines (see dashed lines on the right side). From each line, the set S can contain at most two points. Thus, S contains at most six points, for example the six highlighted points on the right side.

grid. This conjecture remains unresolved, hinting at the challenging combinatorial nature of the no-three-in-line problem, and hence of the GENERAL POSITION SUBSET SELECTION problem as well.

Observing that the computational complexity of the closely-related POINT LINE COVER problem (given a point set in the plane, find a minimum-cardinality set of lines, the size of which is called the *line cover number*, that cover all points) was intensively studied, we try to fill the gap by providing both computational hardness and fixed-parameter tractability results for the much less studied GENERAL POSITION SUBSET SELECTION. In doing so, we particularly consider the parameters solution size k (size of the sought subset in general position) and its dual $h := n - k$, and investigate their impact on the computational complexity of GENERAL POSITION SUBSET SELECTION.

Related Work A general account on applying methods from parameterized complexity analysis to problems from computational geometry is due to Giannopoulos et al. [3]. Payne and Wood [7] provide improved lower bounds on the size of a point set in general position, a question originally studied by Erdős [2]. In his master's thesis, Cao [1] gives a problem kernel of $O(k^4)$ points for GENERAL POSITION SUBSET SELECTION (there called NON-COLLINEAR PACKING problem) and a simple greedy $O(\sqrt{\text{opt}})$-factor approximation algorithm for the maximization version. He also presents an Integer Linear Program formulation for GENERAL POSITION SUBSET SELECTION, and shows that it is in fact the dual of an Integer Linear Program formulation for the POINT LINE COVER problem. As to results for the much better studied POINT LINE COVER, we refer to the work of Kratsch et al. [6], and to the work cited therein.

Our Contributions We show the NP-hardness and the APX-hardness of GENERAL POSITION SUBSET SELECTION. Our main algorithmic results, however, concern the power of polynomial-time data reduction for GENERAL POSITION SUBSET SELECTION: we give an $O(k^3)$-point kernel and an $O(h^2)$-point kernel, and show that the latter kernel is asymptotically optimal under a reasonable complexity-theoretic assumption. Table 1 summarizes our results.

The full version of this work can be found at arXiv, `http://arxiv.org/abs/1508.01097`.

Table 1: Overview of the results we obtain for GENERAL POSITION SUBSET SELECTION, where n is the number of input points, k is the parameter size of the sought subset in general position, $h = n - k$ is the dual parameter, and ℓ is the line cover number.

	Result
Hardness	NP-hard and APX-hard no $2^{o(n)} \cdot n^{O(1)}$-time algorithm (unless the Exponential Time Hypothesis fails) no $O(h^{2-\epsilon})$-point kernel (unless coNP $\subseteq$ NP/poly)
Tractability	$(15k^3)$-point kernel (computable in $O(n^2 \log n)$ time) $O(n^2 \log n + 41^k \cdot k^{2k})$-time solvable $(120\ell^3)$-point kernel (computable in $O(n^2 \log n)$ time) $O(n^2 \log n + 41^{2\ell} \cdot \ell^{4\ell})$-time solvable $O(h^2)$-point kernel (computable in $O(n^3)$ time) $O(2.08^h + n^3)$-time solvable

References

[1] C. CAO, *Study on Two Optimization Problems: Line Cover and Maximum Genus Embedding*. Master's thesis, Texas A&M University, 2012.

[2] P. ERDŐS, On some metric and combinatorial geometric problems. *Discrete Mathematics* **60** (1986), 147–153.

[3] P. GIANNOPOULOS, C. KNAUER, S. WHITESIDES, Parameterized Complexity of Geometric Problems. *The Computer Journal* **51** (2008) 3, 372–384.

[4] R. K. GUY, P. A. KELLY, The no-three-in-line problem. *Canadian Mathematical Bulletin* **11** (1968), 527–531.

[5] R. HALL, T. JACKSON, A. SUDBERY, K. WILD, Some advances in the no-three-in-line problem. *Journal of Combinatorial Theory, Series A* **18** (1975) 3, 336 – 341.

[6] S. KRATSCH, G. PHILIP, S. RAY, Point Line Cover: The Easy Kernel is Essentially Tight. In: *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '14)*. 2014, 1596–1606. Available on arXiv:1307.2521.

[7] M. S. PAYNE, D. R. WOOD, On the General Position Subset Selection Problem. *SIAM Journal on Discrete Mathematics* **27** (2013) 4, 1727–1733.

[8] K. F. ROTH, On a problem of Heilbronn. *Journal of the London Mathematical Society* **1** (1951) 3, 198–204.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 22–25.

Pseudoarboricity or Lowest Maximum Indegree Orientations

Markus Blumenstock[(A)]

[(A)]Institut für Informatik, Johannes Gutenberg-Universität Mainz
mablumen@uni-mainz.de

Abstract

We show that the pseudoarboricity p of a graph can be computed in $\mathcal{O}(|E|^{3/2}\sqrt{\log\log p})$ steps. Our analysis uses a reduction to the lowest maximum indegree orientation problem and a combination of exact and approximation algorithms.

1. Introduction

1.1. Definitions and Preliminaries

Let $G = (V, E)$ be a simple graph. The average density of a subgraph $H = (V_H, E_H) \subseteq G$ is

$$d(H) = \frac{|E_H|}{|V_H|}.$$

An (induced) subgraph with maximum average density is called a densest subgraph of G. We will denote its density by $d^*(G)$. The value $\lceil d^* \rceil$ has interesting connections to other concepts in combinatorics, as Theorem 1.1 and Theorem 1.2 show.

Theorem 1.1 (Picard and Queyranne [9, Corollary 3-1]) *Let G be a simple graph. For the pseudoarboricity $P(G)$, we have $P(G) = \lceil d^*(G) \rceil$.*

Theorem 1.2 (Frank and Gyárfás [5, Theorem 1]) *Let G be a simple graph. Let $\vec{G}$ be an orientation of G such that the maximum indegree is lowest. Then this indegree equals $\lceil d^*(G) \rceil$.*

1.2. Related Work

For brevity, we do not review all algorithms for computation of d^*. The currently fastest known algorithm is due to Goldberg [7] when used with the Goldberg-Rao maximum flow algorithm.

Determining the value $\lceil d^* \rceil$, i.e. pseudoarboricity, is a special case of the covering problem for matroid sums. Gabow and Westermann [6] propose an algorithm specifically for pseudoarboricity, which runs in $\mathcal{O}(|E| \min(\sqrt{|E| \log |V|}, (|V| \log |V|)^{2/3}))$, being the currently fastest known algorithm. Algorithms that use the reduction to the lowest maximum indegree orientation problem and maximum flow algorithms have runtime $\mathcal{O}(|E|^{3/2} \log d^*)$ [1, 2]. Kowalik [8] proposes an approximation scheme for $\lceil d^* \rceil$. Several authors independently describe a greedy 2-approximation algorithm which runs in $\mathcal{O}(|E|)$, e.g. Bezáková [2] and Aichholzer et al. [1].

Table 1: New runtime bounds for the pseudoarboricity problem, depending on d^*. Here, $\log^*$ denotes the iterated logarithm in base 2.

Bound for d^*	Runtime	No.
–	$\mathcal{O}\left(\lvert E\rvert^{3/2}\sqrt{\log\log d^*}\right)$	(I)
$\mathcal{O}\left(\frac{\sqrt{\lvert E\rvert}}{\log\lvert V\rvert}\right)$	$\mathcal{O}\left(\lvert E\rvert^{3/2}\log^* d^*\right)$	(II)
$\mathcal{O}\left(\frac{\sqrt{\lvert E\rvert}}{\log^2\lvert V\rvert}\right)$	$\mathcal{O}\left(\lvert E\rvert^{3/2}\right)$	(III)

1.3. Contributions

We present a selection of improved runtime bounds for the pseudoarboricity problem.

Theorem 1.3 *Pseudoarboricity and a corresponding partition into pseudoforests can be computed in the runtime bounds stated in Table 1.*

2. Preparations

An exact algorithm to determine $\lceil d^*\rceil$ uses Theorem 1.2 and a binary search on an interval of integers that contains $\lceil d^*\rceil$. For some tentative integer d, we try to re-orient a given orientation $\vec{G}$ such that every vertex has at most d ingoing edges (this is called a d-orientation). To do so, we add a source with arcs to every vertex which has more than d ingoing edges: this vertex has to flip at least $\text{indeg}(v)-d$ ingoing edges (more, if outgoing edges flip as well) to reach level d. Likewise, we add a sink with arcs from every vertex with $\text{indeg}(v)<d$ to it. These vertices may rise up to level d by flipping edges. We call this the 're-orientation algorithm'.

Formally, we introduce source and sink nodes $s,t\notin V$, and define the set of arcs A on $V\cup\{s,t\}$ as follows:

$$\begin{aligned}(s,v)\in A \text{ with } c(s,v)=\text{indeg}_{\vec{G}}(v)-d &\iff \text{indeg}_{\vec{G}}(v)>d,\\ (u,v)\in A \text{ with } c(u,v)=1 &\iff u\leftarrow v \text{ in } \vec{G},\\ (v,t)\in A \text{ with } c(v,t)=d-\text{indeg}_{\vec{G}}(v) &\iff \text{indeg}_{\vec{G}}(v)<d.\end{aligned}$$

This network has unit capacities, except for the arcs from the source and the arcs to the sink. Even and Tarjan [4] analyze Dinitz's algorithm on unit capacity networks. We generalize their results to 'almost unit capacity networks'.

Theorem 2.1 *In a flow network where all capacities are integers, and all arcs except the source and sink arcs have capacity* 1*, a maximum flow can be found in* $\mathcal{O}\left(\lvert E\rvert\min\left(\sqrt{\lvert E\rvert},\lvert V\rvert^{2/3}\right)\right)$.

If an approximation of $\lceil d^*\rceil$ is given, we can speed up the binary search.

Lemma 2.2 *Let I be an interval of integers, and let $x \in I$ be the minimum value for which a certain property holds. Let the property also hold for all $x' \in I$ with $x' > x$. Given $d \in I$ with $x \leq d \leq \lceil (1+\delta)x \rceil$ for some $\delta > 0$, a binary search for x can be realized with $\mathcal{O}(\log(\delta x))$ tests.*

We will utilize the upper bound $d^* \in \mathcal{O}(\sqrt{|E|})$ in the proof of Theorem 1.3. This asymptotic bound is immediate from the pseudoarboricity bound given by Gabow and Westermann [6] and Theorem 1.1. However, a tight bound for d^* can be proved with simple counting arguments.

Proposition 2.3 *For a simple graph $G = (V, E)$, we have $d^*(G) \leq \frac{1}{4}\left(\sqrt{8|E|+1} - 1\right)$.*

3. New Complexity Bounds

We can apply Theorem 2.1, Lemma 2.2 and the linear-time 2-approximation algorithm for a new runtime bound of the re-orientation algorithm.

Corollary 3.1 *To compute $\lceil d^* \rceil$, the re-orientation algorithm can be made to run in*

$$\mathcal{O}\left(|E| \min\left(\sqrt{|E|}, |V|^{2/3}\right) \log d^*\right).$$

We will combine the exact algorithm with Kowalik's approximation scheme. The following lemma can be used to turn the re-orientation algorithm into an approximation scheme for $\lceil d^* \rceil$. We note that it can be generalized to approximate d^* directly by using a relaxation of the orientation problem. For this relaxation, see Cohen [3].

Lemma 3.2 *([8, Lemma 2]) Let $\vec{G}$ be a d-orientation of a graph G for $d \in \mathbb{N}$, and let $d > d^*$. Then for every vertex $v \in V$, the distance in $\vec{G}$ to a vertex with indegree smaller than d does not exceed* $\log_{d/d^*} |V|$.

Kowalik uses Lemma 3.2 to stop a test with Dinitz's algorithm after at most $2 + \log_{1+\epsilon} |V|$ blocking flow phases (which each run in $\mathcal{O}(|E|)$) to obtain an ϵ-approximation scheme for $\lceil d^* \rceil$. Finding an integer d and a d-orientation such that $\lceil d^* \rceil \leq d \leq \lceil (1+\epsilon) d^* \rceil$ is thus possible in $\mathcal{O}(|E| \log(|V|)/\epsilon \log d^*)$ by a Taylor expansion of $\ln(1+\epsilon)$ for $\epsilon > 0$.

We can now prove Theorem 1.3. Note that more runtime bounds can be derived for certain bounds on d^* by considering the $|V|^{2/3}$-analysis branch of Dinitz's algorithm.

Proof. (Theorem 1.3) For all claims in Table 1, we initially compute a 2-approximation of $\lceil d^* \rceil$ in linear time in order to estimate d^* and set ϵ up to constant factors, which does not affect the runtime analysis asymptotically. Once $\lceil d^* \rceil$ has been determined, a partition into $\lceil d^* \rceil$ pseudoforests can be found in $\mathcal{O}(|E|^{3/2})$ [6, Theorem 3.2].

Consider claim (I) in Table 1. Check whether $d^* \leq |V|^{0.49}$ (any fixed exponent less than 0.5 would do). If yes, see the proof of claim (III).

Otherwise, we have $\log d^* > 0.49 \log |V|$. We set $\epsilon \simeq \log(|V|)^2/d^*$ and thus the approximation scheme runs in $\mathcal{O}(|E|d^*)$. By Proposition 2.3, this is always in $\mathcal{O}(|E|^{3/2})$.

A binary search on the narrowed search interval now needs $\mathcal{O}(\log(\epsilon d^*)) = \mathcal{O}(\log \log d^*)$ tests by Lemma 2.2. The Gabow-Westermann algorithm can perform this second phase in $\mathcal{O}(|E|^{3/2}\sqrt{\log \log d^*}))$ by setting the parameter for the balanced binary search used in it appropriately (see [6, Section 4]). The claim follows.

We only outline the proof of claim (II): We run the approximation scheme in $i = 1, ..., \log^* d^* - 1$ phases, each time on the search interval left after the previous phase, with parameters

$$\epsilon_1 \simeq \frac{\log d^*}{d^*}, \epsilon_2 \simeq \frac{\log\log d^*}{d^*}, \epsilon_3 \simeq \frac{\log\log\log d^*}{d^*}, ...,$$

followed by the re-orientation algorithm on a constant-size interval.

For claim (III), we set $\epsilon \simeq 1/d^*$. The approximation scheme runs in $\mathcal{O}(|E|^{3/2})$. The second phase is a single re-orientation test with Dinitz's algorithm, which takes $\mathcal{O}(|E|^{3/2})$. □

References

[1] O. AICHHOLZER, F. AURENHAMMER, G. ROTE, *Optimal Graph Orientation with Storage Applications*. SFB-Report F003-51, SFB 'Optimierung und Kontrolle', TU Graz, Austria, 1995.
www.igi.tugraz.at/auren/psfiles/aar-ogosa-95.ps.gz

[2] I. BEZÁKOVÁ, *Compact Representations of Graphs and Adjacency Testing*. Master's thesis, Comenius University, Bratislava, Slovakia, 2000.
http://www.cs.rit.edu/~ib/Papers/BezakovaMagisterThesis00.pdf

[3] N. COHEN, Several Graph Problems and their LP formulation (explanatory supplement for the Sage graph library). 2010.
http://hal.inria.fr/inria-00504914

[4] S. EVEN, R. E. TARJAN, Network Flow and Testing Graph Connectivity. *SIAM Journal on Computing* **4** (1975) 4, 507–518.
http://dx.doi.odoi.org/10.1137/02rg/10.1137/0204043

[5] A. FRANK, A. GYÁRFÁS, How to orient the edges of a graph? In: A. HAJNAL, V. T. SÓS (eds.), *COMBINATORICS: 5th Hungarian Colloquium, Keszthely, June/July 1976, Proceedings*. Number 2 in Colloquia Mathematica Societatis János Bolyai, North Holland Publishing Company, 1978, 353–364.
http://www.renyi.hu/~gyarfas/Cikkek/09_FrankGyarfas_HowToOrientTheEdgesOfAGraph.pdf

[6] H. N. GABOW, H. H. WESTERMANN, Forests, Frames, and Games: Algorithms for Matroid Sums and Applications. *Algorithmica* **7** (1992) 1-6, 465–497.
http://dx.doi.org/10.1007/BF01758774

[7] A. V. GOLDBERG, *Finding a Maximum Density Subgraph*. Technical report, Berkeley, CA, USA, 1984.
http://www.eecs.berkeley.edu/Pubs/TechRpts/1984/CSD-84-171.pdf

[8] L. KOWALIK, Approximation Scheme for Lowest Outdegree Orientation and Graph Density Measures. In: T. ASANO (ed.), *Algorithms and Computation*. Lecture Notes in Computer Science 4288, Springer Berlin Heidelberg, 2006, 557–566.
http://dx.doi.org/10.1007/11940128_56

[9] J.-C. PICARD, M. QUEYRANNE, A Network Flow Solution to Some Nonlinear 0-1 Programming Problems, with Applications to Graph Theory. *Networks* **12** (1982) 2, 141–159.
http://dx.doi.org/10.1002/net.3230120206

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 26–29.

A New Parameterisation for Consensus String Problems

Markus L. Schmid[(A)]

[(A)]Trier University, Fachbereich IV – Abteilung Informatikwissenschaften,
D-54286 Trier, Germany, MSchmid@uni-trier.de

Abstract

The parameterised complexity of the CLOSEST SUBSTRING and CONSENSUS PATTERNS PROBLEM w. r. t. parameter $(\ell - m)$ is investigated, where ℓ is the maximum length of the input strings and m is the length of the solution string. We present an exact exponential time algorithm for both problems, which is based on an alphabet reduction. Furthermore, it is shown that for most combinations of $(\ell - m)$ and one of the classical parameters (m, ℓ, number of input strings k, distance d), we obtain fixed-parameter tractability, but even for constant $(\ell - m)$ and constant alphabet size, both problems are NP-hard.

1. Introduction

We consider the following consensus string problem:

CLOSEST SUBSTRING (CLOSESUBSTR)

Instance: Strings $s_1, s_2, \ldots, s_k$ over some alphabet Σ with $|s_i| \leq \ell$, $1 \leq i \leq k$, for some $\ell \in \mathbb{N}$, and numbers $m, d \in \mathbb{N}$.
Question: Is there a string s with $|s| = m$ such that, for every i, $1 \leq i \leq k$, s_i has a length-m substring s'_i with $\mathsf{d_H}(s, s'_i) \leq d$?[1]

If we instead require $\sum_{i=1}^{k} \mathsf{d_H}(s, s'_i) \leq d$, then it is called the CONSENSUS PATTERNS PROBLEM (CONSPAT). Both CLOSESUBSTR and CONSPAT are classical NP-hard string problems and they have been intensely studied in the multivariate setting (see [2, 3, 5, 7] and, for a survey, [1]).

The most commonly considered parameters are k, m, d and $|\Sigma|$. The existing results show that CLOSESUBSTR and CONSPAT are fixed-parameter intractable, even for strong parameterisations (see Tables 1 and 2). This paper contributes to the multivariate analysis of these problems by investigating the parameter $(\ell - m)$, i. e., the length difference between the input strings and the solution string, the relevance of which is pointed out by the following considerations.

Another NP-hard consensus string problem, that exhibits a much better parameterised complexity, is the CLOSEST STRING PROBLEM (CLOSESTR), which is defined as CLOSESUBSTR, but with the additional restriction $|s_1| = |s_2| = \ldots = |s_k| = m$ (or, equivavalently, $(\ell - m) = 0$).

[1]Here, $\mathsf{d_H}$ denotes the Hamming distance.

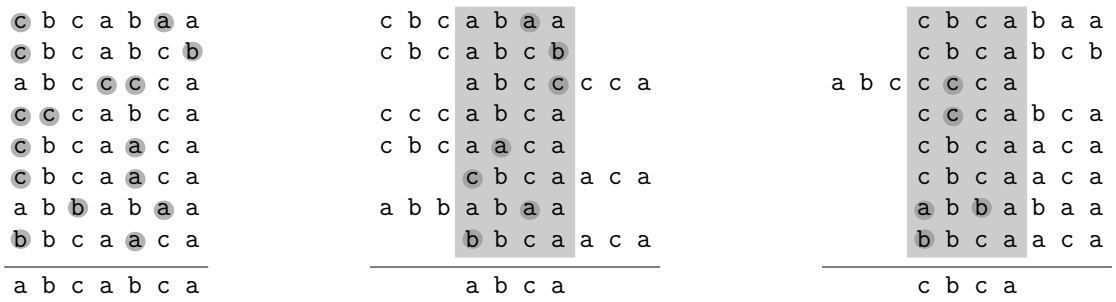


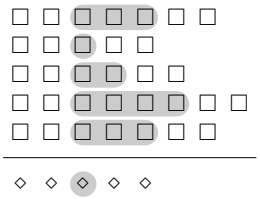
Figure 1: CLOSESTR, CLOSESUBSTR and CONSPAT instance.

Hence, bounding $(\ell - m)$ by 0 makes CLOSESUBSTR much easier. We can show that the fixed-parameter tractability of CLOSESTR w. r. t. k, d and m carries over to CLOSESUBSTR as long as we take $(\ell - m)$ as a second parameter. With respect to CONSPAT, the parameter $(\ell - m)$ plays a quite different role, since CONSPAT becomes trivial if $(\ell - m) = 0$. So the question arises, whether bounding $(\ell - m)$ by a constant $c \geq 1$ yields a polynomial-time solvable variant of CONSPAT or whether it is fixed-parameter tractable w. r. t. $(\ell - m)$. In this regard, we can show a strong negative result, namely that CONSPAT remains NP-hard, even if $(\ell - m) \leq 6$ and $|\Sigma| \leq 5$. Similar to CLOSESUBSTR, some fixed-parameter tractable variants can be obtained by parameterising by $(\ell - m)$ and additional parameters. For proving the hardness of CONSPAT with $(\ell - m) \leq 6$ and $|\Sigma| \leq 5$, we apply an alphabet reduction technique, which also yields an exact exponential-time algorithm for both problems CLOSESUBSTR and CONSPAT.

As an illustration, at the left of Figure 1, a CLOSESTR instance with 8 length-7 input strings is represented as an 8×7 matrix. For $d = 2$, $s = \texttt{abcabca}$ is a possible solution string, since each input string has at most two mismatches with s (which are highlighted in grey). If we interpret the same strings as a CLOSESUBSTR instance with $m = 4$, then we first have to identify a length-4 substring in each of the input strings, i. e., we have to align them such that an 8×4 matrix is obtained (the grey rectangle in the middle of Figure 1). Once such an alignment is fixed, we have to solve CLOSESTR for these aligned substrings. If we consider the case $d = 1$, then `abca` is a solution string (as illustrated at the right of Figure 1). However, the total sum of mismatches is 7, which means that this is not a solution if we interpret the input strings as an instance of CONSPAT with $d = 5$.

2. Alphabet Reduction and Exact Exponential Algorithm

Let $s_1, \ldots, s_k$, $m, d \in \mathbb{N}$ be an instance for CLOSESUBSTR or CONSPAT. Furthermore, let s be a hypothetical solution string. No matter how we choose our alignment, the j^{th} position of s can only be aligned with positions $j, j+1, \ldots, |s_i| - m + j$ of s_i. This is illustrated at the left of Figure 2 for some example instance and the third symbol of s (with $m = 5$). Hence, every position of s corresponds to an *alignment region* (the grey area at the left of Figure 2) of the input strings. The mismatches caused by a position of s only depend on the substrings of its alignment region. Thus, renaming the input strings, such that the structure of each alignment region is preserved, yields an equivalent instance. For example, if $m = 3$, then (as illustrated at the right of Figure 2) the instance over an 8-letter alphabet can be renamed into the equivalent



A B C A C
D B E F
G H A B C F

⇒

1 2 3 1 3
4 2 4 5
5 6 1 2 3 5

Figure 2: Illustrations of the renaming procedure.

k	m	d	$\|\Sigma\|$	ℓ	Result	Reference
–	–	–	–	**p**	FPT	[2]
p	–	–	2	–	W[1]-hard	[3]
p	**p**	**p**	–	–	W[1]-hard	[3]
–	**p**	–	**p**	–	FPT	Trivial
p	–	**p**	2	–	W[1]-hard	[7]

k	m	d	$\|\Sigma\|$	$(\ell-m)$	Results
–	–	–	2	0	NP-hard
p	–	–	–	**p**	FPT
–	**p**	–	–	**p**	FPT
–	–	**p**	–	**p**	FPT

Table 1: Old and new results about CLOSESUBSTR.

instance over a 6-letter alphabet. Formalising this yields the following result:

Theorem 2.1 *Problems* CLOSESUBSTR *and* CONSPAT *can be solved in* $\mathcal{O}^*((k(\ell-m+1))^m)$.

3. Fixed-Parameter Tractability

All results of this section are summarised in Tables 1 and 2.[2]

3.1. CLOSEST SUBSTRING

The parameterised complexity of CLOSESUBSTR is well understood w. r. t. parameters k, m, d, $|\Sigma|$ and ℓ (see left side of Table 1).

If we restrict the parameter $(\ell-m)$ in the strongest possible way, i. e., requiring $(\ell-m)=0$, then CLOSESUBSTR collapses to the problem CLOSESTR, which is NP-hard even if $|\Sigma|=2$ (see [5]); thus, CLOSESUBSTR is most likely fixed-parameter intractable w. r. t. $(\ell-m)$ and $|\Sigma|$. However, adding one of k, m or d to the parameter $(\ell-m)$ yields fixed-parameter tractability. For the parameters $(k,(\ell-m))$ and $(m,(\ell-m))$ this can be easily concluded from known results, whereas for the case where $(\ell-m)$ and d are parameters, an fpt-algorithm for CLOSESTR presented in [6] can be adapted to the problem CLOSESUBSTR.

3.2. CONSENSUS PATTERNS

CONSPAT shows a comparatively unfavourable fixed-parameter behaviour as CLOSESUBSTR (see left side of Table 2). The parameter ℓ is missing from the left side of Table 2 and, it seems as

[2] In all tables, **p** means that the label of this column is treated as a parameter and an integer entry means that the result holds even if this parameter is set to the given constant; problems that are hard for W[1] are not in FPT (under complexity theoretical assumptions, see [4]).

k	m	d	$\|\Sigma\|$	Results	Ref.
p	–	–	2	W[1]-hard	[3]
p	**p**	**p**	–	W[1]-hard	[3]
–	**p**	–	**p**	FPT	Trivial
–	–	**p**	**p**	FPT	[7]

k	m	d	$(\ell-m)$	$\|\Sigma\|$	ℓ	Results
–	**p**	–	**p**	–	**p**	Open
–	–	–	–	**p**	**p**	FPT
p	–	–	–	–	**p**	FPT
–	–	**p**	–	–	**p**	FPT
–	–	–	6	5	–	NP-hard
p	–	–	**p**	–	–	FPT
–	–	**p**	**p**	–	–	FPT

Table 2: Old and new results about CONSPAT.

though this parameter has been neglected in the multivariate analysis of the problem CONSPAT. Unfortunately, we are not able to answer the most important respective question, i. e., whether or not CONSPAT is fixed-parameter tractable w. r. t. ℓ. For all other combinations of parameters including ℓ, fixed-parameter tractability can be easily shown.

It can be easily seen that, unlike as for CLOSESUBSTR, the NP-hardness of CONSPAT is not preserved if $(\ell - m)$ is bounded by 0. By simple enumeration arguments, we can show that CONSPAT is in FPT w. r. t. the combined parameters $(k, (\ell - m))$ and $(d, (\ell - m))$. If CONSPAT is parameterised by $(\ell - m)$ and m, then we arrive again at the open case already mentioned above. Consequently, there are only two cases left open: the parameter $(\ell - m)$ and the combined parameter $((\ell - m), |\Sigma|)$. We answer the question whether for these cases we have fixed-parameter tractability in the negative:

Theorem 3.1 CONSPAT$((\ell - m) = 6, |\Sigma| = 5)$ *is* NP*-hard.*

References

[1] L. BULTEAU, F. HÜFFNER, C. KOMUSIEWICZ, R. NIEDERMEIER, Multivariate Algorithmics for NP-Hard String Problems. *EATCS Bulletin* **114** (2014), 31–73.

[2] P. A. EVANS, A. D. SMITH, H. T. WAREHAM, On the complexity of finding common approximate substrings. *Theoretical Computer Science* **306** (2003), 407–430.

[3] M. R. FELLOWS, J. GRAMM, R. NIEDERMEIER, On The Parameterized Intractability Of Motif Search Problems. *Combinatorica* **26** (2006), 141–167.

[4] J. FLUM, M. GROHE, *Parameterized Complexity Theory*. Springer-Verlag New York, Inc., 2006.

[5] M. FRANCES, A. LITMAN, On covering problems of codes. *Theory of Computing Systems* **30** (1997), 113–119.

[6] J. GRAMM, R. NIEDERMEIER, P. ROSSMANITH, Fixed-Parameter Algorithms for CLOSEST STRING and Related Problems. *Algorithmica* **37** (2003), 25–42.

[7] D. MARX, Closest Substring Problems with Small Distances. *SIAM Journal on Computing* **38** (2008), 1382–1410.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 30–33.

The Semantics of Annihilation Rules in Membrane Computing

Daniel Díaz-Pernil[(A)] Rudolf Freund[(B)]
Miguel A. Gutiérrez-Naranjo[(C)] Alberto Leporati[(D)]

[(A)]Research Group on Computational Topology and Applied Mathematics
Department of Applied Mathematics - University of Sevilla, Spain
sbdani@us.es

[(B)]Technische Universität Wien, Institut für Computersprachen, A-1040 Wien, Austria
rudi@emcc.at

[(C)]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, Spain
magutier@us.es

[(D)]Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca, Italy
alberto.leporati@unimib.it

Abstract

Polarizationless recognizer P systems with active membranes, without dissolution, with division of elementary and non-elementary membranes, with antimatter and matter/antimatter annihilation rules can solve all problems in **NP** when the annihilation rules have (weak) priority over all the other rules. Yet if the the matter/antimatter annihilation rules do not have priority, such recognizer P systems with matter/antimatter annihilation rules characterize exactly **P**. Hence, the semantics of applying the rules constitutes a frontier of tractability.

1. Introduction

The concept of antimatter was first introduced in the framework of membrane computing as a control tool for the flow of spikes in spiking neural P systems e.g., see [4]. In this context, when one spike and one anti-spike appear in the same neuron, the annihilation occurs and both, spike and anti-spike, disappear. The concept of antimatter and matter/antimatter annihilation rules later was adapted to other variants of P systems, and currently it is an active research area, e.g., see [1, 2, 3]. In [3], it has been shown that antimatter and matter/antimatter annihilation rules are a frontier of tractability – recognizer P systems from $\mathcal{AM}^0_{-d,+ne}$ with antimatter and matter/antimatter annihilation rules can solve all problems in **NP**, whereas without these rules we only get **P**.

2. Recognizer P Systems

For the basic elements of membrane computing we refer to the handbook, see [5], and for actual informations to the website [6]. The main *syntactic* ingredients of a cell–like P system are the *membrane structure*, the *multisets*, and the *evolution rules*. A *membrane structure* consists of several membranes arranged hierarchically inside a main membrane, called the *skin*. Each membrane identifies a region inside the system. When a membrane has no membrane inside, it is called *elementary*. The objects are instances of symbols from a finite alphabet, and *multisets of objects* are placed in the regions determined by the membrane structure. The objects can evolve according to given *evolution rules*, associated with the regions. The *semantics* of cell–like P systems is defined by a non-deterministic and synchronous model. A *configuration* of a cell-like P system consists of a membrane structure and a sequence of multisets of objects, each associated with one region of the structure. At the beginning of the computation, the system is in the *initial configuration* also containing an *input multiset*. In each time step the system transforms its current configuration into another configuration by applying the evolution rules to the objects placed inside the regions of the system, in a non-deterministic and maximally parallel manner. A *computation* of the system is a (finite or infinite) sequence of configurations such that each configuration – except the initial one – is obtained from the previous one by a transition. A computation which reaches a configuration where no more rules can be applied to the existing objects and membranes, is called a *halting computation*. The result of a halting computation is usually defined by the multiset associated with a specific output membrane (or the environment) in the final configuration.

In this paper we deal with *recognizer* P systems, where all computations halt and exactly one of the distinguished objects yes and no is sent to the environment, and only in the last step of any computation, in order to signal acceptance or rejection, respectively. All recognizer P systems considered in this paper are *confluent*, meaning that if computations start from the same initial configuration then either all are accepting or all are rejecting. Recognizer P systems can thus be used to recognize formal languages (equivalently, solve decision problems). Let us recall that a decision problem X is a pair (I_X, θ_X) where I_X is a language over a finite alphabet and θ_X is a predicate (a total Boolean function) over I_X. The elements of I_X are called *instances* of the problem, and those for which predicate θ_X is true (respectively false) are called *positive* (respectively *negative*) instances. A *polynomial encoding* of a decision problem X is a pair (cod, s) of functions over I_X, computable in polynomial time by a deterministic Turing machine, such that for each instance $u \in I_X$, $s(u)$ is a natural number representing the *size* of the instance and $cod(u)$ is a multiset representing an encoding of the instance. Polynomial encodings are stable under polynomial time reductions.

2.1. The Classes $\mathcal{AM}^0_{-d,+ne,+ant}$ and $\mathcal{AM}^0_{-d,+ne,+ant\_NoPri}$

Formally, a polarizationless P system with active membranes, without dissolution, with division of elementary and non-elementary membranes and with annihilation rules is a construct of the form $\Pi = (\Gamma, H, \mu, w_1, \ldots, w_m, R)$, where $m \geq 1$ is the initial degree of the system, Γ is the alphabet of *objects*, H is a finite set of *labels* for membranes, μ is a *membrane structure* consisting of m membranes labelled with elements of H, $w_1, \ldots, w_m$ are strings over Γ, describing the *multisets of objects* placed in the m regions of μ, R is a finite set of *rules* of the following

types:

(a_0) $[a \to u]_h$ for $h \in H$, $a \in \Gamma$, $u \in \Gamma^*$ (*object evolution rules*).

(c_0) $[a]_h \to b\,[\ ]_h$ for $h \in H$, $a,b \in \Gamma$ *(send-out rules)*.

(e_0) $[[\,]_{h_1}[\,]_{h_2}]_{h_0} \to [[\,]_{h_1}]_{h_0}\ [[\,]_{h_2}]_{h_0}$, for $h_0,h_1,h_2 \in H$ *(division rules for non-elementary membranes)*.

(f_0) $[a\overline{a} \to \lambda]_h$ for $h \in H$, $a,\overline{a} \in \Gamma$ *(annihilation rules)*.
The pair of objects $a,\overline{a} \in \Gamma$ occurring simultaneously inside membrane h disappears.

The rules are applied in parallel and in a maximal manner. In one step, each object of a membrane can be used by at most one rule (chosen in a non–deterministic way), and each membrane can be the subject of *at most one* rule of types (c_0) and (e_0).

The class of all polarizationless recognizer P systems with active membranes, without dissolution and with division of elementary and non-elementary membranes as well as with matter/antimatter annihilation rules is denoted by $\mathcal{AM}^0_{-d,+ne,ant}$ if these rules have weak priority over the other rules, whereas without this priority we have $\mathcal{AM}^0_{-d,+ne,ant\_NoPri}$.

Theorem 2.1 $\mathbf{PMC}_{\mathcal{AM}^0_{-d,+ne,+ant\_NoPri}} = \mathbf{P}$.

Theorem 2.2 $\mathbf{PMC}_{\mathcal{AM}^0_{-d,+ne,+ant}} = \mathbf{NP}$.

Proof. We show the claim by constructing a family of recognizer P systems solving the **NP**-complete problem SAT. Consider the pairing function $\langle\ ,\ \rangle$ defined by $\langle n,m\rangle = ((n+m)(n+m+1)/2)+n$, which is polynomial-time computable. For any given formula in CNF, $\varphi = C_1 \wedge \cdots \wedge C_m$, with m clauses and n variables $Var(\varphi) = \{x_1,\ldots,x_n\}$ we construct a P system $\Pi(\langle n,m\rangle)$ solving it, where the multiset encoding the problem to be the input of $\Pi(\langle n,m\rangle)$ is $cod(\varphi) = \{x_{i,j} :\ x_j \in C_i\} \cup \{y_{i,j} :\ \neg x_j \in C_i\}$.

$$
\begin{aligned}
\Pi &= (O,\Sigma,H=\{1,2\},\mu = [\,[\ ]_2\,]_1, w_1, w_2, R, i_{in}=2), \text{ where}\\
\Sigma &= \{x_{i,j}, y_{i,j} \mid 1 \le i \le m,\ 1 \le j \le n\},\\
O &= \{d,t,f,F,\overline{F},T,\overline{no}_{n+5},\overline{F}_{n+5},\overline{yes}_{n+6},yes_{n+6},no_{n+6},yes,no\}\\
&\quad \cup \{x_{i,j}, y_{i,j} \mid 1 \le i \le m,\ -1 \le j \le n\} \cup \{\overline{x}_{i,-1}, \overline{y}_{i,-1} \mid 1 \le i \le m\}\\
&\quad \cup \{c_i, \overline{c}_i \mid 1 \le i \le m\} \cup \{e_j \mid 1 \le j \le n+3\}\\
&\quad \cup \{yes_j, no_j, F_j \mid 0 \le j \le n+5\},\\
w_1 &= no_0\ yes_0\ F_0,\ w_2 = d^n\ e_1, \text{ and the following rules in } R:
\end{aligned}
$$

Generation: $[\,d\,]_2 \to [\,t\,]_2[\,f\,]_2$; $[\,t \to \overline{y}_{1,-1}\cdots\overline{y}_{m,-1}\,]_2$; $[\,f \to \overline{x}_{1,-1}\cdots\overline{x}_{m,-1}\,]_2$;
$[\,\overline{x}_{i,-1} \to \lambda\,]_2$, $[\,\overline{y}_{i,-1} \to \lambda\,]_2$, $1 \le i \le m$.

In each step j, $1 \le j \le n$, every elementary membrane is divided, one new membrane corresponding with assigning *true* to variable j and the other one with assigning *false* to it. One step later, proper objects are produced to annihilate the input objects associated to variable j: in the *true* (*false*) case, we introduce the antimatter object for the (negated) variable, which then will annihilate the corresponding (negated) variable. Remaining barred (antimatter) objects not having been annihilated with the input objects, are erased in the next step.

Input Processing: $[\, x_{i,j} \to x_{i,j-1} \,]_2$, $[\, y_{i,j} \to y_{i,j-1} \,]_2$, $1 \le i \le m$, $0 \le j \le n$;
$[\, x_{i,-1}\, \overline{x}_{i,-1} \to \lambda \,]_2$, $[\, y_{i,-1}\, \overline{y}_{i,-1} \to \lambda \,]_2$, $1 \le i \le m$;
$[\, x_{i,-1} \to c_i \,]_2$, $[\, y_{i,-1} \to c_i \,]_2$, $1 \le i \le m$.

Input objects associated with variable j decrement their second subscript during $j+1$ steps to -1. Any variable not representing the desired truth value is eliminated by the corresponding antimatter object, whereas any of the input variables not annihilated then, shows that the associated clause i is satisfied, which situation is represented by the introduction of the object c_i.

Checking: $[\, e_j \to e_{j+1} \,]_2$, $1 \le j \le n+1$; $[\, e_{n+2} \to \overline{c}_1 \cdots \overline{c}_m e_{n+3} \,]_2$;
$[\, c_i\, \overline{c}_i \to \lambda \,]_2$, $[\, \overline{c}_i \to F \,]_2$, $[\, F\, \overline{F} \to \lambda \,]_2$, $1 \le i \le m$; $[\, e_{n+3} \to \overline{F} \,]_2$, $[\, \overline{F} \,]_2 \to [\ ]_2 T$.

It takes $n+2$ steps to produce objects c_i for every satisfied clause, possibly multiple times. Starting from object e_1, we have obtained the object e_{n+2} until then; from this object e_{n+2}, at step $n+2$ one anti-object is produced for each clause. Any of these clause anti-objects that is not annihilated, is transformed into F, showing that the chosen variable assignment did not satisfy the corresponding clause. It remains to notice that object T is sent to the skin (at step $n+4$) if and only if an object $\overline{F}$ did not get annihilated, i.e., no clause failed to be satisfied.

Output: $[\, yes_j \to yes_{j+1} \,]_1$, $[\, no_j \to no_{j+1} \,]_1$, $0 \le j \le n+5$; $[\, F_j \to F_{j+1} \,]_1$, $0 \le j \le n+4$;
$[\, T \to \overline{no}_{n+5} \overline{F}_{n+5} \,]_1$; $[\, no_{n+5}\, \overline{no}_{n+5} \to \lambda \,]_1$; $[\, no_{n+6} \,]_1 \to [\ ]_1 no$;
$[\, F_{n+5}\, \overline{F}_{n+5} \to \lambda \,]_1$; $[\, F_{n+5} \to \overline{yes}_{n+6} \,]_1$;
$[\, yes_{n+6}\, \overline{yes}_{n+6} \to \lambda \,]_1$; $[\, yes_{n+6} \,]_1 \to [\ ]_1 yes$.

If no object T has been sent to the skin, then the initial no-object can count up to $n+6$ and then send out the negative answer no, while the initial object F counts up to $n+5$, generates the antimatter object for the yes-object at stage $n+6$ and annihilates with the corresponding object yes at stage $n+6$. If (at least one) object T arrives in the skin, then the object no is annihilated at stage $n+5$ before being sent out in the next step, and the object F is annihilated before it could annihilate with the object yes, so that the positive answer yes is sent out in step $n+6$. □

References

[1] A. ALHAZOV, B. AMAN, R. FREUND, P systems with anti-matter. In: *Membrane Computing - 15th International Conference, CMC 2014, Prague, Czech Republic, August 20-22, 2014, Revised Selected Papers*. Lecture Notes in Computer Science 8961, Springer, 2014, 66–85.

[2] A. ALHAZOV, B. AMAN, R. FREUND, GH. PĂUN, Matter and Anti-Matter in Membrane Systems. In: *DCFS 2014*. Lecture Notes in Computer Science 8614, Springer, 2014, 65–76.

[3] D. DÍAZ-PERNIL, F. PEÑA-CANTILLANA, A. ALHAZOV, R. FREUND, M. GUTIÉRREZ-NARANJO, Antimatter as a frontier of tractability in membrane computing. *Fundamenta Informaticae* **134** (2014), 83–96.

[4] L. PAN, GH. PĂUN, Spiking neural P systems with anti-spikes. *International Journal of Computers, Communications & Control* **4** (2009) 3, 273–282.

[5] GH. PĂUN, G. ROZENBERG, A. SALOMAA (eds.), *The Oxford Handbook of Membrane Computing*. Oxford University Press, Oxford, England, 2010.

[6] THE P SYSTEMS WEBSITE, `http://ppage.psystems.eu`.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 34–37.

Der GapSet Algorithmus

Andrea Jaax[(A)] Stefan Näher[(B)]

[(A)]Universität Trier
jaax@uni-trier.de

[(B)]Universität Trier
naeher@uni-trier.de

Zusammenfassung

In dieser Arbeit wird der GapSet Algorithmus präsentiert. Dies ist ein neuer Algorithmus, der zur Berechnung minimaler Schnitte in Graphen entwickelt und implementiert wurde und stellt eine Verallgemeinerung des Algorithmus von Hao und Orlin dar [3]. Der flussbasierte Minimum Cut Algorithmus baut auf drei Charakteristika auf: der Informations–Wiederverwendung, der Partitionierung der Knotenmenge in GapSets und der Technik Flussverdopplung. Die Komplexität dieses Algorithmus beträgt $\mathcal{O}(n^2\sqrt{m})$. Er gehört somit zu den schnellsten theoretischen Minimum Cut Algorithmen.

1. Das Minimum Cut Problem

Ein Schnitt C eines ungerichteten Graphen G, teilt die Knotenmenge in zwei disjunkte, nichtleere Teilmengen S und T. Der Wert eines Schnittes entspricht der Summe der Kantenkapazitäten, welche einen Endpunkt in S und einen in T haben, d.h. $cap(C) = \sum_{e=(v,w)\in E v\in S, w\in T} cap(e)$. Ein (s,t)–Schnitt teilt den Graphen in zwei Teilmengen, so dass s und t auf unterschiedlichen Seiten des Schnittes liegen. Derjenige mit kleinstem Wert wird jeweils als minimaler $\big((s,t)$–$\big)$ Schnitt bezeichnet. Für weitere Informationen verweisen wir auf [1, 4, 5].

2. Die drei Charakteristika des GapSet Algorithmus

Der GapSet Algorithmus berechnet $n-1$ minimale (S,t)–Schnitte mit einem leicht modifizierten Preflow–Push Algorithmus. Hierbei arbeitet der Algorithmus mit einer Menge S von Quellknoten und einer Menge T, welche alle noch zu betrachtenden Senken beinhaltet. GapSets werden mit T_i bezeichnet.

2.1. Informations–Wiederverwendung

Berechnete Fluss- und Distanzwerte einer minimalen (S,t)–Schnitt Berechnung können als Startwert der nächsten verwendet werden. Hierbei spielt die Wahl der nächsten Senke eine wichtige Rolle.

Remark 2.1 *Die Wiederverwendung der Distanzmarkierung legt die Wahl der nächsten Senke t_i wie folgt fest:*

$$t_i \in \{v \in T \mid d(v) \leq d(w) \ \ \forall w \in T\}.$$

Die Bedingungen einer gültigen Distanzmarkierung sind in diesem Algorithmus gelockert:

Definition 2.2 *Eine Distanzmarkierung ist* gültig*, wenn die folgenden Ungleichungen gelten:*

$$d(t) = \min\{d(v) \mid v \in T\}, \tag{1}$$

$$d(v) \leq d(w) + 1 \textit{ for all edges } (v,w) \in E_f. \tag{2}$$

2.2. GapSets

Bei der Wiederverwendung der Distanzmarkierung ist eine besondere Gap Behandlung zwingend notwendig. Diese wird mit Hilfe der sogenannten GapSets realisiert.

Definition 2.3 *Eine* GapSet T_i *ist eine Knotenmenge $T_i \subseteq T = V \setminus S$, welche bisher noch nicht von einer Gap geteilt wurde. Die GapSet, welche die Senke t der aktuellen minimalen (S,t)–Schnitt Berechnung enthält, wird als* Arbeitsmenge T_w *bezeichnet.*

GapSets werden verwendet um eine geeignete Gap Behandlung zu realisieren. Knoten, welche die aktuelle Senke nicht mehr erreichen können, sollen in einer minimalen (S,t)–Schnitt Berechnung keine Rolle mehr spielen, ihre Distanzmarkierung jedoch erhalten bleiben. Wird eine GapSet als Arbeitsmenge ausgewählt, befindet sie sich in einem gültigen Zustand.

Definition 2.4 *Eine GapSet T_i befindet sich in einem* gültigen Zustand*, wenn die Summe der Überschüsse in T_i dem maximalen Fluss nach T_i entspricht. Es gilt:*

$$\sum_{v \in T_i} excess(v) = \sum_{e=(v,w) \in G, v \notin T_i, w \in T_i} cap(e).$$

2.3. Flussverdopplung

GapSets sind erst dann unabhängig voneinander, wenn sie sich alle in einem gültigen Zustand befinden. Dieser Zustand kann mit Hilfe der Flussverdopplung bei Entstehung einer neuen GapSet erreicht werden.

Algorithm 1 Flussverdopplung

```
1: function FLOWDUPLICATION(set of nodes T_w, set of nodes T_new)
2:     for e = (v,w) ∈ E_f, v ∈ T_w, w ∈ T_new do
3:         excess(w) ← excess(w) + cap_f(e);
4:     end for
5: end function
```

3. Der iterative GapSet Algorithmus

Aufbauend auf den drei Charakteristika erhalten wir den iterativen GapSet Algorithmus.
Ein minimaler (S,t)–Schnitt wird mit Hilfe der Funktion MINSTCUTFD(node t, set of nodes T_i) (Algorithmus 3) berechnet. Diese basiert auf dem Preflow–Push Algorithmus und wird mit der besonderen Gap Behandlung erweitert.

Algorithm 2 GapSet Algorithmus

```
1:  function GapSetMinCut(graph G, node_array cap)
2:      choose node s arbitrarily;
3:      for all nodes v ∈ V do
4:          d(v) ← 0;
5:      end for
6:      for all edges e ∈ E do
7:          if source(e) = s then
8:              push δ := cap_f(e) units of flow from s to target(e);
9:          else
10:             f(e) ← 0;
11:         end if
12:     end for
13:     t_min ← nil; f_min ← ∞; T_w ← V \ {s};
14:     while ∃ T_i ≠ ∅ do                                          ▷ main loop
15:         choose t ∈ {v ∈ T_i | T_i arbitrary GapSet and d(v) ≤ d(w)∀w ∈ T_i} ;
16:         MinStCutFD(t, T_i);
17:         if f_{S,t} < f_min then
18:             t_min ← t; f_min ← f_{S,t};
19:         end if
20:         T_i = T_i \ {t};
21:         for all edges e ∈ E with source(e) = t and target(e) ∈ T_i do
22:             push δ := cap_f(e) units of flow from t to target(e);
23:         end for
24:     end while
25: end function
```

4. Komplexität und Korrektheit

Die Korrektheit des Algorithmus wird in Theorem 4.1, die Komplexität in Theorem 4.2 gezeigt. Befinden sich GapSets in ihrem gültigen Zustand, dann werden nur Knoten und Kanten innerhalb der jeweiligen GapSet zur Berechnung minimaler (S,t)–Schnitte betrachtet. Außerdem ist die Distanzmarkierung innerhalb jeder GapSet gültig. Diese Erkenntnisse führen zur Korrektheit des Algorithmus.

Theorem 4.1 *Sei G ein ungerichteter Graph mit einer Kapazitätsfunktion $cap : E \to \mathbb{R}^+$. Dann löst der GapSet Algorithmus (Algorithmus 2) das minimale Schnitt Problem für Graph G.*

Proof. Sei $[S^*, T^*]$ ein minimaler Schnitt des Graphen G. Der GapSet Algorithmus berechnet unter Berücksichtigung der Wahl der nächsten Senke $n-1$ maximale (S,t)–Fluss Berechnungen mit $t \in T$. Sei bei diesen Flussberechnungen t^* die erste Senke aus T^* und T_i die zugehörige GapSet mit $t^* \in T_i$, dann ist $S \cup T_i^{\downarrow}$, mit $T_k^{\downarrow}$ als die Vereinigung aller später entstandenen GapSets inklusive der Arbeitsmenge T_w, eine Teilmenge von S^*. Dann berechnet der Schleifendurchlauf mit t^* als Senke den minimalen Schnitt von G. □

Die Analyse der Komplexität leitet sich fast vollständig aus der Analyse der Komplexität des

Algorithm 3 Berechnung eines minimalen (S,t)–Schnittes

```
1:  function MinStCutFD(node t, set of nodes T_w)
2:      while there is an active node v ∈ T_w with d(v) < n do
3:          if there is an admissible edge e = (v,w) with w ∈ T_w then        ▷ Push
4:              push δ := min{excess(v), cap_f(e)} units of flow from v to w;
5:          else
6:              if ∄e = (v,w) with w ∈ T_w then                              ▷ new GS
7:                  T_new ← {v}; T_w ← T_w \ T_new;
8:                  FlowDuplication(T_w, T_new);
9:              else
10:                 if d(w) ≠ d(v) for all w ∈ T_w then                      ▷ new GS
11:                     T_new ← {w | w ∈ T_w ∧ d(w) ≥ d(v)}; T_w ← T_w \ T_new;
12:                     FlowDuplication(T_w, T_new);
13:                 else
14:                     d(v) = 1 + min{d(w) : (v,w) ∈ E_f}                   ▷ Relabel
15:                 end if
16:             end if
17:         end if
18:     end while
19: end function
```

Preflow–Push Algorithmus ab (siehe [2]). Neben der Anzahl der Relabel und Push Operationen, kann die Komplexität der Verwaltung der GapSets auf einfache Weise auf $\mathcal{O}(n^2)$ abgeschätzt werden. Die Komplexität der Flussverdopplung (Algorithmus 1) beträgt $\mathcal{O}(m)$.

Theorem 4.2 *Der GapSet Algorithmus (Algorithmus 2) benötigt bei Verwendung der Highest–Label Regel $\mathcal{O}(n^2)$ für die Verwaltung der GapSets, $\mathcal{O}(n^2)$ Relabel Operationen, $\mathcal{O}(nm)$ saturierende und $\mathcal{O}(n^2\sqrt{m})$ nicht–saturierende Push Operationen.*

Literatur

[1] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] A. V. Goldberg, R. E. Tarjan, A New Approach to the Maximum Flow Problem. In: *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*. STOC '86, ACM, New York, NY, USA, 1986, 136–146.

[3] J. Hao, J. B. Orlin, A Faster Algorithm for Finding the Minimum Cut in a Graph. In: *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '92, Society for Industrial and Applied Mathematics, Philadelphia, 1992, 165–174.

[4] S. O. Krumke, H. Noltemeier, *Graphentheoretische Konzepte und Algorithmen*. 1 edition, Teubner Verlag, 2005.

[5] K. Mehlhorn, S. Näher, *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 38.

A Simple and Fast Linear-Time Algorithm for Proportional Apportionment

Raphael Reitzig[(A)] Sebastian Wild[(A)]

[(A)]TU Kaiserslautern, Fachbereich Informatik
{reitzig,wild}@cs.uni-kl.de

Abstract

Cheng and Eppstein describe a linear-time algorithm for computing highest-average allocations in proportional apportionment scenarios, for instance assigning seats to parties in parliament so that the distribution of seats resembles the vote tally as well as possible.

We propose another linear-time algorithm that consists of only one call to a rank selection algorithm after elementary preprocessing. Our algorithm is conceptually simpler and faster in practice than the one by Cheng and Eppstein.

A detailed version can be found on arXiv: http://arxiv.org/abs/1504.06475

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 39–43.

Why Is Dual-Pivot Quicksort Fast?

Sebastian Wild[(A)]

[(A)]Fachbereich Informatik, TU Kaiserslautern
wild@cs.uni-kl.de

Abstract

I discuss the new dual-pivot Quicksort that is nowadays used to sort arrays of primitive types in Java. I sketch theoretical analyses of this algorithm that offer a possible, and in my opinion plausible, explanation why (a) dual-pivot Quicksort is faster than the previously used (classic) Quicksort and (b) why this improvement was not already found much earlier.

1. Introduction

Quicksort became popular shortly after its original presentation by Hoare [7] and many authors contributed variations of and implementation tricks for the basic algorithm. From a practical point of view, the most notable improvements appear in [17, 15, 2, 14].

After the improvements to Quicksort in the 1990's, almost all programming libraries used almost identical versions of the algorithm: the classic Quicksort implementation had reached calm waters.

It was not until 2009, over a decade later, that previously unknown Russian developer Vladimir Yaroslavskiy caused a sea change upon releasing the outcome of his free-time experiments to the public: a dual-pivot Quicksort implementation that clearly outperforms the classic Quicksort in Oracle's Java 6. The core innovation is the arguably natural ternary partitioning algorithm given to the right.

Yaroslavskiy's finding was so surprising that people were initially reluctant to believe him, but his Quicksort has finally been deployed to millions of devices with the release of Java 7 in 2011.

How could this substantial improvement to the well-studied Quicksort algorithm escape the eyes of researchers around the world for nearly 50 years?

```
DUALPIVOTQUICKSORT(A, left, right)  // sort A[left..right]
 1  if right − left ≥ 1
        // Take outermost elements as pivots (replace by sampling)
 2      p := min{A[left], A[right]}
 3      q := max{A[left], A[right]}
 4      ℓ := left + 1;  g := right − 1;  k := ℓ
 5      while k ≤ g
 6          if A[k] < p
 7              Swap A[k] and A[ℓ];  ℓ := ℓ + 1
 8          else if A[k] ≥ q
 9              while A[g] > q and k < g
10                  g := g − 1
11              end while
12              Swap A[k] and A[g];  g := g − 1
13              if A[k] < p
14                  Swap A[k] and A[ℓ];  ℓ := ℓ + 1
15              end if
16          end if
17          k := k + 1
18      end while
19      ℓ := ℓ − 1;  g := g + 1
        // Put pivots to final position:
20      Swap A[left] and A[ℓ];  Swap A[right] and A[g]
21      DUALPIVOTQUICKSORT(A, left , ℓ − 1)
22      DUALPIVOTQUICKSORT(A, ℓ + 1, g − 1)
23      DUALPIVOTQUICKSORT(A, g + 1, right)
24  end if
```

For programs as heavily used as library sorting methods, it is advisable to back up experimental data with mathematically proven properties. The latter consider however only a *model* of reality, which may or may not reflect accurately enough the behavior of actual machines.

The answer to above question is in part a tale of the pitfalls of theoretical models, so we start with a summary of the mathematical analysis of Quicksort and the underlying model in Section 2. We then briefly discuss the "memory wall" metaphor and its implications for Quicksort in Section 3, and finally propose an alternative model in Section 4 that offers an explanation for the superiority of Yaroslavskiy's Quicksort.

2. Analysis of Quicksort

The classical model for the analysis of sorting algorithm considers the average number of key comparisons on random permutations. Quicksort has been extensively studied under this model, including variations like choosing the pivot as median of a sample [7, 4, 15, 10, 3]: Let c_n denote the expected number of comparisons used by classic Quicksort (as given in [16]), when each pivot is chosen as median of a sample of $2t+1$ random elements. c_n fulfills the recurrence

$$c_n \;=\; n-1 \;+\; \sum_{\substack{0\le j_1,j_2\le n-1\\ j_1+j_2=n-1}} \cdot \frac{\binom{j_1}{t}\binom{j_2}{t}}{\binom{n}{2t+1}}(c_{j_1}+c_{j_2}) \tag{1}$$

since $n-1$ comparisons are needed in the first partitioning step, and we have two recursive calls of random sizes, where the probability to have sizes j_1 and j_2 is given by the fraction of binomials (see [9] for details). This recurrence can be solved asymptotically [4, 15] to

$$c_n \;\sim\; \frac{1}{H_{2(t+1)}-H_{t+1}}\cdot n\ln n,$$

where $H_n=\sum_{i=1}^{n} 1/i$ is the nth harmonic number and $f(n)\sim g(n)$ means $\lim_{n\to\infty} f(n)/g(n)=1$. The mathematical details are beyond the scope of this abstract, but a rather elementary derivation is possible [10]. Large values of t are impractical; a good compromise in practice is given by the "ninther", the median of three medians, each chosen from three elements [2]. This scheme can be analyzed similarly to the above [3].

The author generalized Equation (1) to Yaroslavskiy's Quicksort [19, 18, 9]. Note that unlike for classic Quicksort, the comparison count of Yaroslavskiy's partitioning depends on pivot values, so its expected value has to be computed over the choices for the pivots. We obtain for tertiles-of-$(3t+2)$

$$c_n \;=\; \left(\frac{5}{3}-\frac{1}{9t+12}\right)\cdot n + O(1) \;+\; \sum_{\substack{0\le j_1,j_2,j_3\le n-2\\ j_1+j_2+j_3=n-2}} \frac{\binom{j_1}{t}\binom{j_2}{t}\binom{j_3}{t}}{\binom{n}{3t+2}}\cdot(c_{j_1}+c_{j_2}+c_{j_3}); \tag{2}$$

with solution

$$c_n \;\sim\; \frac{\frac{5}{3}-\frac{1}{9t+12}}{H_{3(t+1)}-H_{t+1}}\cdot n\ln n.$$

Oracle's Java runtime library previously used classic Quicksort with ninther, and now uses Yaroslavskiy's Quicksort with tertiles-of-five; the average number of comparisons are asymptotically $1.5697n\ln n$ vs. $1.7043n\ln n$. According to the comparison model, Yaroslavskiy's algorithm is significantly *worse* than classic Quicksort! Moreover, this result does not change qualitatively if we consider *all* primitive instructions of a machine instead of only comparisons [19, 9]. It is thus not surprising that researchers found the use of two pivots not promising [15, 6].

But if Yaroslavskiy's algorithm actually uses more comparisons and instructions, how comes it is still faster in practice? And why was this discrepancy between theory and practice not noticed earlier? The reason is most likely related to a phenomenon known as the "memory wall" [20, 13] or the "von-Neumann bottleneck" [1]: Processor speed has been growing considerably faster than memory bandwidth for a long time.

3. The Memory Wall

Based on the extensive data for the STREAM benchmark [12, 11], CPU speed has increased with an average annual growth rate of 46% over the last 25 years, whereas memory bandwidth, the amount of data transferable between RAM and CPU in a given amount of time, has increased by 37% per year. Even though one should not be too strict about the exact numbers as they are averages over very different architectures, a significant increase in *imbalance* is undeniable. Figure 1 gives a direct quantitative view of this trend.

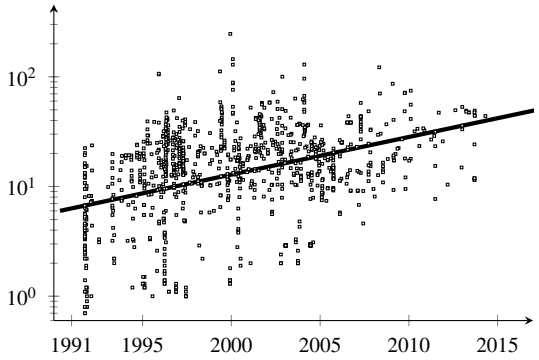


Figure 1: Development of CPU speed against memory bandwidth over the last 25 years. Each point shows one reported result of the STREAM benchmark [12, 11], with the date on the x-axis and the machine balance (peak MFLOPS divided by Bandwidth in MW/s in the "triad" benchmark) on a logarithmic y-axis. The fat line shows the linear regression (on log-scale). Data is taken from `www.cs.virginia.edu/stream/by_date/Balance.html`.

If the imbalance between CPU and memory transfer speed continues to grow exponentially, at some point in the future any further improvements of CPUs will be futile: the processor is waiting for data all the time; we hit a "memory wall".

It is debatable if and when this extreme will be reached [5, 13], and consequences certainly depend on the application. In any case, however, the (average) relative costs of memory accesses have increased significantly over the last 25 years.

So when Quicksort with two pivots was first studied, researchers correctly concluded that it does not pay off. But computers have changed since then, and so must our models.

4. Scanned Elements

Our new cost model for sorting counts the number of *"scanned elements"*. An element scan is essentially an accesses "$A[i]$" to the input array A, but we count all accesses as one that use the same index variable i *and* the same value for i. For example, a linear scan over A entails n scanned elements, and several interleaved scans (with different index variables) cost

the traveled distance, summed up over all indices, even when the scanned ranges overlap. We do not distinguish read and write accesses.

We claim that for algorithms built on potentially interleaved sequential scans, in particular for classic and dual-pivot Quicksort, the number of scanned elements is asymptotically proportional to the amount of data transfered between CPU and main memory [9].

Scanned elements are related to cache misses [8], but the latter is a machine-dependent quantity, whereas the former is a clean, abstract cost measure that is easy to analyze: One partitioning step of classic Quicksort scans A exactly once, resulting in n scanned elements. In Yaroslavskiy's partitioning, indices k and g together scan A once, but index ℓ scans the leftmost segment a second time. On average, the latter contains a third of all elements, yielding $\frac{4}{3}n$ scanned elements in total.

Using these in recurrences (1) resp. (2) yields $1.5697n\ln n$ vs. $1.4035n\ln n$ scanned elements; the Java 7 Quicksort saves 12% of the element scans over the version in Java 6, which matches the roughly 10% speedup observed in running time studies.

5. Conclusion

Memory speed has not fully kept pace with improvements in processing power. This growing imbalance forces us to economize on memory accesses in algorithms that were almost entirely CPU-bound in the past, and calls for new cost models for the analysis of algorithms. For sorting algorithms that build on sequential scans over their input, the proposed "scanned elements" counts serve as such a model and give a good indication of the amount of memory traffic caused by an algorithm. It is exactly this data traffic where dual-pivot outclasses classic Quicksort, offering a plausible explanation for its superiority in practice.

References

[1] J. BACKUS, Can Programming Be Liberated from the Von Neumann Style? A Functional Style and Its Algebra of Programs. *Communications of the ACM* **21** (1978) 8, 613–641.

[2] J. L. BENTLEY, M. D. MCILROY, Engineering a sort function. *Software: Practice and Experience* **23** (1993) 11, 1249–1265.

[3] M. DURAND, Asymptotic analysis of an optimized quicksort algorithm. *Information Processing Letters* **85** (2003) 2, 73–77.

[4] M. H. VAN EMDEN, Increasing the efficiency of quicksort. *Communications of the ACM* (1970), 563–567.

[5] M. A. ERTL, The Memory Wall Fallacy. 2001.
`https://www.complang.tuwien.ac.at/anton/memory-wall.html`

[6] P. HENNEQUIN, *Analyse en moyenne d'algorithmes : tri rapide et arbres de recherche*. PhD Thesis, Ecole Politechnique, Palaiseau, 1991.

[7] C. A. R. HOARE, Quicksort. *The Computer Journal* **5** (1962) 1, 10–16.

[8] S. KUSHAGRA, A. LÓPEZ-ORTIZ, A. QIAO, J. I. MUNRO, Multi-Pivot Quicksort: Theory and Experiments. In: *ALENEX 2014*. SIAM, 2014, 47–60.

[9] C. MARTÍNEZ, M. E. NEBEL, S. WILD, Analysis of Pivot Sampling in Dual-Pivot Quicksort. *Algorithmica* (2015).
http://arxiv.org/abs/1412.0193

[10] C. MARTÍNEZ, S. ROURA, Optimal Sampling Strategies in Quicksort and Quickselect. *SIAM Journal on Computing* **31** (2001) 3, 683–705.

[11] J. D. MCCALPIN, *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. Technical report, University of Virginia, Charlottesville, Virginia, 1991-2007. Continually updated technical report.
http://www.cs.virginia.edu/~mccalpin/papers/bandwidth/bandwidth.html

[12] J. D. MCCALPIN, Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (1995), 19–25.
http://www.cs.virginia.edu/~mccalpin/papers/balance/index.html

[13] S. A. MCKEE, Reflections on the Memory Wall. In: *Proceedings of the first conference on computing frontiers*. 2004, 162–167.

[14] D. R. MUSSER, Introspective Sorting and Selection Algorithms. *Software: Practice and Experience* **27** (1997) 8, 983–993.

[15] R. SEDGEWICK, *Quicksort*. PhD Thesis, Stanford University, 1975.

[16] R. SEDGEWICK, Implementing Quicksort programs. *Communications of the ACM* **21** (1978) 10, 847–857.

[17] R. C. SINGLETON, Algorithm 347: an efficient algorithm for sorting with minimal storage [M1]. *Communications of the ACM* **12** (1969) 3, 185–186.

[18] S. WILD, *Java 7's Dual Pivot Quicksort*. Master thesis, University of Kaiserslautern, 2012.
http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:hbz:386-kluedo-34638

[19] S. WILD, M. E. NEBEL, Average Case Analysis of Java 7's Dual Pivot Quicksort. In: L. EPSTEIN, P. FERRAGINA (eds.), *ESA 2012*. LNCS 7501, Springer, 2012, 825–836.
http://arxiv.org/abs/1310.7409

[20] W. A. WULF, S. A. MCKEE, Hitting the Memory Wall: Implications of the Obvious. 1995.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 44–47.

Revisiting Shinohara's Algorithm for Computing Descriptive Patterns

Henning Fernau[(A)] Florin Manea[(B)] Robert Mercaş[(B)]
Markus L. Schmid[(A)]

[(A)]Trier University, Fachbereich IV – Abteilung Informatikwissenschaften, D-54286 Trier, Germany, {Fernau,MSchmid}@uni-trier.de

[(B)]Kiel University, Department of Computer Science, D-24098 Kiel, Germany, {flm,rgm}@informatik.uni-kiel.de

Abstract

A pattern α is a word of terminals and variables, and it describes the pattern language $L(\alpha)$ of all words that can be obtained by uniformly replacing variables with terminal words. In 1982, Shinohara presents an algorithm that computes a pattern that is descriptive for a finite set S of words, i. e., its pattern language contains S in the closest possible way. We generalise Shinohara's algorithm to subclasses of patterns and characterise those subclasses for which it is applicable. Furthermore, we characterise those classes for which Shinohara's algorithm has a polynomial running time (assuming $\mathsf{P} \neq \mathsf{NP}$). Moreover, we also investigate the complexity of finding a pattern that separates two sets of words.

1. Introduction

The class of pattern languages was introduced by Angluin [1] as a formalism to describe similarities of words with respect to their repeating factors. For example, the commonalities of the words $w_1 = \mathtt{abbaabaa}$, $w_2 = \mathtt{baabbabaabba}$ and $w_3 = \mathtt{abaaaba}$ can be described by the pattern $\alpha = x_1\mathtt{ba}x_1x_2$, where x_1 and x_2 are variables. The pattern language $L(\alpha)$ is the set of all words that can be obtained from α by uniformly substituting the occurrences of variables x_1 and x_2 by some non-empty words. Pattern languages are important in the context of learning theory (in fact, their introduction brought new life to the field of inductive inference (see, e. g., [2, 5, 6, 8, 3] and, for a survey, [7]), due to the relevance of so-called *descriptive patterns*, introduced in [1]. A pattern α is descriptive of a *sample* S (i. e., a finite set of words) if $S \subseteq L(\alpha)$ and there is no pattern β with $S \subseteq L(\beta) \subset L(\alpha)$. For example, $x_1\mathtt{baa}x_2x_3\mathtt{a}$ is descriptive of $S = \{w_1, w_2, w_3\}$, while the pattern α introduced above is not, since $S \subseteq L(x_1\mathtt{ba}x_1x_2\mathtt{a}) \subset L(\alpha)$.

Independent of its application for inductive inference, the task of computing descriptive patterns constitutes an interesting problem in its own right. The main obstacle of the application of descriptive patterns is that deciding on whether a given word can be generated by a given pattern, i. e., the membership problem for pattern languages, is NP-complete and any algorithm

computing a descriptive pattern of *maximal length* necessarily solves the membership problem and therefore, assuming $\mathsf{P} \neq \mathsf{NP}$, it cannot have a polynomial running time (see [1]).

In [6], Shinohara introduces an exponential time algorithm that computes descriptive patterns by performing membership queries, and he also provides subclasses of patterns with polynomial membership problem for which his algorithm computes descriptive patterns in polynomial time.[1] We unify and further extend both Angluin's insights with respect to hardness of descriptive patterns as well as Shinohara's work on their efficient computation. We show that for every sample S a Π-descriptive pattern exists if and only if the pattern $x_1 \in \Pi$ and, furthermore, that a modified version of Shinohara's algorithm can be used to compute Π-descriptive patterns (of maximal length) if and only if Π is a *Shinohara-class*, i. e., it contains the set $\{x_1x_2\cdots x_k \mid k \in \mathbb{N}\}$ and, for every $\alpha \in \Pi$, the pattern α' obtained by substituting some length i suffix of α by a sequence of new variables $y_1y_2\cdots y_i$ is also in Π. Within the set of Shinohara-classes of patterns, we prove that Π-descriptive patterns can be computed in polynomial time if and only if the question whether $\alpha \in \Pi$ and the membership problem for Π can be decided in polynomial time. We also investigate the *consistency problem* for classes Π of patterns, which is to decide, for two given finite sets P and N of words, whether there exists a pattern $\alpha \in \Pi$ with $P \subseteq L(\alpha)$ and $L(\alpha) \cap N = \emptyset$. This problem is much more difficult than the membership problem or the problem of computing descriptive patterns (under the assumption $\mathsf{P} \neq \mathsf{NP}$).

We conclude this introduction by some definitions. By $\mathrm{var}(\alpha)$ we denote the set of variables occurring in α, α is in *canonical form* if, for some $k \in \mathbb{N}$, $\mathrm{var}(\alpha) = \{x_1, \ldots, x_k\}$ and, for every i, $1 \leq i \leq k-1$, the leftmost occurrence of x_i is to the left of the leftmost occurrence of x_{i+1}. By $\mathrm{cf}(\alpha)$, we denote its canonical form. We define $\Sigma\text{-PAT} = (\Sigma \cup X)^+$ and $\text{PAT} = \bigcup_\Sigma \Sigma\text{-PAT}$. A class $\Pi \subseteq \text{PAT}$ is *natural* if, for a given α, $\mathrm{cf}(\alpha) \in \Pi$ is decidable and $x_1 \in \Pi$. The class Π is called *tractable* if $\mathrm{cf}(\alpha) \in \Pi$ and the membership problem for Π-pattern languages can be decided in polynomial time. For any pattern α and for every i, $0 \leq i \leq |\alpha|$, we define $\mathrm{tg}(\alpha, i) = \mathrm{cf}(\alpha[1..i] \cdot y_1y_2\cdots y_{|\alpha|-i})$, where $\{y_1, y_2, \ldots, y_{|\alpha|-i}\} \subseteq (X \setminus \mathrm{var}(\alpha))$ with $|\{y_1, y_2, \ldots, y_{|\alpha|-i}\}| = |\alpha| - i$. Furthermore, $\mathrm{tg}(\alpha) = \{\mathrm{tg}(\alpha, i) \mid 0 \leq i \leq |\alpha|\}$, $\mathrm{tg}(\Pi) = \bigcup_{\alpha\in\Pi} \mathrm{tg}(\alpha)$, and a natural class Π is a *Shinohara-class* if, for every $k \in \mathbb{N}$, Π contains a pattern of length k and $\mathrm{tg}(\Pi) = \Pi$.

2. The Hardness of Computing Π-Descriptive Patterns

We first note that descriptive patterns can only be computed for natural classes.

Theorem 2.1 *Let $\Pi \subseteq$ PAT. There is an effective procedure that, for a sample S, computes a Π-descriptive pattern of S if and only if Π is natural.*

Moreover, for the *polynomial time* computation of descriptive patterns, the polynomial time decidability of the membership problem and whether $\mathrm{cf}(\alpha) \in \Pi$ is crucial.

Lemma 2.2 *Let Π be a natural class of patterns. If there exists a polynomial time algorithm that, for a given sample S of size 2, computes a pattern that is Π-descriptive of S, then the membership problem for Π-pattern languages as well as the question whether, for a given pattern α, $\mathrm{cf}(\alpha) \in \Pi$ is decidable in polynomial time.*

[1] Note that the concept of descriptiveness can be easily restricted to subclasses Π of patterns; more precisely, a pattern α is Π-descriptive if $\alpha \in \Pi$, $S \subseteq L(\alpha)$ and there is no other pattern $\beta \in \Pi$ with $S \subseteq L(\beta) \subset L(\alpha)$.

3. Computing Descriptive Patterns for Shinohara-Classes

Lemma 2.2 shows that by computing Π-descriptive patterns, we necessarily solve the membership problem as well as the question $\mathrm{cf}(\alpha) \in \Pi$. However, if Π is a Shinohara-class, then a generalisation of Shinohara's algorithm [6] can compute Π-descriptive patterns such, that the only possibly non-efficient elements are membership queries and queries $\mathrm{cf}(\alpha) \in \Pi$.

This algorithm (called Π-DESCPAT in the following) can be sketched as follows. We start with a pattern $\alpha = x_1 x_2 \cdots x_m$, where m is the length of a shortest word w in the sample S. Then we move over α from left to right and at every position i, we try to refine α by first replacing x_i by the i^{th} symbol of w and then consecutively by all the variables that occur in the prefix $\alpha[1..i-1]$. As soon as one of these refinements yields a pattern that describes the sample S (and that is still in Π), we move on to the next position and if all refinements fail, then we keep variable x_i at position i (which means that x_i occurs in the final pattern that is computed). Since this algorithm always computes a descriptive pattern of the length of a shortest word in the sample, it will always produce a descriptive pattern of maximal length.

Lemma 3.1 *Let* Π *be a Shinohara-class and let* S *be a sample with shortest word* w*. On input* (S, w)*,* Π-DESCPAT *computes a* Π*-descriptive pattern of* S*. If* Π *is tractable, then* Π-DESCPAT *can be implemented so that it has polynomial running time.*

From Lemmas 2.2 and 3.1 we can conclude the following meta-theorem:

Theorem 3.2 *Let* Π *be a Shinohara-class. There exists a polynomial time algorithm that, for a given sample* S*, computes a pattern that is* Π*-descriptive of* S *if and only if* Π *is tractable.*

The significance of Theorem 3.2 is brought out by the observation that many classes of patterns that are known to be tractable are in fact Shinohara-classes, e. g., the class $\mathrm{PAT}_{\mathrm{reg}}$ of *regular patterns*, where every variable has only one occurrence (e. g., $x_1\mathtt{a}\mathtt{b}x_2x_3\mathtt{a}x_4$), and the class $\mathrm{PAT}_{\mathrm{nc}}$ of *non-cross* patterns, where the occurrences of variables are sorted by their index (e. g., $x_1\mathtt{a}x_1x_1x_2\mathtt{b}x_2x_3\mathtt{a}\mathtt{b}x_3x_3$). In [4], an infinite hierarchy of classes of patterns has been introduced, where every level of the hierarchy is a tractable Shinohara-class. We recall the definition of this hierarchy. For every variable y in α, $\mathrm{sc}_\alpha(y) = \{i, i+1, \ldots, j\}$, where i is the leftmost and j the rightmost position of y in α. The scopes of some variables $y_1, y_2, \ldots, y_k \in \mathrm{var}(\alpha)$ coincide if $\bigcap_{1 \leq i \leq k} \mathrm{sc}_\alpha(y_i) \neq \emptyset$. The scope coincidence degree of α ($\mathrm{scd}(\alpha)$ for short) is the maximum number of variables in α such that their scopes coincide. For every $k \in \mathbb{N}$, let $\mathrm{PAT}_{\mathrm{scd}\leq k} = \{\alpha \in \mathrm{PAT} \mid \mathrm{scd}(\alpha) \leq k\}$. Since, for every $k \in \mathbb{N}$, the membership problem for $\mathrm{PAT}_{\mathrm{scd}\leq k}$-pattern languages is solvable in polynomial time [4] and $\mathrm{PAT}_{\mathrm{scd}\leq k}$ is a Shinohara-class, we can compute $\mathrm{PAT}_{\mathrm{scd}\leq k}$-descriptive patterns in polynomial time. Furthermore, by increasing the bound on the scope coincidence degree, we can boost the accuracy of the computed descriptive patterns at the expense of a slower running time and, conversely, by decreasing this bound, we improve on the running time, but lose accuracy of the computed descriptive patterns.

4. The Consistency Problem for Patterns

The consistency problem (sometimes also called *separation problem*) is a formalisation of the natural task to find a rule that separates one set of examples from another and it arises in vari-

ous contexts, e. g., learning theory (PAC leraning), artificial intelligence, model checking. For arbitrary classes Π of patterns, the Π-consistency problem is in Σ_2^P, the second level of the polynomial-time hierarchy. That it is harder than the membership problem for pattern languages is also pointed out by the following result, since it shows that the consistency problem is NP-hard for classes Π, for which the membership problem can be solved in polynomial time.

Theorem 4.1 *Let $\Pi \subseteq$ PAT, c be a terminal symbol and $\Gamma = \{\beta_1 \cdots \beta_n \mid n \in \mathbb{N}, \beta_i \in \{x_i c, c x_i\}, 1 \leq i \leq n\}$. If $\Gamma \subseteq \Pi$, then the Π-consistency problem is NP-hard.*

Analogously to Lemma 2.2 with respect to the complexity of computing descriptive patterns, we can show that, under certain restrictions, the complexity of the membership problem for Π-pattern languages also necessarily transfers to the complexity of the Π-consistency problem.

Theorem 4.2 *Let $\Pi \subseteq$ PAT, such that, for all $\alpha \in \Pi$, α has at most k different terminal symbols. If the Π-consistency problem is solvable in polynomial time, then the membership problem for Π-pattern languages is solvable in polynomial time.*

References

[1] D. ANGLUIN, Finding patterns common to a set of strings. *Journal of Computer and System Sciences* **21** (1980), 46–62.

[2] S. LANGE, R. WIEHAGEN, Polynomial-time inference of arbitrary pattern languages. *New Generation Computing* **8** (1991), 361–370.

[3] Z. MAZADI, Z. GAO, S. ZILLES, Distinguishing Pattern Languages with Membership Examples. In: *Proceedings of the 8th International Conference on Language and Automata Theory and Applications, LATA*. LNCS 8370, 2014, 528–540.

[4] D. REIDENBACH, M. L. SCHMID, Patterns with Bounded Treewidth. *Information and Computation* **239** (2014), 87–99.

[5] R. REISCHUK, T. ZEUGMANN, Learning One-Variable Pattern Languages in Linear Average Time. In: *Proc. 11th Annual Conference on Computational Learning Theory, COLT 1998*. 1998, 198–208.

[6] T. SHINOHARA, Polynomial Time Inference of Pattern Languages and Its Application. In: *Proc. 7th IBM Symposium on Mathematical Foundations of Computer Science*. 1982, 191–209.

[7] T. SHINOHARA, S. ARIKAWA, Pattern Inference. In: K. JANTKE, S. LANGE (eds.), *Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report*. Lecture Notes in Artificial Intelligence 961, Springer, Berlin, 1995, 259–291.

[8] T. SHINOHARA, H. ARIMURA, Inductive inference of unbounded unions of pattern languages from positive data. *Theoretical Computer Science* **241** (2000), 191–209.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 48–51.

Die verallgemeinerte Lamplighter-Gruppe als Automatengruppe

Carolin Albrecht[(A)]

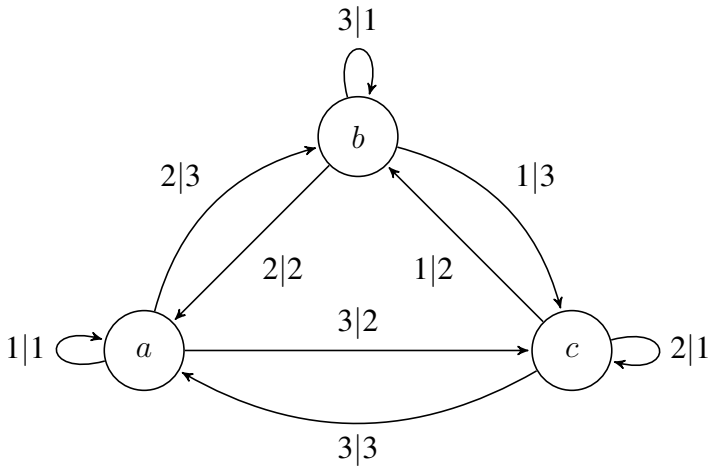
[(A)] Carolin.Albrecht@hhu.de

Zusammenfassung

In meinem Vortrag wird die Isomorphie der Lamplighter-Gruppe $\mathbb{Z}_3 \wr \mathbb{Z}$ zu einer Gruppe, die von einem bestimmten Automaten erzeugt wird, bewiesen. Nach [2] erzeugt eine invertierbare Mealy-Maschine eine Automatengruppe. Diese Automatengruppe hat bestimmte Eigenschaften, auf welche ebenfalls in [2] näher eingegangen wird. In [1] präsentieren I. Bondarenko, D. D'Angeli und E. Rodaro eine Mealy-Maschine A, und zeigen, dass die von A erzeugte Automatengruppe G_A isomorph zu $\mathbb{Z}_3 \wr \mathbb{Z}$ ist. Dieser Beweis wird in meiner Bachelorarbeit näher erläutert, auf der auch mein Vortrag aufbaut.

1. Der Automat A

Sei $A = (\Sigma, \Sigma, S, \delta, \lambda)$ die Mealy-Maschine mir den folgenden Daten. Sei $\Sigma = \{1,2,3\}, S = \{a,b,c\}$ und seien $\delta : S \times \Sigma \to S$, $\lambda : S \times \Sigma \to \Sigma$ so, dass sich A wie folgt darstellen lässt:



2. Die aus A erzeugte Gruppe G_A

Nach [2] lässt sich aus dem obigen Automaten A eine Gruppe erzeugen. Sie ergibt sich aus der Kranzrekursion von A:

$$a \mathrel{\widehat{=}} (a,b,c)(2\ 3),$$
$$b \mathrel{\widehat{=}} (c,a,b)(1\ 3),$$
$$c \mathrel{\widehat{=}} (b,c,a)(1\ 2),$$

Dabei erzeugen die Elemente a, b und c die Automatengruppe G_A. Die Multiplikation dieser Gruppe ist gegeben durch:

$$g \cdot h \mathrel{\widehat{=}} (g|_1 h|_{1\pi_g}, g|_2 h|_{2\pi_g}, \ldots, g|_d h|_{d\pi_g})\pi_g\pi_h$$

das Inverse ist gegeben durch:

$$g^{-1} \mathrel{\widehat{=}} (g|_{1\pi_g^{-1}}, g|_{2\pi_g^{-1}}, \ldots, g|_{d\pi_g^{-1}})\pi_g^{-1},$$

Auch der inverse und der duale Automat lassen sich durch Kranzrekursionen darstellen. Die inversen Elemente in G_A entsprechen der Kranzrekursion des inversen Automaten.

Es lässt sich außerdem zeigen, dass $ab^{-1} = bc^{-1} = ca^{-1} =: \alpha$, mit der Kranzrekursion $\alpha \mathrel{\widehat{=}} (\alpha,\alpha,\alpha)(1\ 3\ 2)$. Das Element α hat dann Ordnung 3, und sein Inverses ist $\alpha^{-1} = ac^{-1} = ba^{-1} = cb^{-1}$. Es gilt außerdem $a^{-1}b = b^{-1}c = c^{-1}a$ und $a^{-1}c = b^{-1}a = c^{-1}b$.

Daraus folgt, dass G_A von a und α erzeugt werden kann. Da wir aus [2] wissen, dass gilt

$$\left\langle \alpha, a \mid \alpha^3, [a^{-n}\alpha a^n, a^{-m}\alpha a^m], n,m \in \mathbb{Z} \right\rangle \simeq \mathbb{Z}_3 \wr \mathbb{Z},$$

bleibt nur noch zu zeigen, dass gilt

$$G_A = \left\langle \alpha, a \mid \alpha^3, [a^{-n}\alpha a^n, a^{-m}\alpha a^m], n,m \in \mathbb{Z} \right\rangle.$$

3. Die Untergruppe W von $\mathrm{Aut}(\Sigma^*)$

Sei W die Teilmenge der Automorphismengruppe von Σ^*, so dass für alle Elemente g aus W gilt: Es gibt Permutationen $(\pi_1, \pi_2, \ldots)$, $\pi_i \in \mathrm{Alt}_3$ für alle i aus $\mathbb{N}$, wobei Alt_3 die alternierende Gruppe der Ordnung 3 ist, sodass für $x_i \in X,\ n \in \mathbb{N}$ gilt:

$$(x_1\ x_2\ \ldots\ x_n)g = x_1\pi_1\ x_2\pi_2\ \ldots x_n\pi_n.$$

Dann gelten folgende Sätze nach [1]:

Satz 1 *Die Menge W ist eine Untergruppe der Automorphismengruppe* $\mathrm{Aut}(\Sigma^*)$.

Satz 2 *Die Gruppe W ist abelsch und hat den Gruppenexponenten* 3.

Satz 3 *Es gilt $\alpha, \alpha^{-1}, a^{-1}b, a^{-1}c \in W$.*

Satz 4 *Für alle $g \in W$ gilt $g = (h,h,h)\pi$ für ein $h \in W$, $\pi \in \mathrm{Alt}_3$.*

Lemma 1 *Für alle Elemente $g \in W$ und $x, y \in \{a, b, c\}$ gilt:*

$$x^{-1}gy, xgy^{-1} \in W.$$

Sei die Ordnung eines Wortes w definiert durch die Summe der Exponenten der Buchstaben in w. Sie wird bezeichnet mit $\mathrm{ord}(w)$.

Lemma 2 *Falls $g \in G_A$ als Wort über $\{a^{\pm 1}, b^{\pm 1}, c^{\pm 1}\}$ mit $\mathrm{ord}(w) = 0$ dargestellt werden kann, gilt $g \in W$. Im Besonderen gilt $g^3 = Id$ und diese Elemente kommutieren.*

Da alle $a^{-n}\alpha a^n$ die Bedingung aus Lemma 2 erfüllen, folgt:

Korollar 1 $[a^{-n}\alpha a^n, a^{-m}\alpha a^m] = Id$ *in* G_A.

4. Transitivität von G_A auf Σ^n

Lemma 3 *Die Automatengruppe G_A agiert transitiv auf Σ^n für alle $n \in \mathbb{N}$.*

Korollar 2 *Jedes Wort der Länge n über $\{a, b, c\}$ ist die Sektion eines anderen Wortes der Länge n über $\{a, b, c\}$.*

Korollar 3 *Die Semigruppe, die von $\{a, b, c\}$ generiert wird, ist frei.*

4.1. Beweis von $G_A \simeq \mathbb{Z}_3 \wr \mathbb{Z}$

Proof. Sei N die Untergruppe von G_A, die von den Elementen $a^{-n}\alpha a^n$, $n \in \mathbb{Z}$ generiert wird. Dann ist N nach Lemma 2 eine normale, abelsche Untergruppe mit Exponent 3. Wir wollen nun zeigen, dass gilt:

$$N = \bigoplus_{\mathbb{Z}} \langle a^{-n}\alpha a^n \rangle \simeq \bigoplus_{\mathbb{Z}} \mathbb{Z}_3$$

Sei e eine Relation wie folgt:

$$(a^{-n_1}\alpha^{\epsilon_1}a^{n_1})(a^{-n_2}\alpha^{\epsilon_2}a^{n_2})\dots(a^{-n_k}\alpha^{\epsilon_k}a^{n_k}) = e,$$

wobei $\epsilon_i \in \{\pm 1\}$ und $n_1 < n_2 < \dots < n_k$. Dabei werde $\alpha = ab^{-1}$ und $\alpha^{-1} = ac^{-1}$ substituiert. Durch Einsetzen in den obigen Ausdruck ergibt sich die Relation

$$a^{-n_1+1}w^{-1}a^{n_k} = e,$$

wobei w ein Wort über $\{a, b, c\}$ mit mindestens einem Auftreten von b oder c ist. Durch Korollar 3 erhalten wir einen Widerspruch, woraus die Behauptung folgt: Nach Korollar 3 hat das Element a eine endliche Ordnung und es gilt $N \cap \langle a \rangle = \{e\}$. Da $G_A = N\langle a \rangle$, und a auf N durch eine Shift-Konjugation wirkt, erhalten wir $G_A \simeq \mathbb{Z}_3 \wr \mathbb{Z}$. □

Literatur

[1] I. BONDARENKO, D. D'ANGELI, E. RODARO: The Lamplighter group $\mathbb{Z}_3 \wr \mathbb{Z}$ generated by a bireversible Automaton. 2015. Aufzurufen unter: http://arxiv.org/pdf/1502.07981v1

[2] L. BARTHOLDI, P. SILVA: Groups defined by automata. 2010. Aufzurufen unter: http://arxiv.org/pdf/1012.1531v1

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 52–55.

Linking Theorems for Tree Transducers

Zoltán Fülöp[(A)] Andreas Maletti[(B)]

[(A)]Department of Foundations of Computer Science, University of Szeged
Árpád tér 2, H-6720 Szeged, Hungary fulop@inf.u-szeged.hu

[(B)]Institute of Computer Science, Universität Leipzig
Augustusplatz 10–11, 04109 Leipzig, Germany maletti@informatik.uni-leipzig.de

Given a word (or vector) $w \in \Sigma^*$ and $1 \leq i \leq |w|$, we write w_i for the i^{th} letter in w. Further, for every position $w \in \mathrm{pos}(t)$ and tree $t \in T_\Sigma(S)$, we denote by $t(w)$ the label of t at w, and by $t|_w$ the w-rooted subtree of t. The positions are totally ordered by the lexicographic order $\sqsubseteq$ on $\mathbb{N}_+^*$ and partially ordered by the prefix order $\leq$ on $\mathbb{N}_+^*$. Given a finite set $P \subseteq \mathbb{N}_+^*$ of positions, we let $\vec{P} = (w_1, \ldots, w_k)$ be the vector of the positions of P in lexicographic order, where $P = \{w_1, \ldots, w_k\}$ with $w_1 \sqsubset \cdots \sqsubset w_k$. For a sequence $\vec{u} = (u_1, \ldots, u_n)$ of trees and positions $\vec{w} = (w_1, \ldots, w_n)$ of t that are pairwise incomparable with respect to $\leq$, we let $t[\vec{u}]_{\vec{w}}$ denote the tree obtained from t by replacing (in parallel) all subtrees $t|_{w_i}$ at w_i by u_i for all $1 \leq i \leq n$. In the special case $n = 1$, we also use the notation $t[u_1]_{w_1}$. For every $s \in S$, we let $\mathrm{pos}_s(t) = \{w \in \mathrm{pos}(t) \mid t(w) = s\}$. If $|\mathrm{pos}_s(t)| \leq 1$ for every $s \in S$, then the tree $t \in T_\Sigma(S)$ is linear, and we denote the set of all linear trees of $T_\Sigma(S)$ by $T_\Sigma^{\mathrm{lin}}(S)$. We reserve the sets $X = \{x_i \mid i \in \mathbb{N}_+\}$ and $X_n = \{x_i \mid 1 \leq i \leq n\}$ of variables. A tree $t \in T_\Sigma(X_n)$ is an n-context over Σ if t is linear and all variables of X_n occur in t. The set of all n-contexts over Σ is denoted by $C_\Sigma(X_n)$. Given $c \in C_\Sigma(X_n)$ and $t_1, \ldots, t_n \in T_\Sigma$, we write $c[t_1, \ldots, t_n]$ for $c[\vec{t}]_{\vec{w}}$, where $\vec{t} = (t_1, \ldots, t_n)$ and $\vec{w} = (w_1, \ldots, w_n)$ with $w_i \in \mathrm{pos}_{x_i}(c)$ being the unique position of x_i in c for every $1 \leq i \leq n$.

A *multi bottom-up tree transducer* (for short: MBOT) is a tuple $M = (Q, \Sigma, I, R)$ where Q is the alphabet of *states*, $I \subseteq Q$ contains the *initial states*, Σ is the alphabet of *input* and *output symbols* such that $\Sigma \cap Q = \emptyset$, and $R \subseteq T_\Sigma^{\mathrm{lin}}(Q) \times Q \times T_\Sigma(Q)^*$ is the nonempty, finite set of *rules*. We write $\ell \xrightarrow{q} \vec{r}$ for a rule $\langle \ell, q, \vec{r} \rangle \in R$. We require that all states in $\vec{r}$ appear in ℓ for every $\langle \ell, q, \vec{r} \rangle \in R$. If $|\vec{r}| \leq 1$ for all $\ell \xrightarrow{q} \vec{r}$ in R, then M is a (linear) *extended top-down tree transducer with regular look-ahead* [1, 3, 8] (for short: XTOP$^{\mathrm{R}}$), and if $|\vec{r}| = 1$ for all $\ell \xrightarrow{q} \vec{r}$ in R, then it is a (linear) *nondeleting* XTOP$^{\mathrm{R}}$ (for short: n-XTOP). Finally, it is *ε-free* if $\ell \notin Q$ for all $\ell \xrightarrow{q} \vec{r}$ in R. Each rule $\ell \xrightarrow{q} \varepsilon$ is a *look-ahead rule* because it can be used to check whether an input subtree belongs to a certain regular tree language [5]. For the remaining discussion, let $M = (Q, \Sigma, I, R)$ be an MBOT. An example is the ε-free MBOT $M_{\mathrm{ex}} = (\{q\}, \Sigma, \{q\}, R)$ with $\Sigma = \{\sigma, \gamma_1, \gamma_2, \alpha\}$ and the set R of rules containing $\sigma(\alpha, q, \alpha) \xrightarrow{q} \sigma(q, \alpha, q)$, $\gamma_1(q) \xrightarrow{q} \gamma_1(q) \,.\, \gamma_1(q)$, $\gamma_2(q) \xrightarrow{q} \gamma_2(q) \,.\, \gamma_2(q)$, and $\alpha \xrightarrow{q} \alpha \,.\, \alpha$.

A *link* is just an element $(v, w) \in \mathbb{N}_+^* \times \mathbb{N}_+^*$. A *sentential form over Q and Σ* is a tuple $\langle \xi, A, D, \zeta \rangle$, where $\xi, \zeta \in T_\Sigma(Q)$ and $A, D \subseteq \mathrm{pos}(\xi) \times \mathrm{pos}(\zeta)$. Elements in A and D are called *active* and *disabled* links, respectively. We denote by $\mathcal{SF}(Q, \Sigma)$ the set of all sentential

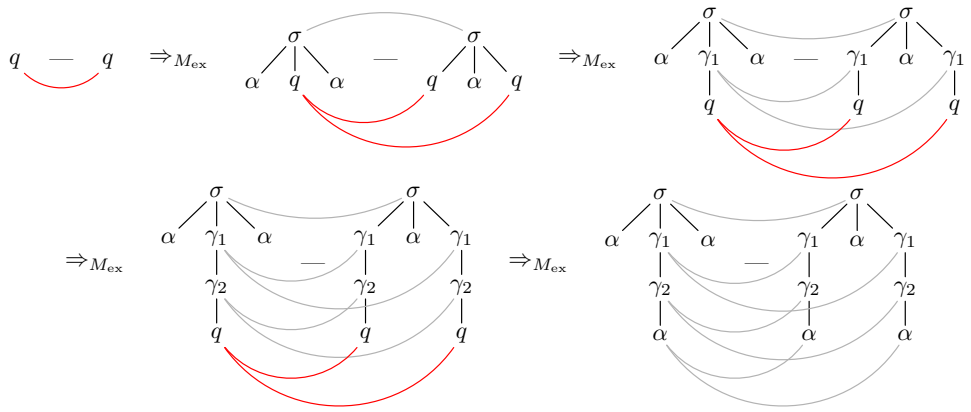


Figure 1: A derivation of the MBOT M_{ex}. The active links are clearly marked, whereas disabled links are shown in light gray.

forms over Q and Σ. The *link structure* $\text{links}_{v,\vec{w}}(\ell \overset{q}{—} \vec{r})$ of the rule $\ell \overset{q}{—} \vec{r} \in R$ for positions v and $\vec{w} = (w_1, \ldots, w_{|\vec{r}|})$ with $v, w_1, \ldots, w_{|\vec{r}|} \in \mathbb{N}_+^*$ is

$$\text{links}_{v,\vec{w}}(\ell \overset{q}{—} \vec{r}) = \bigcup_{p \in Q} \bigcup_{i=1}^{|\vec{r}|} \{(vv', w_i w') \mid v' \in \text{pos}_p(\ell), w' \in \text{pos}_p(r_i)\} \ .$$

Given $\langle \xi, A, D, \zeta \rangle, \langle \xi', A', D', \zeta' \rangle \in \mathcal{SF}(Q, \Sigma)$, we write $\langle \xi, A, D, \zeta \rangle \Rightarrow_M \langle \xi', A', D', \zeta' \rangle$ if there exist a rule $\ell \overset{q}{—} \vec{r} \in R$, an input position $v \in \text{pos}_q(\xi)$, and actively linked output positions $\vec{w} = A(v)$ such that (i) $|\vec{r}| = |\vec{w}|$, $\xi' = \xi[\ell]_v$, and $\zeta' = \zeta[\vec{r}]_{\vec{w}}$, and (ii) $D' = D \cup L$ and $A' = (A \setminus L) \cup \text{links}_{v,\vec{w}}(\ell \overset{q}{—} \vec{r})$ with $L = \{(v, w) \mid w \in A(v)\}$. The *set $\mathcal{SF}(M)$ of sentential forms computed by M* is $\{\langle \xi, A, D, \zeta \rangle \in \mathcal{SF}(Q, \Sigma) \mid \exists q \in I \colon \langle q, \{(\varepsilon, \varepsilon)\}, \emptyset, q \rangle \Rightarrow_M^* \langle \xi, A, D, \zeta \rangle\}$, and the *dependencies $\mathcal{D}(M)$ computed by M* are $\{\langle t, D, u \rangle \mid t, u \in T_\Sigma, \langle t, \emptyset, D, u \rangle \in \mathcal{SF}(M)\}$. Finally, the *tree relation computed by M* is $M = \{\langle t, u \rangle \mid \langle t, D, u \rangle \in \mathcal{D}(M)\}$. A short derivation using the MBOT M_{ex} is shown in Figure 1.

Next, we introduce some important properties for sets of links, sentential forms, and the set of dependencies computed by an MBOT (see [6]). A set $L \subseteq \mathbb{N}_+^* \times \mathbb{N}_+^*$ of links is (i) *input hierarchical* if $v_1 < v_2$ implies both $w_2 \not< w_1$ and that there exists $(v_1, w_1') \in L$ with $w_1' \leq w_2$, and (ii) *strictly input hierarchical* if $v_1 < v_2$ implies $w_1 \leq w_2$ and $v_1 = v_2$ implies that w_1 and w_2 are comparable with respect to $\leq$, for all $(v_1, w_1), (v_2, w_2) \in L$. A sentential form $\langle \xi, A, D, \zeta \rangle$ is (strictly) input hierarchical whenever $A \cup D$ is. Finally, $\mathcal{D}(M)$ has those properties if for each $\langle t, D, u \rangle \in \mathcal{D}(M)$ the corresponding sentential form $\langle t, \emptyset, D, u \rangle$ has them [i.e., D has them]. The property *(strictly) output hierarchical* can be defined by requiring the corresponding input-side property for the inverted set L^{-1} of links, the inverted sentential form $\langle \zeta, A^{-1}, D^{-1}, \xi \rangle$, and the set $\mathcal{D}(M)^{-1} = \{\langle u, D^{-1}, t \rangle \mid \langle t, D, u \rangle \in \mathcal{D}(M)\}$. The links L of the last derivation step of Figure 1 are input hierarchical and strictly output hierarchical, but not strictly input hierarchical.

Theorem 1 (see [6, Lm. 22]) *Let M be an* MBOT. *(i) The set $\mathcal{D}(M)$ is input hierarchical and strictly output hierarchical. (ii) If M is an* XTOP$^{\text{R}}$*, then $\mathcal{D}(M)$ is also strictly input hierarchical.*

Let $b \in \mathbb{N}$. A sentential form $\langle \xi, A, D, \zeta \rangle \in \mathcal{SF}(Q, \Sigma)$ has (i) *link distance b in the input* if for all links $(v_1, w_1), (v_1 v', w_2) \in A \cup D$ with $|v'| > b$ there exists a link $(v_1 v, w_3) \in A \cup D$

such that $v < v'$ and $1 \leq |v| \leq b$, and (ii) *strict link distance b in the input* if for all positions $v_1, v_1v' \in \mathrm{pos}(\xi)$ with $|v'| > b$ there exists a link $(v_1v, w_3) \in A \cup D$ such that $v < v'$ and $1 \leq |v| \leq b$. The set $\mathcal{D}(M)$ of dependencies has those properties if for each $\langle t, D, u\rangle \in \mathcal{D}(M)$ the corresponding sentential form $\langle t, \emptyset, D, u\rangle$ has them. Moreover, $\mathcal{D}(M)$ is *(strictly) link-distance bounded in the input* if there exists an integer $b \in \mathbb{N}$ such that it has (strict) link distance b in the input. A sentential form $\langle \xi, A, D, \zeta\rangle$ and $\mathcal{D}(M)$ have *(strict) link distance b in the output* if $\langle \zeta, A^{-1}, D^{-1}, \xi\rangle$ and $\mathcal{D}(M)^{-1}$ have (strict) link distance b in the input, respectively.

Theorem 2 *$\mathcal{D}(M)$ is link-distance bounded in the input and strictly link-distance bounded in the output. If M is an n-*XTOP*, then $\mathcal{D}(M)$ is also strictly link-distance bounded in the input.*

Our linking theorems establish the existence of certain interrelated links, which are forced simply by a subset of the computed tree relation. We need the following utility definitions. A tree $t \in T_\Sigma$ is a *chain* (or unary tree) if $\mathrm{pos}(t) \subseteq \{1\}^*$, and t is a *binary* tree if $\mathrm{pos}(t) \subseteq \{1,2\}^*$. A tree language $T \subseteq T_\Sigma$ is (i) *unary shape-complete* if for every chain $t \in T_\Sigma$ there exists a tree $t' \in T$ with $\mathrm{pos}(t') = \mathrm{pos}(t)$, and (ii) *binary shape-complete* if for every binary tree $t \in T_\Sigma$ there exists a tree $t' \in T$ with $\mathrm{pos}(t') = \mathrm{pos}(t)$. We now start with a linking theorem for the composition of arbitrarily many ε-free XTOP$^{\mathrm{R}}$. This theorem is only applicable to tree relations, which contain a sub-relation that is obtained with the help of an input and an output context into which we can plug trees from a unary shape-complete tree language. If such a tree relation τ is computed by a composition $\tau = M_1 ; \cdots ; M_k$ of ε-free XTOP$^{\mathrm{R}}$ $M_1, \ldots, M_k$, then we can deduce a dependency and the natural links relating the corresponding subtrees of the contexts.

Theorem 3 *Let $k, n \in \mathbb{N}_+$ and $M_1, \ldots, M_k$ be ε-free* XTOP$^{\mathrm{R}}$ *over Σ such that*

$$\{\langle c[t_1, \ldots, t_n], c'[t_1, \ldots, t_n]\rangle \mid t_1 \in T_1, \ldots, t_n \in T_n\} \subseteq M_1 ; \cdots ; M_k$$

for some $c, c' \in C_\Sigma(X_n)$ and unary shape-complete tree languages $T_1, \ldots, T_n \subseteq T_\Sigma$. Then there exist trees $t_1 \in T_1, \ldots, t_n \in T_n$, dependencies $\langle u_0, D_1, u_1\rangle \in \mathcal{D}(M_1), \ldots, \langle u_{k-1}, D_k, u_k\rangle \in \mathcal{D}(M_k)$ with $u_0 = c[t_1, \ldots, t_n]$ and $u_k = c'[t_1, \ldots, t_n]$, and a link $(v_{ji}, w_{ji}) \in D_i$ for each $1 \leq i \leq k$ and $1 \leq j \leq n$ such that (i) $\mathrm{pos}_{x_j}(c') \leq w_{jk}$ for all $1 \leq j \leq n$, (ii) $v_{ji} \leq w_{j(i-1)}$ for all $2 \leq i \leq k$ and $1 \leq j \leq n$, and (iii) $\mathrm{pos}_{x_j}(c) \leq v_{j1}$ for all $1 \leq j \leq n$.

We know that ε-free MBOT and several relevant subclasses are closed under composition [4]. Therefore, our second linking theorem concerns a single ε-free MBOT.

Theorem 4 *Let $n \in \mathbb{N}_+$ and $M = (Q, \Sigma, I, R)$ be an ε-free* MBOT *such that*

$$\{\langle c[t_1, \ldots, t_n], c'[t_1, \ldots, t_n]\rangle \mid t_1 \in T_1, \ldots, t_n \in T_n\} \subseteq M$$

for some $c, c' \in C_\Sigma(X_n)$ and binary shape-complete tree languages $T_1, \ldots, T_n \subseteq T_\Sigma$. Then there exist trees $t_1 \in T_1, \ldots, t_n \in T_n$, a dependency $\langle c[t_1, \ldots, t_n], D, c'[t_1, \ldots, t_n]\rangle \in \mathcal{D}(M)$ and a link $(v_j, w_j) \in D$ for every $1 \leq j \leq n$ such that (i) $\mathrm{pos}_{x_j}(c) \leq v_j$ for all $1 \leq j \leq n$ and (ii) $\mathrm{pos}_{x_j}(c') \leq w_j$ for all $1 \leq j \leq n$.

As application we present a classical result of [1], which states that the class of tree relations computed by XTOP$^{\mathrm{R}}$ (as well as those computed by n-XTOP) is not closed under composition.

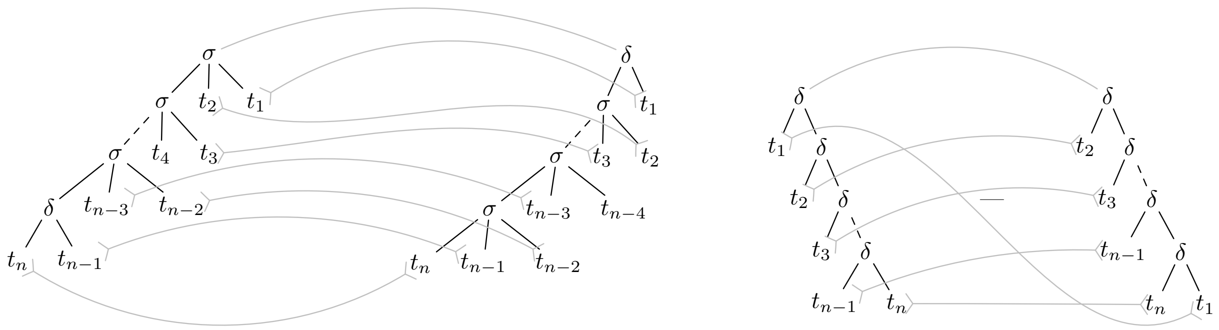


Figure 2: Counterexample relations of [1] (left) and M_{tpc}^{-1} (right) with links, which we conclude from Theorems 3 and 4, respectively, where an inverse arrow head indicates that the link refers to a node (not necessarily the root) inside the subtree that the spline points to.

Theorem 5 ([1, Sect. 3.4]) *The class of tree relations computable by ε-free* XTOP$^{\text{R}}$ *(or ε-free n-*XTOP*) is not closed under composition.*

Next we apply Theorem 4 and show that the inverse of abstract topicalization [2] cannot be computed by any ε-free MBOT. A tree language $L \subseteq T_\Sigma$ is *regular* [5] if there exists an MBOT M such that $L = \{t \mid \langle t,u \rangle \in M\}$. A tree relation $\tau \subseteq T_\Sigma \times T_\Sigma$ is *regularity preserving* if $\tau(L) = \{u \mid \langle t,u \rangle \in \tau,\, t \in L\}$ is regular for every regular tree language $L \subseteq T_\Sigma$.

Theorem 6 ([7, Thm. 8]) *The class of regularity preserving tree relations computable by ε-free* MBOT *is not closed under inverses.*

References

[1] A. ARNOLD, M. DAUCHET, Morphismes et Bimorphismes d'Arbres. *Theoret. Comput. Sci.* **20** (1982) 1, 33–93.

[2] N. CHOMSKY, *The Minimalist Program*. Current Studies in Linguistics, MIT Press, 1995.

[3] J. ENGELFRIET, Top-down Tree Transducers with Regular Look-ahead. *Math. Systems Theory* **10** (1977) 1, 289–303.

[4] J. ENGELFRIET, E. LILIN, A. MALETTI, Composition and Decomposition of Extended Multi Bottom-up Tree Transducers. *Acta Inf.* **46** (2009) 8, 561–590.

[5] F. GÉCSEG, M. STEINBY, Tree Languages. In: G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages*. 3. chapter 1, Springer, 1997, 1–68.

[6] A. MALETTI, Tree transformations and dependencies. In: *Proc. MOL*. LNAI 6878, Springer, 2011, 1–20.

[7] A. MALETTI, The Power of Regularity-Preserving Multi Bottom-up Tree Transducers. In: *Proc. CIAA*. LNCS 8587, Springer, 2014, 278–289.

[8] A. MALETTI, J. GRAEHL, M. HOPKINS, K. KNIGHT, The Power of Extended Top-down Tree Transducers. *SIAM J. Comput.* **39** (2009) 2, 410–430.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 56–59.

RegEx und Wortgleichungen: Ein Vergleich

Dominik D. Freydenberger

Universität Bayreuth
ddfy@ddfy.de

1. Einleitung

Reguläre Ausdrücke sind sowohl in der praktischen, als auch in der theoretischen Informatik von großer Bedeutung. Die vergleichsweise geringe Ausdrucksstärke von regulären Sprachen führt allerdings dazu, dass in vielen Fällen Erweiterungen betrachtet werden. Dabei werden häufig Wiederholungs- bzw. Gleichheitsoperatoren eingeführt, die nicht nur die Ausdrucksstärke, sondern auch die Handhabbarkeit der Modelle deutlich verändern. Obwohl sich viele dieser Modelle in ihren Details unterscheiden, ist es oft möglich, Gemeinsamkeiten der unterschiedlichen Modelle für obere und untere Schranken auszunutzen. Allerdings scheint eine allgemeine Theorie der „Regularität mit Wiederholungen“ bisher nicht in Reichweite zu sein; und es ist unklar, ob sie überhaupt möglich oder sinnvoll ist.

Als (kleiner) Schritt in Richtung einer solchen allgemeinen Theorie vergleicht dieser Vortrag zwei prominente Modelle aus diesem Umfeld. Das erste dieser Modelle sind *RegEx*, reguläre Ausdrücke die durch die in der Praxis verwendeten Wiederholungsoperatoren erweitert werden. Das zweite Modell sind *Wortgleichungen mit regulären Nebenbedingungen*, die in den letzten Jahren in der Wortkombinatorik ausführlich untersucht wurden. Außerdem werden aus diesen Betrachtungen neue Erkenntnisse zu den von Fagin et al. [2] eingeführten *Kern-Abschnittsanfragen* gewonnen.

Die hier vorgestellten Resultate bauen auf einer früheren Version von [4] auf, die auf dem Theorietag 2014 vorgestellt wurde. Um die Seitenbegrenzung einzuhalten, wurde auf erläuternde Erklärungen verzichtet. Dieser Bericht dient daher vor allem als Ergänzung des Vortrags.

1.1. RegEx

Aus Platzgründen erläutern wird dieses Modell nur durch Beispiele definiert; für eine vollständige Definition der Semantik verweise ich auf [3] oder [4]. Zusätzlich zu den bekannten Bestandteilen der klassischen regulären Ausdrücke darf ein *erweiterter regulärer Ausdruck* (kurz: *RegEx*) noch zweite weitere Operationen verwenden, nämlich die *Variablenbindung* $x\{\cdots\}$ und die *Variablenreferenz* $\&x$. Dabei wird der Ausdruck von links nach rechts gelesen. Ein Teilausdruck $x\{\alpha\}$ erzeugt ein Wort w auf die gleiche Art wie α, dieses wird außerdem in der Variable x gespeichert. Jedes Vorkommen von $\&x$ wiederholt nun dieses Wort w. Es ist auch möglich, Variablen neu zu binden.

Beispiel 1.1 *Sei $\alpha := x\{\mathtt{a}^*\}\mathtt{b}(x\{\mathtt{a}^*\} \vee (\mathtt{a}^*))b\&x$. Dann ist $\mathcal{L}(\alpha)$ die Menge aller Wörter der Form $\mathtt{a}^i\mathtt{ba}^j\mathtt{ba}^k$ mit $i,j,k \geq 0$, für die $k = i$ oder $k = j$ gilt.*

Sei $\beta := x\{\mathtt{aa}^+\}\&x^+$. Dann ist $\mathcal{L}(\beta)$ die Menge aller Wörter $\mathtt{a}^n$, $n \geq 2$, n ist keine Primzahl.

Eine RegEx ist *variablen-stern-frei*, wenn kein Teilausdruck der Form α^* weder Variablenbindungen, noch Variablenreferenzen enthält. Wir bezeichnen diese RegEx auch als *vsf-RegEx*.

1.2. Kern-Abschnittsanfragen

Allgemein ist eine Abschnittsanfrage eine Funktion, die ein Wort $w \in \Sigma^*$ auf eine Relation über Abschnitten von w abbildet. Wir betrachten hier nur eine eingeschränkte Klasse dieser Anfragen, die in [2] als *core spanners* bezeichnet wird. Diese bauen auf sogenannten *Regex-Formeln* auf; diese sind reguläre Ausdrücke, die zusätzlich um Variablen zum Markieren von Match-Bereichen erweitert werden. Zusätzlich dazu benötigen wir noch weitere Definitionen: Sei $w \in \Sigma^*$. Wir bezeichnen die Menge $[w] := \{i \in \mathbb{N} \mid 1 \leq i \leq |w|+1\}$ als die *Positionen* von w. Durch jedes Paar natürlicher Zahlen i,j mit $1 \leq i \leq j \leq |w|+1$ wird ein *Abschnitt von w* definiert als $[i,j\rangle := \{k \in \mathbb{N} \mid i \leq k < j\}$.

Dabei seien $[i,j\rangle$ und $[i',j'\rangle$ genau dann identisch, wenn $i = i'$ und $j = j'$ Ein Abschnitt von w beschreibt also nicht nur ein Teilwort von w, sondern die exakte Stelle, an der dieses Teilwort in w vorkommt. Der erste Buchstabe erhält hierbei die Position 1, und ist $w = w_1 \cdots w_n$ (mit $w_k \in \Sigma$), so entspricht der Abschnitt $[i,j+1\rangle$ dem Teilstück $w_i \cdots w_j$.

Beispiel 1.2 *Die Regex-Formel α über dem Terminalalphabet $\Sigma := \{\mathtt{a},\mathtt{b}\}$ und dem Variablenalphabet $V := \{x,y,z\}$ sei definiert als $\alpha := \Sigma^* \cdot z\big\{x\{\mathtt{a}\} \cdot \Sigma^* \cdot y\{\mathtt{b}\}\big\} \cdot \Sigma^*$. Außerdem sei $w := \mathtt{ababa}$. Es gilt $[w] = \{1,\ldots,6\}$. Die von α beschriebene Abschnittsanfrage $[\![\alpha]\!]$ bildet nun w auf die folgende Relation ab: $\{([1,2\rangle,[2,3\rangle,[1,3\rangle),([1,2\rangle,[4,5\rangle,[1,5\rangle),([3,4\rangle,[4,5\rangle,[3,5\rangle)\}$. Die Regex-Formel α bildet also w auf die Relation aller Tupel (x,y,z) ab, in denen x und y ein* `a` *bzw.* `b` *beschreiben, die aufeinander folgen. Dabei enthält z den kompletten Abschnitt, der zwischen diesen beiden Buchstaben aufgespannt wird. Hierbei entsprechen (zum Beispiel) die Abschnitte $[1,2\rangle$ und $[2,3\rangle$ unterschiedlichen Vorkommen von* `a`*. Da sie sich auf unterschiedliche Stellen von w beziehen, sind sie nicht identisch.*

Die in [2] definierten *core spanner* erweitern Regex-Formeln um die relationalen Operatoren *Vereinigung* $\cup$, *Projektion* π_X (wobei X eine Menge von Variablen ist), *natürlicher Join* $\bowtie$, und *Selektion* $\varsigma^{=}_{x_1,\ldots,x_n}$. Hierbei überprüft der Join, ob die in den Variablen gespeicherten Abschnitte identisch sind, während die Selektion lediglich überprüft, ob die von diesen Abschnitten beschriebenen Teilwörter gleich sind.

Anhand dieser Operatoren können nun komplexere Anfragen beschrieben werden. Die Menge aller Anfragen, die anhand von Regex-Formeln und den oben angegebenen Operatoren beschrieben werden können, bezeichnen wir als *Kern-Abschnittsanfragen*.

Wir interessieren uns dabei besonders für *boolesche Anfragen*, also Anfragen, bei denen auf die leere Variablenmenge projiziert wird. Die können als Entscheidungsfunktionen auf Wörtern interpretiert werden; somit definiert jede dieser Anfragen eine Sprache.

1.3. Patternsprachen und Wortgleichungen

Ein *Pattern* ist ein Wort $\alpha \in (\Sigma \cup X)^+$ über einem endlichen Terminalalphabet Σ und einem unendlichen Variablenalphabet X und definiert die Sprache

$$\mathcal{L}(\alpha) := \{\sigma(\alpha) \mid \sigma \text{ ist eine Patternsubstitution}\}.$$

Hierbei ist eine *Patternsubstitution* ein Homomorphismus $\sigma\colon (\Sigma \cup X)^* \to \Sigma^*$, d. h. Variablen werden homomorph durch Terminalwörter ersetzt.

Eine *Wortgleichung* $\eta := (\eta_L, \eta_R)$ wird definiert durch zwei Pattern η_L, η_R; hierbei kann η auch als $\eta_L = \eta_R$ geschrieben werden. Eine Patternsubstitution σ ist eine *Lösung* von η, wenn $\sigma(\eta_L) = \sigma(\eta_R)$. Für jedes Tupel $(x_1, \ldots, x_n)$ aus Variablen in η definieren wir die Relation

$$\mathcal{L}_{(x_1,\ldots,x_n)}(\eta) = \{(\sigma(x_1), \ldots, \sigma(x_n)) \mid \sigma \text{ ist eine Lösung von } \eta\}.$$

Eine *reguläre Nebenbedingungsfunktion* für η ist eine Funktion N, die jeder Variable in η eine reguläre Sprache $N(x)$ zuordnet. Eine Lösung σ von η ist eine Lösung von η unter den Nebenbedingungen N, wenn $\sigma(x) \in N(x)$ für jede Variable x in η gilt.

Die Menge der Relationen, die durch Wortgleichungen ausgedrückt werden können, entspricht genau der Menge der Relationen, die in er *existentiellen Theorie der Konkatenation* ausgedrückt werden können (siehe [1]). Diese verwendet als atomare Formeln Wortgleichungen und ergänzt diese durch $\wedge$, $\vee$ sowie existentielle Quantoren. Die Äquivalenz gilt analog, wenn diese Theorie durch reguläre Nebenbedingungen erweitert wird.

2. Resultate

2.1. Ausdrückbarkeit

Die folgenden Resultate basieren darauf, dass Wortgleichungen mit regulären Nebenbedingungen die gleiche Ausdrucksstärke haben wie die durch reguläre Nebenbedingungen erweiterte existentielle Theorie der Konkatenation:

Theorem 2.1 *Aus jeder Kern-Abschnittsanfrage lässt sich in polynomieller Zeit in eine äquivalente Wortgleichung mit regulären Nebenbedingungen erzeugen.*

Theorem 2.2 *Aus jeder vsf-RegEx lässt sich in polynomieller Zeit in eine äquivalente Wortgleichung mit regulären Nebenbedingungen erzeugen.*

Lemma 2.3 *Sei L eine Sprache über einem unären Alphabet Σ, die durch Wortgleichung mit regulären Nebenbedingungen definiert wird. Dann ist L semi-linear.*

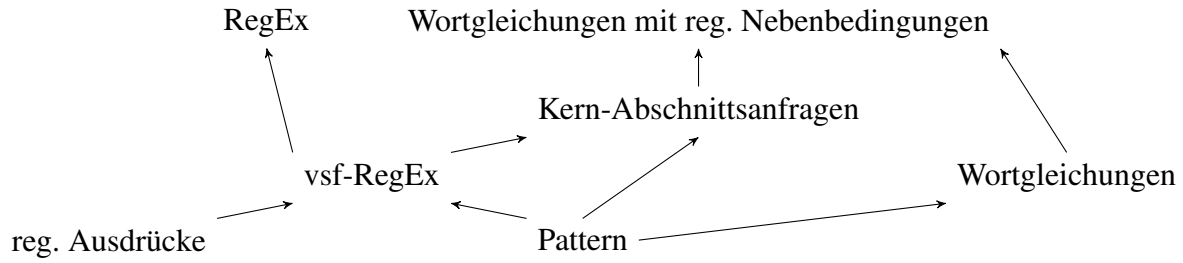
Proposition 2.4 *Es existiert eine Sprache, die sowohl durch eine Wortgleichungen mit regulären Nebenbedingungen als auch durch eine RegEx-Sprache definiert wird, die jedoch keine vsf-RegEx-Sprache ist.*

Theorem 2.5 *Es existiert eine Sprache, die durch eine Wortgleichungen mit regulären Nebenbedingungen definiert wird, jedoch keine RegEx-Sprache ist.*

Da die verwendete Sprache auch durch Kern-Abschnittsanfragen ausgedrückt werden kann, beantwortet dies eine offene Frage von Fagin et al. [2]. Außerdem gilt:

Proposition 2.6 *Es existieren Pattern α und β, so dass $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$ nicht durch eine RegEx ausgedrückt werden kann.*

Neben den anschließend vorgestellten Konsequenzen ergibt sich der folgende Zusammenhang:



2.2. Konsequenzen

Durch dieses Beobachtungen lassen sich frühere Resultate übertragen. Aus den in [3] vorgestellten Resulaten zu vsf-RegEx folgt:

Theorem 2.7 *Seien $L_1, L_2 \subseteq \Sigma^*$ Sprachen, die durch Wortgleichungen mit regulären Nebenbedingungen definiert sind. Die folgenden Probleme sind unentscheidbar:*
1. Ist $L_1 = L_2$? *2. Ist $L_1 = \Sigma^*$?* *3. Ist L_1 regulär?*

Außerdem ist der Tradeoff von Wortgleichungen mit regulären Nebenbedingungen zu regulären Ausdrücken durch keine rekursive Funktion beschränkt.

Da die Erfüllbarkeit von Wortgleichungen mit regulären Nebenbedingungen in PSPACE entschieden werden kann (siehe [1]), gilt:

Theorem 2.8 *Sei P eine Kern-Abschnittsanfrage. Die folgenden Probleme sind PSPACE-vollständig: 1. Ist P erfüllbar?* *2. Ist P hierarchisch?*

Theorem 2.9 *Seien α, β vsf-RegEx. Das folgende Problem ist PSPACE-vollständig:*
Ist $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta) = \emptyset$?

Literatur

[1] V. DIEKERT, Makanin's Algorithm. In: M. LOTHAIRE (ed.), *Algebraic Combinatorics on Words*. chapter 12, Cambridge University Press, 2002, 387–442.

[2] R. FAGIN, B. KIMELFELD, F. REISS, S. VANSUMMEREN, Document Spanners: A Formal Approach to Information Extraction. *J. ACM* **62** (2015) 2, 12.

[3] D. D. FREYDENBERGER, Extended Regular Expressions: Succinctness and Decidability. *Theory Comput. Sys.* **53** (2013) 2, 159–193.

[4] D. D. FREYDENBERGER, M. HOLLDACK, Document Spanners: From Expressive Power to Decision Problems. – Eingereicht.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 60–62.

Longest Gapped Repeats and Palindromes

Florin Manea

Department of Computer Science, Christian-Albrechts University of Kiel,
Christian-Albrechts-Platz 4, 24118 Kiel, Germany, flm@informatik.uni-kiel.de

Gapped repeats and palindromes were investigated already for a long time (see, e.g., [4, 6]), with motivation coming from the analysis of DNA and RNA structures, where tandem repeats or hairpin structures play important roles in revealing structural and functional information of the analysed genetic sequence. More precisely, a gapped repeat (respectively, palindrome) occurring in a word w is a factor uvu (respectively, u^Rvu) of w. The middle part v of such structures is called gap, while the two factors u (or the factors u^R and u) are called left and right arms. Generally, the previous works (e.g., [1, 6, 5, 3]) were interested in finding all the gapped repeats and palindromes, under certain restrictions on the length of the gap or on the relation between the arm of the repeat or palindrome and the gap.

Here, we are mainly interested in the construction of longest previous gapped repeat or palindrome tables: for each position i of the word we want to compute the longest factor occurring both at position i and once again on a position $j < i$ (or, respectively, whose mirror image occurs in the prefix of length $i-1$ of w) such that there is a gap (subject to various restrictions) between i and the previous occurrence of the respective factor (mirrored factor). Such tables give us a good image of the long gapped repeats and symmetries of the input word.

We first consider the case when the length of the gap is between a lower bound g and an upper bound G, where g and G are given as input (so, may depend on the input word). This extends naturally the case of lower bounded gaps. In the second case, the gap is lower bounded by a function that depends on the positions of the word; i.e., we might have a different bound at each position. Finally, following [6], we analyse gapped repeats uvu and palindromes u^Rvu where the length of the gap v is upper bounded by the length of the arm u; these structures are called long armed repeats and palindromes, respectively. In all cases, efficient algorithms constructing longest previous repeats or palindromes tables are given.

A more detailed description of the problems is given in the following. The results presented here were published in [2].

Problem 1 *Given w of length n and two integers g and G, such that $0 \leq g < G \leq n$, construct the arrays $LPrF_{g,G}[\cdot]$ and $LPF_{g,G}[\cdot]$ defined for $1 \leq i \leq n$:*

a. *$LPrF_{g,G}[i] = \max\{|u| \mid$ there exists v such that u^Rv is a suffix of $w[1..i-1]$ and u is prefix of $w[i..n]$, with $g \leq |v| < G\}$.*

b. *$LPF_{g,G}[i] = \max\{|u| \mid$ there exists v such that uv is a suffix of $w[1..i-1]$ and u is prefix of $w[i..n]$, with $g \leq |v| < G\}$.*

We are able to solve Problem 1(a) in linear time $\mathcal{O}(n)$. Problem 1(b) is solved here in $\mathcal{O}(n \log n)$ time. Intuitively, when trying to compute the longest prefix u of $w[i..n]$ such that $u^R v$ is a suffix of $w[1..i-1]$ with $g < |v| \leq G$, we just have to compute the longest common prefix between $w[i..n]$ and the words $w[1..j]^R$ with $g < i - j \leq G$. The increased difficulty in solving the problem for repeats (reflected in the increased complexity of our algorithm) seems to come from the fact that when trying to compute the longest prefix u of $w[i..n]$ such that uv is a suffix of $w[1..i-1]$ with $g < |v| \leq G$, it is hard to see where the uv factor may start, so we have to somehow try more variants for the length of u. In [1], the authors give an algorithm that finds all maximal repeats (i.e., repeats whose arms cannot be extended) with gap between a lower and an upper bound, running in $\mathcal{O}(n \log n + z)$ time, where z is the number of such repeats. It is worth noting that there are words (e.g., $(a^2b)^{n/3}$, from [1]) that may have $\Theta(nG)$ maximal repeats uvu with $|v| < G$, so for $G > \log n$ and $g = 0$, for instance, our algorithm is faster than an approach that would first use the algorithms of [1] to get all maximal repeats, and then process them somehow to solve Problem 1(b).

In the second case, the gaps are only lower bounded; however, the bound on the gap allowed at a position is defined by a function depending on that position.

Problem 2 *Given w of length n and the values $g(1), \ldots, g(n)$ of $g : \{1, \ldots, n\} \to \{1, \ldots, n\}$, construct the arrays $LPrF_g[\cdot]$ and $LPF_g[\cdot]$ defined for $1 \leq i \leq n$:*

a. *$LPrF_g[i] = \max\{|u| \mid$ there exists v such that $u^R v$ is a suffix of $w[1..i-1]$ and u is prefix of $w[i..n]$, with $g(i) \leq |v|\}$.*

b. *$LPF_g[i] = \max\{|u| \mid$ there exists v such that uv is a suffix of $w[1..i-1]$ and u is prefix of $w[i..n]$, with $g(i) \leq |v|\}$.*

The setting of this problem can be seen as follows. An expert preprocesses the input word (in a way specific to the framework in which one needs the problem solved), and detects the length of the gap occurring at each position (so, computes $g(i)$ for all i). These values and the word are then given to us, to compute the arrays defined in our problems. We are solve both problems in linear time.

Finally, following [5, 6], we analyse gapped repeats uvu and palindromes $u^R vu$ where the length of the gap v is upper bounded by the length of the arm u; these structures are called long armed repeats and palindromes, respectively.

Problem 3 *Given w of length n, construct the arrays $LPal[\cdot]$ and $LRep[\cdot]$, defined for $1 \leq i \leq n$:*

a. *$LPal[i] = \max\{|u| \mid$ there exists v such that $u^R v$ is a suffix of $w[1..i-1], u$ is a prefix of $w[i..n]$, and $|v| \leq |u|\}$.*

b. *$LRep[i] = \max\{|u| \mid$ there exists v such that uv is a suffix of $w[1..i-1], u$ is a prefix of $w[i..n]$, and $|v| \leq |u|\}$.*

In [5] one proposes an algorithm finding the set S of all factors of a word of length n which are maximal long armed palindromes (i.e., the arms cannot be extended to the right or to the left) in $\mathcal{O}(n + |S|)$ time; no upper bound on the possible size of the set S was given in [5], but it is widely believed to be $\mathcal{O}(n)$. Using the algorithm of [5] as an initial step, we solve

Problem 3(a) in $\mathcal{O}(n+|S|)$ time. In [6] the set of maximal long armed repeats with non-empty gap is shown to be of linear size and is computed in $\mathcal{O}(n)$ time. We use this algorithm and a linear time algorithm finding the longest square centred at each position of a word to solve Problem 3(b) in linear time.

Our algorithms are generally based on efficient data-structures. On one hand, we use efficient word-processing data structures like suffix arrays or longest common prefix structures. On the other hand, we heavily use specific data-structures for maintaining efficiently collections of disjoint sets, under union and find operations. Alongside these data-structures, we make use of a series of remarks of combinatorial nature, providing insight in the repetitive structure of the words.

Literatur

[1] G. S. BRODAL, R. B. LYNGSØ, C. N. S. PEDERSEN, J. STOYE, Finding Maximal Pairs with Bounded Gap. In: *Proc. 10th Annual Symposium on Combinatorial Pattern Matching*. LNCS 1645, Springer, 1999, 134–149.

[2] M. DUMITRAN, F. MANEA, Longest Gapped Repeats and Palindromes. In: *Proc. 40th International Symposium Mathematical Foundations of Computer Science*. LNCS 9234, Springer, 2015, 205–217.

[3] P. GAWRYCHOWSKI, F. MANEA, Longest α-Gapped Repeat and Palindrome. In: *Proc. 20th International Symposium Fundamentals of Computation Theory*. LNCS 9210, Springer, 2015, 27–40.

[4] D. GUSFIELD, *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, New York, NY, USA, 1997.

[5] R. KOLPAKOV, G. KUCHEROV, Searching for gapped palindromes. *Theor. Comput. Sci.* **410** (2009) 51, 5365–5373.

[6] R. KOLPAKOV, M. PODOLSKIY, M. POSYPKIN, N. KHRAPOV, Searching of Gapped Repeats and Subrepetitions in a Word. In: *Proc. CPM*. LNCS 8486, 2014, 212–221.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 63–66.

An Approach to Computing Downward Closures

Georg Zetzsche

Technische Universität Kaiserslautern
Fachbereich Informatik
Concurrency Theory Group
zetzsche@cs.uni-kl.de

1. Abstractions

A fruitful idea in the analysis of complex systems is that of abstractions: Instead of working with the original model, one considers a model that has a simpler structure but preserves pertinent properties.

A prominent example of such an abstraction is the *Parikh image*, which is available whenever the semantics of a model is given as a formal language. If L is a language over an alphabet $X = \{x_1, \ldots, x_n\}$, then its *Parikh image* $\Psi(L)$ consists of all vectors $(a_1, \ldots, a_n) \in \mathbb{N}^n$ such that there is a $w \in L$ in which x_i occurs a_i times for $i \in \{1, \ldots, n\}$. The Parikh image is a useful abstraction, provided that for the languages L at hand, $\Psi(L)$ can be described using simpler means than L itself. This means, most applications of Parikh images are confined to situations where $\Psi(L)$ is semilinear (or, equivalently, definable in Presburger arithmetic). For example, this is the case for context-free languages [13], languages accepted by blind (or revesal-bounded) counter automata [11], and combinations thereof [10, 8, 2].

Of course, there are important types of languages that do not guarantee semilinearity of their Parikh image. For example, Parikh images of languages accepted by higher-order pushdown automata are not semilinear in general: It is easy to construct a second-order pushdown automaton for the language $\{a^{2^n} \mid n \geq 0\}$.

2. Downward closures

However, there is an abstraction of languages that guarantees a simple description for *every language* whatsoever and still reflects important properties of the abstracted language—the downward closure. For words $u, v \in X^*$, we write $u \preceq v$ if $u = u_1 \cdots u_n$ and $v = v_0 u_1 v_1 \cdots u_n v_n$ for some $u_1, \ldots, u_n, v_0, \ldots, v_n \in X^*$. Then, the *downward closure of* L is defined as

$$L{\downarrow} = \{u \in X^* \mid \exists v \in L \colon u \preceq v\}.$$

In other words, the downward closure consists of the set of all (not necessarily contiguous) subwords of members of L. It has the remarkable property that it is regular for every set of words $L \subseteq X^*$, which is due to the fact that the subword ordering $\preceq$ is a well-quasi-ordering [9].

Moreover, downward closures preserve useul information about the abstracted language. Suppose L describes the behavior of a system that is observed through a lossy channel, meaning that on the way to the observer, arbitrary actions can get lost. Then, $L{\downarrow}$ is the set of words received by the observer [7]. Hence, given the downward closure as a finite automaton, we can decide whether two systems are equivalent under such observations, and even whether one system includes the behavior of another.

However, while there always *exists* a finite automaton for the downward closure, it appears to be difficult to *compute* these automata. There are few language classes for which computability has been established. Computability is currently known for

- *context-free languages* and *algebraic extensions* [3, 15],
- *0L-systems* and *context-free FIFO rewriting systems* [1],
- *Petri net languages* [7], and
- *stacked counter automata* [16].

Downward closures are not computable for reachability sets of *lossy channel systems* [12] and *Church-Rosser languages* [6].

3. New approach

This talk demonstrates a recently introduced [17] general method for the computation of downward closures. It relies on a fairly simple idea and reduces the computation to the so-called *simultaneous unboundedness problem (SUP)*. The latter asks, given a language $L \subseteq a_1^* \cdots a_n^*$, whether for each $k \in \mathbb{N}$, there is a word $a_1^{x_1} \cdots a_n^{x_n} \in L$ such that $x_1, \ldots, x_n \geq k$. Many language classes studied in formal language theory are *full trios*, meaning that they are closed under homomorphisms, inverse homomorphisms, and regular intersection.

Theorem 3.1 *For each full trio $\mathcal{C}$, the following are equivalent:*

- *Downward closures are computable for $\mathcal{C}$.*
- *The SUP is decidable for $\mathcal{C}$.*

This method yields new, sometimes greatly simplified, algorithms for each of the computability results above. It also opens up a range of other language classes to the computation of downward closures.

First, it implies computability for every full trio that exhibits effectively semilinear Parikh images. This re-proves computability results for *context-free languages* and *stacked counter automata* [16], but also applies to many other classes, such as the *multiple context-free languages* [14]. Second, the method yields the computability for *matrix grammars* [4, 5], a powerful grammar model that generalizes Petri net and context-free languages. Third, it is applied to obtain computability of downward closures for the *indexed languages*.

References

[1] P. A. ABDULLA, L. BOASSON, A. BOUAJJANI, Effective Lossy Queue Languages. In: *Proc. of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001)*. LNCS, Springer-Verlag, Heidelberg, 639–651.

[2] P. BUCKHEISTER, G. ZETZSCHE, Semilinearity and Context-Freeness of Languages Accepted by Valence Automata. In: *Proc. of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS 2013)*. LNCS 8087, Springer-Verlag, Heidelberg, 2013, 231–242.

[3] B. COURCELLE, On constructing obstruction sets of words. *Bulletin of the EATCS* **44** (1991), 178–186.

[4] J. DASSOW, G. PĂUN, *Regulated rewriting in formal language theory*. Springer-Verlag, Berlin, 1989.

[5] J. DASSOW, G. PĂUN, A. SALOMAA, Grammars with Controlled Derivations. In: G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages*. 2, Springer-Verlag, Berlin, 1997, 101–154.

[6] H. GRUBER, M. HOLZER, M. KUTRIB, The size of Higman-Haines sets. *Theoretical Computer Science* **387** (2007) 2, 167–176.

[7] P. HABERMEHL, R. MEYER, H. WIMMEL, The Downward-Closure of Petri Net Languages. In: *Proc. of the 37th International Colloquium on Automata, Languages and Programming (ICALP 2010)*. LNCS 6199, Springer-Verlag, Heidelberg, 466–477.

[8] M. HAGUE, A. W. LIN, Model Checking Recursive Programs with Numeric Data Types. In: *Computer Aided Verification*. LNCS 6806, Springer-Verlag, Heidelberg, 2011, 743–759.

[9] L. H. HAINES, On free monoids partially ordered by embedding. *Journal of Combinatorial Theory* **6** (1969) 1, 94–98.

[10] T. HARJU, O. IBARRA, J. KARHUMÄKI, A. SALOMAA, Some Decision Problems Concerning Semilinearity and Commutation. *Journal of Computer and System Sciences* **65** (2002) 2, 278–294.

[11] O. H. IBARRA, Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM* **25** (1978) 1, 116–133.

[12] R. MAYR, Undecidable problems in unreliable computations. *Theoretical Computer Science* **297** (2003) 1-3, 337–354.

[13] R. J. PARIKH, On Context-Free Languages. *Journal of the ACM* **13** (1966) 4, 570–581.

[14] H. SEKI, T. MATSUMURA, M. FUJII, T. KASAMI, On multiple context-free grammars. *Theoretical Computer Science* **88** (1991) 2, 191–229.

[15] J. VAN LEEUWEN, Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics* **21** (1978) 3, 237–252.

[16] G. ZETZSCHE, Computing Downward Closures for Stacked Counter Automata. In: *Proc. of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*. 743–756.

[17] G. ZETZSCHE, An Approach to Computing Downward Closures. In: *Proc. of the 42th International Colloquium on Automata, Languages and Programming (ICALP 2015)*. LNCS 9135, Springer-Verlag, Heidelberg, 2015, 440–451. Full version available at `http://arxiv.org/abs/1503.01068`.

THEORIE-TAGE 2015 H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 67–70.

Zu Beziehungen zwischen subregulären Sprachfamilien

Markus Holzer Bianca Truthe

Justus-Liebig-Universität Gießen, Institut für Informatik
Arndtstr. 2, D-35392 Gießen, Germany
{markus.holzer,bianca.truthe}@informatik.uni-giessen.de

Zusammenfassung

Wir zeigen, dass jede definite Sprache von einem geordneten deterministischen endlichen Automaten akzeptiert wird. Damit wird ein auf dem Theorietag 2014 vorgestelltes Ergebnis verbessert. Außerdem beweisen wir, dass jede Sprache, die regulär und unter Suffixbildung abgeschlossen ist, auch potenzseparierend ist. Dies steht im Gegensatz dazu, dass nicht-reguläre Sprachen, die unter Suffix-Bildung abgeschlossen sind, nicht notwendigerweise potenzseparierend sind. Zum Schluss wird eine Hierarchie präsentiert, die zusätzlich die Familien der monoidalen, endlichen, nilpotenten, kombinatorischen, regulären nicht-zählenden, regulären kommutativen, regulären zirkulären und vereinigungsfreien Sprachen enthält.

1. Einleitung

Die Langfassung dieses Beitrages ist auf der Konferenz NCMA 2015 vorgestellt worden und im Tagungsband [1] erschienen.

Zu einem Alphabet V bezeichnen wir mit V^* die Menge aller Wörter, mit V^n die Menge aller Wörter der Länge n und mit $V^{\leq n}$ die Menge aller Wörter der Länge höchstens n über V. Das Leerwort wird mit λ bezeichnet.

Es sei L eine Sprache über einem Alphabet V. Wir sagen, die Sprache L ist in Bezug auf V

- monoidal, falls $L = V^*$ gilt,
- kombinational, falls sie die Form $L = V^*A$ für eine Teilmenge $A \subseteq V$ hat,
- definit, falls sie die Form $L = A \cup V^*B$ für zwei endliche Teilmengen A und B von V^* hat,
- nilpotent, falls sie oder ihr Komplement $V^* \setminus L$ endlich ist,
- kommutativ, falls sie mit jedem Wort auch all seine Permutationen enthält,
- zirkulär, falls sie mit jedem Wort auch all seine zyklischen Verschiebungen enthält,
- suffix-abgeschlossen, falls sie mit jedem Wort auch all seine Suffixe enthält,
- nicht-zählend (non-counting, star-free), falls es eine natürliche Zahl $k \geq 1$ so gibt, dass für je drei Wörter $x \in V^*$, $y \in V^*$ und $z \in V^*$ die Bedingung $xy^kz \in L$ genau dann, wenn $xy^{k+1}z \in L$ gilt,

- potenzseparierend (power-separating), falls es eine natürliche Zahl $m \geq 1$ so gibt, dass zu jedem Wort $x \in V^*$ entweder $J_x^m \cap L = \emptyset$ oder $J_x^m \subseteq L$ mit $J_x^m = \{ x^n \mid n \geq m \}$ gilt,
- geordnet, falls sie von einem deterministischen endlichen Automaten (V, Z, z_0, F, δ) akzeptiert wird, bei dem eine Ordnung $\prec$ auf der Zustandsmenge so existiert, dass $z \prec z'$ die Beziehung $\delta(z,a) \preceq \delta(z',a)$ für jeden Buchstaben $a \in V$ impliziert,
- vereinigungsfrei (union-free), falls sie als regulärer Ausdruck nur unter Verwendung von Konkatenation und Stern beschrieben werden kann.

Mit ***FIN***, ***MON***, ***COMB***, ***NIL***, ***DEF***, ***ORD***, ***RNC***, ***SUF***, ***RSUF***, ***PS***, ***RPS***, ***RCOMM***, ***RCIRC***, ***UF***, und ***REG***, bezeichnen wir die Familien aller endlichen, monoidalen, kombinationalen, nilpotenten, definiten, geordneten, regulären nicht-zählenden, suffix-abgeschlossenen, regulären suffix-abgeschlossenen, potenzseparierenden, regulären potenzseparierenden, regulären kommutativen, regulären zirkulären, vereinigungsfreien bzw. regulären Sprachen.

2. Ergebnisse

Satz 2.1 *Jede definite Sprache wird von einem geordneten deterministischen endlichen Automaten akzeptiert.*

Beweis. Es seien V ein Alphabet und L eine definite Sprache. Dann gibt es zwei endliche Sprachen $A \subset V^*$ und $B \subset V^*$ mit $L = A \cup V^*B$. Es sei $m = \max\{ |w| \mid w \in A \cup B \}$.

Wir konstruieren einen deterministischen endlichen Automaten $\mathcal{A} = (V, Z, z_\lambda, F, \delta)$, der die Sprache L akzeptiert, so, dass in den Zuständen das zuletzt gelesene Teilwort bis zur Länge $m+1$ gespeichert wird. Wir setzen

$$Z = \{ z_w \mid w \in V^{\leq m+1} \}$$

als Zustandsmenge und z_λ als Startzustand. Die akzeptierenden Zustände sind jene Zustände z_w, bei denen das Wort w oder jedes Wort uw für $u \in V^*$ zur Sprache L gehört:

$$F = \{ z_w \mid w \in A \cup B \cup (V^+B \cap V^{\leq m}) \cup \{ w \mid |w| = m+1 \text{ und } w \in V^*B \} \}.$$

Zu einem Wort $w \in V^*$ und einer natürlichen Zahl k sei $[w]_k$ das längste Suffix von w mit einer Länge von höchstens k:

$$[w]_k = \begin{cases} w, & \text{falls } |w| \leq k, \\ v, & \text{falls } |w| > k,\ |v| = k, \text{ und es gibt ein Wort } u \in V^* \text{ mit } w = uv. \end{cases}$$

Die Überführungsfunktion des Automaten ist definiert durch

$$\delta(z_w, x) = z_{[wx]_{m+1}}$$

für jedes Wort $w \in V^{\leq m+1}$ und jeden Buchstaben $x \in V$.

Wir ordnen die Zustände entsprechend der alphabetischen Ordnung der zugehörigen Spiegelwörter: Für je zwei Wörter $w_1 \in V^{\leq m+1}$ und $w_2 \in V^{\leq m+1}$ gilt

$$z_{w_1} \prec z_{w_2} \text{ genau dann, wenn } w_1^R \prec w_2^R$$

gilt. □

In der folgenden Abbildung ist der konstruierte Automat zu dem Beispiel $V = \{a, b\}$ und $L = \{ba\} \cup V^*\{aa, b\}$ zu sehen. Die dicke Linie überstreicht von links oben nach rechts unten die Zustände in geordneter Reihenfolge.

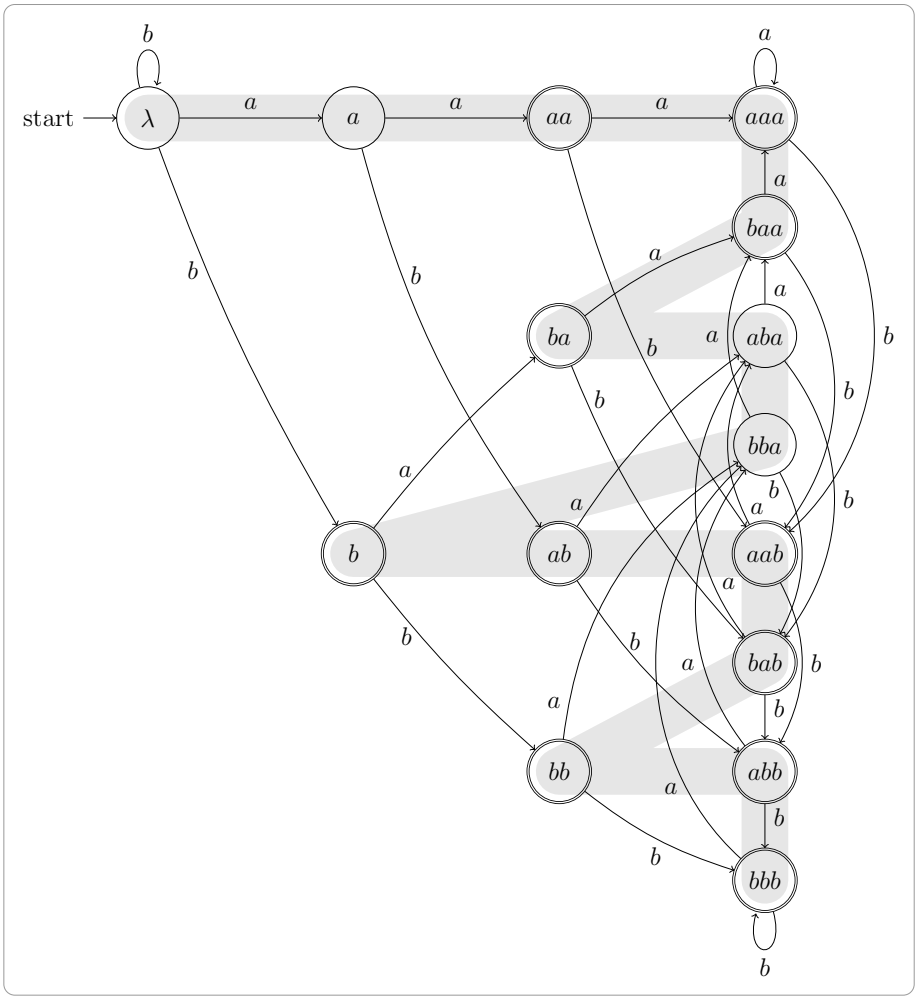
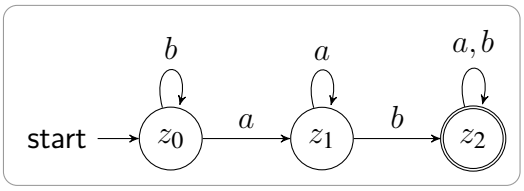


Abbildung 1: Automat, der die Sprache $\{ba\} \cup V^*\{aa, b\}$ akzeptiert

Satz 2.2 *Es gibt einen geordneten deterministischen endlichen Automaten, dessen akzeptierte Sprache nicht definit ist.*

Beweis. Die Sprache $L = \{a, b\}^*\{ab\}\{a, b\}^*$ ist nicht definit, wird aber von dem geordneten Automaten $\mathcal{A} = (\{a, b\}, \{z_0, z_1, z_2\}, z_0, \{z_2\}, \delta)$ akzeptiert, dessen Überführungsfunktion δ in der folgenden Abbildung dargestellt ist:

	z_0	z_1	z_2
a	z_1	z_1	z_2
b	z_0	z_2	z_2



□

Mit einer Art „Pumping-Argument“ kann die folgende Aussage bewiesen werden.

Satz 2.3 *Es gilt* $\mathit{RSUF} \subset \mathit{RPS}$.

Dies steht im Gegensatz zu dem nicht-regulären Fall.

Satz 2.4 *Die Sprachfamilien* SUF *und* PS *sind unvergleichlich.*

Vergleiche mit anderen Sprachfamilien liefern die folgende Hierarchie. Ein Pfeil von einem Eintrag X zu einem Eintrag Y stellt die echte Teilmengenbeziehung $X \subset Y$ dar. Wenn zwei Familien nicht durch einen gerichteten Pfad verbunden sind, dann sind sie unvergleichlich.

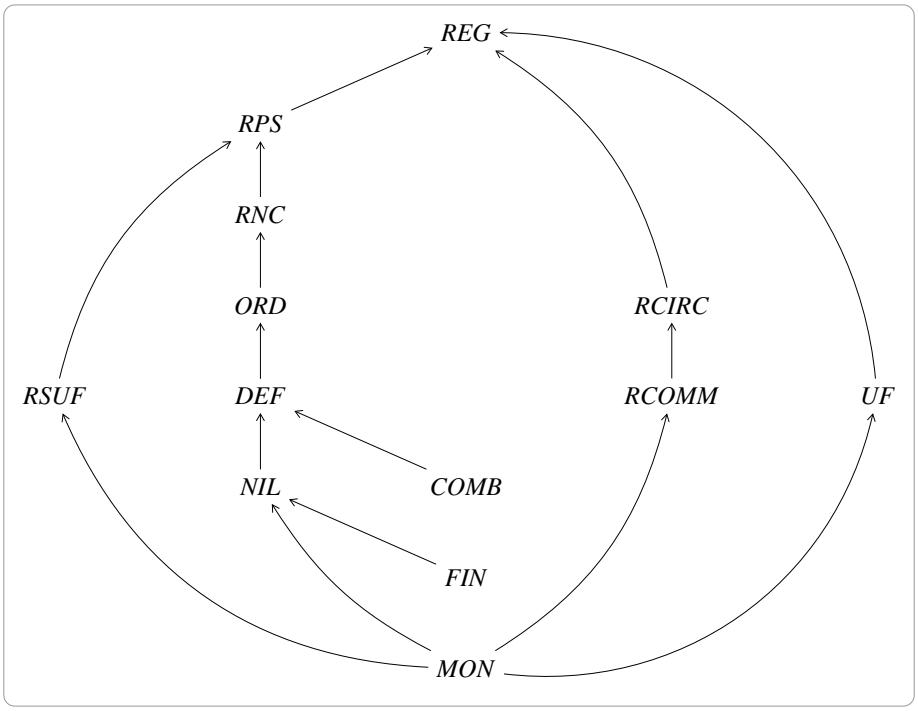


Abbildung 2: Hierarchie subregulärer Familien

Literatur

[1] M. Holzer, B. Truthe, On Relations Between Some Subregular Language Families. In: *Seventh Workshop on Non-Classical Models of Automata and Applications (NCMA), Porto, Portugal, August 31 – September 1, 2015, Proceedings*. books@ocg.at 318, Österreichische Computer Gesellschaft, Austria, 2015, 109–124.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 71–73.

Infinite and Bi-infinite Words with Decidable Monadic Theories

Dietrich Kuske[(A)] Jiamou Liu[(B)] Anastasia Moskvina[(B)]

[(A)]Technische Universität Ilmenau, Germany

[(B)]Auckland University of Technology, New Zealand

Abstract

We study word structures of the form $(D, \leq, P)$ where D is either $\mathbb{N}$ or $\mathbb{Z}$, $\leq$ is the natural linear ordering on D and $P \subseteq D$ is a predicate on D. In particular we show:

(a) The set of recursive ω-words with decidable monadic second order theories is Σ_3-complete.

(b) We characterise those sets $P \subseteq \mathbb{Z}$ that yield bi-infinite words $(\mathbb{Z}, \leq, P)$ with decidable monadic second order theories.

(c) We show that such "tame" predicates P exist in every Turing degree.

(d) We determine, for $P \subseteq \mathbb{Z}$, the number of predicates $Q \subseteq \mathbb{Z}$ such that $(\mathbb{Z}, \leq, P)$ and $(\mathbb{Z}, \leq, Q)$ are indistinguishable.

Through these results we demonstrate similarities and differences between logical properties of infinite and bi-infinite words.

The decision problem for logical theories of linear structures and their expansions has been an important question in theoretical computer science. Büchi in [1] proved that the monadic second order theory of the linear ordering $(\mathbb{N}, \leq)$ is decidable. Expanding the structure $(\mathbb{N}, \leq)$ by unary functions or binary relations typically leads to undecidable monadic theories. Hence many works have been focusing on structures of the form $(\mathbb{N}, \leq, P)$ where P is a unary predicate. Elgot and Rabin [4] showed that for many natural unary predicates P, such as the set of factorial numbers, the set of powers of k, and the set of kth powers (for fixed k), the structure $(\mathbb{N}, \leq, P)$ has decidable monadic second order theory; on the other hand, there are structures $(\mathbb{N}, \leq, P)$ whose monadic theory is undecidable [2]. Numerous subsequent works further expanded the field [9, 3, 8, 7, 6, 5].

1. Semenov generalised periodicity to a notion of "almost periodicity". While periodicity implies that certain patterns are repeated through a fixed period, almost periodicity captures the fact that certain patterns occur before the expiration of some period. This led him to consider "recurrent structures" within an infinite word. Such a recurrent structure is captured by a certain function, which he called "indicator of recurrence". In [8], he provided a full characterisation: $(\mathbb{N}, \leq, P)$ has decidable monadic theory if and only if P is recursive and there is a recursive indicator of recurrence for P.

2. Rabinovich and Thomas generalised periodicity to a notion of "uniform periodicity". Such a uniform periodicity condition is captured by a *homogeneous set* which exists by Ramsey's theorem. More precisely, a k-homogeneous set for $(\mathbb{N}, \leq, P)$ partitions the natural numbers into infinitely many finite segments that all have the same k-type. A uniformly homogeneous set specifies an ascending sequence of numbers that ultimately becomes k-homogeneous for any $k > 0$. In [6], Rabinovich and Thomas provided a full characterisation: $(\mathbb{N}, \leq, P)$ has a decidable monadic theory if and only if P is recursive and there is a recursive uniformly homogeneous set.

   Note that a recursive uniformly homogeneous set describes *how to* divide $(\mathbb{N}, \leq, P)$ such that the factors all have the same k-type. If P is recursive, this implies that the recurring k-type can be computed. A weakening of the existence of a recursive uniformly homogeneous set is therefore the requirement that one can compute a k-type such that $(\mathbb{N}, \leq, P)$ *can, in some way,* be divided. Nevertheless, Rabinovich and Thomas also showed that the monadic second order theory of $(\mathbb{N}, \leq, P)$ is decidable if and only if P is recursive and there is a "recursive type-function".

This paper has three general goals: The first is to compare these characterisations in some precise sense. The second is to investigate the above results in the context of *bi-infinite words*, which are structures of the form $(\mathbb{Z}, \leq, P)$. The third is to compare the logical properties of infinite words and bi-infinite words. More specifically, the paper discusses:

(a) We analyze the recursion-theoretical bound of the set of all computable predicates $P \subseteq \mathbb{N}$ where $(\mathbb{N}, \leq, P)$ has a decidable monadic theory. The second characterisation by Rabinovich and Thomas turns out to be a Σ_5-statement. In contrast, the characterisation by Semenov and the 1st characterisation by Rabinovich and Thomas both consist of Σ_3 statements, and hence deciding if a given $(\mathbb{N}, \leq, P)$ has decidable monadic theory is in Σ_3. We show that the problem is in fact Σ_3-complete. Hence these two characterisations are optimal in terms of their recursion-theoretical complexity.

(b) We then investigate which of the three characterisations can be lifted to bi-infinite words, i.e., structures of the form $(\mathbb{Z}, \leq, P)$ with $P \subseteq \mathbb{Z}$. It turns out that this is nicely possible for Semenov's characterisation and for the second characterisation by Rabinovich and Thomas, but not for their first one.

(c) If the monadic second order theory of $(\mathbb{N}, \leq, P)$ is decidable, then P is recursive. For bi-infinite words of the form $(\mathbb{Z}, \leq, P)$, this turns out not to be necessary. We actually show that every Turing degree contains a set $P \subseteq \mathbb{Z}$ such that the monadic second order theory of $(\mathbb{Z}, \leq, P)$ is decidable.

(d) Finally, we investigate how many bi-infinite words are indistinguishable from $(\mathbb{Z}, \leq, P)$. It turns out that this depends on the periodicity properties of P: if P is periodic, there are only finitely many equivalent bi-infinite words, if P is recurrent and non-periodic, there are $2^{\aleph_0}$ many, and if P is not recurrent, then there are $\aleph_0$ many.

References

[1] J. BÜCHI, On a decision method in restricted second order arithmetics. In: E. NAGEL, et al. (eds.), *Proc. Intern. Congress on Logic, Methodology and Philosophy of Science*. Stanford University Press, Stanford, 1962, 1–11.

[2] J. BÜCHI, L. LANDWEBER, Definability in the monadic second-order theory of successor. *Journal of Symbolic Logic* **34** (1969), 166–170.

[3] O. CARTON, W. THOMAS, The monadic theory of morphic infinite words and generalizations. *Information and Computation* **176** (2002) 1, 51–56.

[4] C. ELGOT, M. RABIN, Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *Journal of Symbolic Logic* **31** (1996) 2, 169–181.

[5] A. RABINOVICH, On decidability of monadic logic of order over the naturals extended by monadic predicates. *Information and Computation* **205** (2007), 870–889.

[6] A. RABINOVICH, W. THOMAS, Decidable Theories of the Ordering of Natural Numbers with Unary Predicates. In: *CSL'06*. Lecture Notes in Comp. Science vol. 4207, Springer, 2006, 562–574.

[7] A. SEMENOV, Decidability of monadic theories. In: *MFCS*. Lecture Notes in Computer Science vol. 176, 1984, 162–175.

[8] A. SEMENOV, Logical theories of one-place functions on the set of natural numbers. *Mathematics in the USSR – Izvestia* **22** (1984), 587–618.

[9] D. SIEFKES, Decidable extensions of monadic second order successor arithmetic. In: *Automatentheorie und formale Sprachen*. Bibliogr. Inst., Mannheim, 1969, 441–472.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 74–77.

Parameterized Prefix Distance Between Regular Languages

Martin Kutrib Katja Meckel Matthias Wendlandt

Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
{kutrib,meckel,matthias.wendlandt}@informatik.uni-giessen.de

Abstract

We investigate the parameterized prefix distance between regular languages. The prefix distance between words is extended to languages in such a way that the distances of all words up to length n to the mutual other language are summed up. Tight upper bounds for the distance between unary as well as non-unary regular languages are derived. It is shown that there are pairs of languages having a constant, degree k polynomial, and exponential distance. Moreover, for every constant and every polynomial, languages over a binary alphabet are constructed that have exactly that distance. From the density and census functions of regular languages the orders of possible distances between languages are derived and are shown to be decidable. [1]

1. Introduction

In this work, we investigate the parameterized prefix distance between regular languages. In [2] several notions of distances have been extended from distances between strings to distances between languages (see also [4]). To this end, a relative distance between a language L_1 and a language L_2 is defined to be the supremum of the minimal distances of all words from L_1 to L_2. The distance between L_1 and L_2 is defined as the maximum of their mutual relative distances. Since here we are interested in computations of faulty finite state devices that are still tolerable, we stick with the prefix distance and consider a parameterized extension. For words w_1 and w_2 the prefix distance sums up the number of all letters of w_1 and w_2 that do not belong to a common prefix of these words. One can suppose that on the common prefixes the computations of both machines are the same until a faulty component comes into play and the computations diverge. The parameterized prefix distance between languages sums up the distances of all words up to length n from one language to their closest words from the other language, and vice versa.

Since the distance between identical words should always be 0, for the distances between languages, the number of words in their symmetric difference plays a crucial role. In this

[1] This work was already published in SOFSEM 2014: Theory and Practice of Computer Science, LNCS, vol. 8327, pp. 419 – 430.

connection we utilize the density and census functions that count the number of words in a language. The study of densities of regular languages has a long history (see, for example, [1, 3, 5, 6, 7, 8]).

2. Preliminaries

In general, a *distance* over Σ^* is a function $d : \Sigma^* \times \Sigma^* \to \mathbb{N} \cup \{\infty\}$ satisfying, for all $x, y, z \in \Sigma^*$, the conditions $d(x, y) = 0$ if and only if $x = y$, $d(x, y) = d(y, x)$, and $d(x, y) \leq d(x, z) + d(z, y)$.

For example, words w_1 and w_2 over Σ^* can be compared by summing up the number of all letters of w_1 and w_2 that do not belong to a common prefix of these words. This so-called prefix distance $d_{\text{pref}} : \Sigma^* \times \Sigma^* \to \mathbb{N}$ between words is defined to be $d_{\text{pref}}(w_1, w_2) = |w_1| + |w_2| - 2\max\{\, |v| \mid w_1, w_2 \in v\Sigma^* \,\}$. Clearly, $d_{\text{pref}}(w_1, w_2) = 0$ if and only if $w_1 = w_2$, and $d_{\text{pref}}(w_1, w_2) = |w_1| + |w_2|$ if and only if the first letters of w_1 and w_2 are different. Moreover, the prefix distance between two words can be large if their length difference is large.

Distances over Σ are extended to distances between a word and a language by taking the minimum of the distances between the word and the words belonging to the language. For the prefix distance we obtain pref-$d : \Sigma^* \times 2^{\Sigma^*} \to \mathbb{N} \cup \{\infty\}$ which is defined to be

$$\text{pref-}d(w, L) = \begin{cases} \min\{\, d_{\text{pref}}(w, w') \mid w' \in L \,\} & \text{if } L \neq \emptyset \\ \infty & \text{otherwise} \end{cases}.$$

Clearly, pref-$d(w, L) = 0$ if $w \in L$.

The next step is to extend the distance between a word and a language to a distance between two languages $L_1, L_2 \subseteq \Sigma^*$. This can be done by taking the maximum of the suprema of the distances of all words from L_1 to L_2 and vice versa. However, here we are interested in a parameterized definition, where the distance additionally depends on the length of the words. So, the *parameterized prefix distance* between languages pref-$D : \mathbb{N} \times 2^{\Sigma^*} \times 2^{\Sigma^*} \to \mathbb{N} \cup \{\infty\}$ is defined by

$$\text{pref-}D(n, L_1, L_2) = \sum_{\substack{w \in L_1, \\ 0 \leq |w| \leq n}} \text{pref-}d(w, L_2) + \sum_{\substack{w \in L_2, \\ 0 \leq |w| \leq n}} \text{pref-}d(w, L_1).$$

In general, one cannot expect to obtain a convenient description of the parameterized prefix distance for *all* n. So, in the following, if not stated otherwise, it is understood that pref-$D(n, L_1, L_2) = f(n)$ means pref-$D(n, L_1, L_2) = f(n)$, for all n greater than some constant n_0.

3. Results

First, we investigate the range of possible parameterized distances between regular languages.

To determine the upper bound of the prefix distance between two languages $L_1, L_2 \subseteq \Sigma^*$ we consider some word $w \in L_1$ and the shortest word $s \in L_2$. In any case we have the inequation pref-$d(w, L_2) \leq |w| + |s|$ and, thus, the word w contributes in a maximal way to the distance if it does not have a common prefix with s. In this case, we have pref-$d(w, L_2) = |w| + |s|$. This observation leads to a general upper bound as follows.

Proposition 3.1 *Let $L_1, L_2 \subseteq \Sigma^*$ be two non-empty languages, $m_1 = \min\{\,|w| \mid w \in L_1\,\}$ and $m_2 = \min\{\,|w| \mid w \in L_2\,\}$ be the lengths of the shortest words of L_1 and L_2, respectively, and $m = \min\{m_1, m_2\}$, and $M = \max\{m_1, m_2\}$. Then*

$$\text{pref-}D(n, L_1, L_2) \leq \sum_{i=m}^{n} |\Sigma|^i \cdot (i + M).$$

Proposition 3.1 is the best possible, in the sense that there are worst case languages for which it is matched.

Proposition 3.2 *For any $M = m \geq 0$, there are binary regular languages $L_1, L_2 \subseteq \{a,b\}^*$ so that* pref-$D(n, L_1, L_2) = \sum_{i=m}^{n} |\Sigma|^i \cdot (i + M)$, *where m is the minimum and M is the maximum of the lengths of the shortest words in L_1 and L_2.*

So far, we considered languages over alphabets with at least two letters. For unary languages the situation changes significantly. An immediate observation is, that every two words have a distance to each other which is given by their length difference only.

Proposition 3.3 *Let $L_1 \subseteq \{a\}^*$ and $L_2 \subseteq \{a\}^*$ be two non-empty unary languages. Then* pref-$D(n, L_1, L_2) \leq \frac{n(n+1)}{2} + 1$.

As for the general case, the upper bound for the parameterized prefix distance of unary regular languages is tight. However, the witness languages $L_1 = aa^*$ and $L_2 = \{\lambda\}$ are the only ones whose distance meets the upper bound.

Proposition 3.4 *There are unary regular languages $L_1, L_2 \subseteq \{a\}^*$ so that their prefix distance is* pref-$D(n, L_1, L_2) = \frac{n(n+1)}{2} + 1$.

So far, we have explored the upper and lower bounds for parameterized prefix distances. Next, we are interested in the question which functions are possible to obtain by considering the prefix distance of two regular languages.

Proposition 3.5 *There are regular languages L_1 and L_2 even over a binary alphabet so that* pref-$D(n, L_1, L_2) \in \Theta(n2^n)$.

Proposition 3.6 *Let $c \geq 1$ be an integer. Then there are unary regular languages L_1 and L_2 so that* pref-$D(n, L_1, L_2) = c$, *for all $n \geq c$.*

Theorem 3.7 *Let $p(n) = x_k \cdot n^k + x_{k-1} \cdot n^{k-1} + \cdots + x_0$ be a polynomial of degree $k \geq 0$ with integer coefficients x_i, $0 \leq i \leq k$, and $x_k \geq 1$. Then two regular languages L_1 and L_2 over the alphabet $\{a,b\}$ can effectively be constructed so that* pref-$D(n, L_1, L_2) = p(n)$, *for all $n \geq n_0$, where n_0 is some constant.*

From a practical as well as from a theoretical point of view, it is interesting to decide the order of magnitude of the distance between regular languages. In the definition of the distances, the number of words in the symmetric difference of the languages plays a crucial role. Summing up the distance of each of these words gives the distance of two languages. So, the question

arises of how many words up to a certain length are in a given language. The function that counts the number of words of a fixed length n is called the *density function* (see, for example, [7, 8] and the references therein). The function that counts the number of words up to a given length n is called *census function*. Clearly, both are closely related.

The results in [7] for the density function imply a decision procedure for the question whether the census function of a regular language is polynomial or exponential, and for the former cases, whether it is of a certain degree.

Theorem 3.8 *Let A be a DFA. Then it is decidable whether* cens_A *is exponential or a polynomial. If it is a polynomial, the degree can be computed.*

Now we turn to the classes of parameterized prefix distances between regular languages. As mentioned before, for their computation the words in their symmetric difference are central, since only these contribute to the distance.

Theorem 3.9 *Let L_1 and L_2 be two regular languages. Then it is decidable whether the parameterized prefix distance* pref-$D(n, L_1, L_2)$ *is ultimately constant.*

Theorem 3.10 *Let L_1 and L_2 be two regular languages. Then it is decidable whether the parameterized prefix distance* pref-$D(n, L_1, L_2)$ *is exponential.*

Theorem 3.11 *Let L_1 and L_2 be two regular languages and $k \geq 1$ be a constant. Then it is decidable whether the parameterized prefix distance* pref-$D(n, L_1, L_2)$ *belongs to $\Omega(n^k) \cap O(n^{k+1})$.*

References

[1] J. BERSTEL, C. REUTENAUER, *Rational series and their languages*. EATCS monographs on theoretical computer science, Springer, 1988.

[2] C. CHOFFRUT, G. PIGHIZZINI, Distances between languages and reflexivity of relations. *Theoretical Computer Science* **286** (2002), 117 – 138.

[3] S. EILENBERG, *Automata, Languages, and Machines*. Academic Press, 1976.

[4] J. B. KRUSKAL, An overview of sequence comparison. In: *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, 1983.

[5] A. SALOMAA, M. SOITTOLA (eds.), *Automata-Theoretic Aspects of Formal Power Series*. Texts and monographs in computer science, Springer, 1978.

[6] M. SCHÜTZENBERGER, Finite counting automata. *Information and Control* **5** (1962), 91 – 107.

[7] A. SZILARD, S. YU, K. ZHANG, J. SHALLIT, Characterizing regular languages with polynomial densities. In: *Mathematical Foundations of Computer Science (MFCS 1992)*. LNCS 629, Springer, 1992, 494 – 503.

[8] S. YU, *Regular Languages*, Handbook of Formal Languages 1, chapter 2, Springer, Berlin, 1997, 41 – 110.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 78–80.

Characterizations of $+\omega$-picture languages

Parvaneh Babari[(A)] Nicole Schweikardt[(B)]

[(A)]Universität Leipzig
babari@informatik.uni-leipzig.de

[(B)]Humboldt Universität Berlin
schweikn@informatik.hu-berlin.de

Zusammenfassung

The theory of two-dimensional languages as a generalization of formal string languages was motivated by problems arising from image processing and pattern recognition [19], and also plays a role in the theory of cellular automata and other devices of parallel computing [18, 21]. In the nineties, Restivo and Giammarresi defined the family of recognizable finite picture languages [8, 10]. This family is very robust and has been characterized by many different devices. Several research groups obtained a description of recognizable picture languages in terms of automata, sets of tiles, rational operations, or existential monadic second-order logic [8, 9, 10, 11, 15, 16, 17]. Thomas [22], Finkel [7], Gnanasekaran, Dare, Subramanian and Siromoney [4, 12, 13] established considerable results over the family of infinite picture lan guages k nown as $\omega\omega$-picture languages. They can be considered as extensions of infinite string languages to two dimensions. One can consider $\omega\omega$-pictures as digitized images of symbols occupying a quadrant of a plane. The motivation for considering $\omega\omega$-pictures lies in the fact that pictures of functions can be considered as infinite digitized images [20]. In this paper, we introduce a new class of two-dimensional languages and we call them $+\omega$-picture languages. These are languages with properties of both families of finite and infinite picture languages. These new family can be also motivated by problems arising in digitization. In our setting, this term can be used to describe the scanning of analog sources, such as taped videos captured by objects like digital security cameras, into computers for editing or any other computer processing. We summarize our results as follows:

- We introduce a new class of two-dimensional languages called $+\omega$-picture languages. These are pictures with finite number of rows and infinite number of columns. Considering finitely many rows in our setting permits us to define two concatenation operations which involve languages of finite pictures and produce $+\omega$-picture languages.
- To characterize this family, we define tiling systems with a Büchi accepting condition. We introduce the notions of run of this device and also recognizability of $+\omega$-picture languages in terms of it. Indeed, when restricting to the one-dimensional case, this device determines a non-deterministic Büchi automaton $\mathcal{A}$ over the input alphabet

Σ. We apply Büci tiling systems to extend the well-known characterization of Büchi recognizable ω-languages found by Büchi to our setting. Due to infinity in two dimensions in the setting of $\omega\omega$-picture languages, establishing such a Büchi characterization is not possible.

- Motivated by the seminal Büchi-Elgot-Trakhtenbrot theorem [3, 6, 23] about the expressive equivalence of finite automata and monadic second-order (**MSO**) logic, we introduce an existential monadic second-order logic augmented with an additional existential quantifier that is applied necessarily to second-order variables for infinite sets. Applying such an additional quantifier permits us to express the Büchi acceptance condition of the system without the use of horizontal linear ordering, and as a result by means of some techniques and results from graph theory we show that the class of $+\omega$-picture languages recognized by tiling systems and the one defined by this new existential monadic second-order logic is equivalent.

In the ongoing research several related open problems arise. As the most important and interesting open problem, we can mention the generalization of these results to the quantitative setting. Recently, in [2], the equivalence of a quantitative automaton model over finite pictures and a fragment of a quantitative monadic second-order logic has been considered. An extension of these results to the setting of $+\omega$-picture languages could be interesting.

Literatur

[1] J-H. Altenbernd, W. Thomas, S. Wöhrle. Tiling systems over infinite pictures and their acceptance conditions, in: DLT 2002, in: Lecture Notes in Computer Science, Vol. 2450, 297-306, Springer (2002).

[2] P. Babari, M. Droste. A Nivat theorem for weighted picture automata and weighted MSO logics, in: LATA 2015, in: Lecture Notes in Computer Science, Vol. 8977, 703-715, Springer (2015).

[3] J. R. Büchi. Weak second order arithmetic and finite automata. Zeitschrift für Mathematische Logik und Grundlagen der Informatik, Vol. 6, 66-92 (1960).

[4] V. R. Dare, K. G. Subramanian, D. G. Thomas, R. Siromoney, Infinite arrays and recognizability. International Journal of Pattern Recognition and Artificial Intelligence, Vol. 14, 525-536 (2000).

[5] H. D. Ebbinghaus, J. Flum, W. Thomas. Mathematical logic. 2nd ed., Springer-Verlag/Berlin/Heidelberg/New York (1994).

[6] C. C. Elgot. Decision problems of finite automata design and related arithmetics. Transactions of The American Mathematical Society, Vol. 98, 21-51 (1961).

[7] O. Finkel. On recognizable languages of infinite pictures. International Journal of Foundations of Computer Science, Vol. 15(6), 823-840 (2004).

[8] D. Giammarresi, A. Restivo. Recognizable picture languages. International Journal of Pattern Recognition and Artificial Intelligence, Vol. 6(2, 3), 241-256 (1992).

[9] D. Giammarresi and A. Restivo. Two-dimensional finite state recognizability, Fundamental Informaticae, Vol. 25(3), 399-422 (1996).

[10] D. Giammarresi, A. Restivo. Two-dimensional languages, In G. Rozenberg and A. Salomaa, editors, Handbook of Formal Languages, vol.3, 215-267, Springer (1997).

[11] D. Giammarresi, A. Restivo, S. Seibert, W. Thomas. Monadic second-order logic over rectangular pictures and recognizability by tiling systems. Information and computation, Vol. 125(1), 32-45 (1996).

[12] S. Gnanasekaran, V. R. Dare. Infinite arrays and domino systems. Electronic Notes in Discrete Mathematics, Vol. 12, 349-359 (2003).

[13] S. Gnanasekaran, V. R. Dare. On recognizable infinite array languages. in: IWCIA 2004, in: Lecture Notes in Computer Science, Vol. 3322, 209-218 (2005).

[14] W. P. Hanf, Model-theoretic methods in the study of elementary logic, in: The Theory of Models (J. W. Addison, L. Henkin, and A. Tarski, Eds.), 132-145, North-Holland, Amsterdam (1965).

[15] K. Inoue, I. Takanami. A survey of two-dimentional automata theory. Information Sciences, Vol. 55, 99-121 (1991).

[16] K. Inoue, I. Takanami. A characterization of recognizable picture languages, in: ICPIA, in: Lecture Notes in Computer Science, Vol. 654, 133-143, Springer, Berlin (1992).

[17] M. Latteux, D. Simplot. Recognizable picture languages and domino tiling. Theoretical Computer Science, Vol. 178, 275-283 (1997).

[18] K. Lindgren, C. Moore, M. Nordahl. Complexity of two-dimensional patterns. Journal of Statistical Physics, Vol. 91(5-6), 909-951 (1998).

[19] M. Minski, S. Papert. Perceptron, M.I.T. Press, Cambridge, Mass (1969).

[20] A. Nakamura, H. Ono. Pictures of functions and their acceptability by automata. Theoretical Computer Science. Vol. 23, 37-48 (1983).

[21] R.A. Smith. Two-dimensional formal languages and pattern recognition by cellular automata. 12th IEEE FOCS Conference Record, 144-152 (1971).

[22] W. Thomas. On logics, tilings, and automata. In: 18th ICALP, in: Lecture Notes in Computer Science, Vol. 510, 441-453, Springer-Verlag (1991).

[23] B. Trakhtenbrot. Finite automata and logic of monadic predicates. Doklady Akademii Nauk SSSR, Vol. 140, 326-329 (1961).

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 81–84.

Contextual Array Grammars with Matrix and Regular Control

Henning Fernau[(A)] Rudolf Freund[(B)] Rani Siromoney[(C)]
K.G. Subramanian[(D)]

[(A)]Universität Trier, FB 4 – Abteilung Informatikwissenschaften, D-54296 Trier, Germany
fernau@uni-trier.de

[(B)]Technische Universität Wien, Institut für Computersprachen, A-1040 Wien, Austria
rudi@emcc.at

[(C)]Chennai Mathematical Institute, Kelambakkam 603103, India
siromoney@cmi.ac.in

[(D)]School of Computer Sciences, Universiti Sains Malaysia, 11800 Penang, Malaysia
kgsmani1948@yahoo.com

Abstract

We investigate the computational power of d-dimensional contextual array grammars with matrix control and regular control languages. For $d \geq 2$, d-dimensional contextual array grammars are less powerful than matrix contextual array grammars, which themselves are less powerful than contextual array grammars with regular control languages. Yet in the case of a one-letter alphabet, the family of 1-dimensional array languages generated by contextual array grammars with regular control languages coincides with the family of regular 1-dimensional array languages.

1. Introduction

Contextual string grammars were introduced by Solomon Marcus [5] with motivations arising from descriptive linguistics. A contextual string grammar consists of a finite set of strings (*axioms*) and a finite set of productions, which are pairs (s, c) where s is a string, the *selector,* and c is the *context*, i. e., a pair of strings, $c = (u, v)$, over the alphabet under consideration. Starting from an axiom, contexts iteratively are added as it is indicated by the productions, which yields new strings. In contrast to usual sequential string grammars in the Chomsky hierarchy, these contextual string grammars are pure grammars where new strings are not obtained by rewriting, but by adjoining strings. Regulated rewriting with different control mechanisms has been studied extensively especially for string grammars (e.g., see [1]), for example, grammars with control languages and matrix grammars, but then also for array grammars, e.g., see [2]. Non-isometric contextual array grammars (with regulation) were considered in [4].

2. Definitions

For notions and notations as well as results related to formal language theory, we refer to books like [7]. The families of λ-free (λ denotes the empty string) regular string languages (over a k-letter alphabet) is denoted by $\mathcal{L}(REG)$ ($\mathcal{L}\left(REG^k\right)$). For the definitions and notations for arrays and sequential array grammars we refer to [2, 6, 8].

Definition 2.1 *A regular* d-dimensional array grammar *is a quintuple* $G = (d, N, T, \#, P, S)$ *where* N *is the alphabet of non-terminal symbols,* T *is the alphabet of terminal symbols,* $N \cap T = \emptyset$, $\# \notin N \cup T$; P *is a finite non-empty set of regular* d*-dimensional array productions over* $N \cup T$, *and* S *is the start symbol. A regular* d*-dimensional array production either is of the form* $A \to b$, $A \in N$, $b \in T$, *or* $Av\# \to bC$, $A, C \in N$, $b \in T$, $v \in \mathbb{Z}^d$ *with* $\|v\| = 1$. *The application of* $A \to b$ *means replacing* a *by* b *in a given array.* $Av\# \to bC$ *can be applied if in the underlying array we find a position* u *occupied by* A *and a blank symbol at position* $u + v$; A *then is replaced by* b, *and* # *by* C. *The array language generated by* G *is the set of all* d*-dimensional arrays derivable from the initial array* $\{(\Omega_d, S)\}$.

The family of Λ*-free* d*-dimensional array languages (of equivalence classes) of arrays over a* k*-letter alphabet generated by regular* d*-dimensional array grammars is denoted by* $\mathcal{L}\left(d\text{-}REGA^k\right)$ *(*$\left[\mathcal{L}\left(d\text{-}REGA^k\right)\right]$*). For arbitrary alphabets, we omit the superscript* k.

For a string w, $[arr(w)]$ denotes the one-dimensional array image of w; conversely, $[str(\mathcal{A})]$ denotes the string image of the one-dimensional array $\mathcal{A}$.

Theorem 2.2 *For all* $k \geq 1$,

$$\left[\mathcal{L}\left(1\text{-}REGA^k\right)\right] = \left[arr\left(\mathcal{L}\left(REG^k\right)\right)\right] \text{ and } str\left(\left[\mathcal{L}\left(1\text{-}REGA^k\right)\right]\right) = \mathcal{L}\left(REG^k\right).$$

Definition 2.3 *A* d-dimensional contextual array grammar *(*$d \in \mathbb{N}$*) is a construct* $G = (d, V, \#, P, A)$ *where* V *is an alphabet not containing the blank symbol* #, A *is a finite set of axioms, i. e., of* d*-dimensional arrays in* V^{+d}, *and* P *is a finite set of rules of the form* $\left(U_\alpha, \alpha, U_\beta, \beta\right)$ *where* $U_\alpha, U_\beta \subseteq \mathbb{Z}^d$, $U_\alpha \cap U_\beta = \emptyset$, *and* U_α, U_β *are finite and non-empty, as well as* $\alpha : U_\alpha \to V$ *and* $\beta : U_\beta \to V$.

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

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

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

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

The special type of d-dimensional contextual array grammars where axioms are connected and rule applications preserve connectedness is denoted by $d\text{-}ContA$, the corresponding family of d-dimensional array languages by $\mathcal{L}(d\text{-}ContA)$; by $\mathcal{L}\left(d\text{-}ContA^k\right)$ we denote the corresponding family of d-dimensional array languages over a k-letter alphabet.

Example 2.4 *For the singleton language* $L_{\perp} = \left\{ \begin{array}{c} a \\ a \\ a\ a\ a\ a\ a \end{array} \right\} \subset \left[\{a\}^{+2}\right]$ *we have* $L_{\perp} \in [\mathcal{L}(2\text{-}ContA)] \setminus [\mathcal{L}(2\text{-}REGA)]$. □

Theorem 2.5 $\left[arr\left(\mathcal{L}\left(REG^1\right)\right)\right] = \left[\mathcal{L}\left(1\text{-}REGA^1\right)\right] \subseteq \left[\mathcal{L}\left(1\text{-}ContA^1\right)\right]$.

Proof. Any non-empty language $L \subseteq \{a\}^+$ in $\mathcal{L}\left(REG^1\right)$ is of the form $L = \bigcup_{i=1}^{n} \left\{a^{kn+d_i} \mid n \geq 0\right\}$ for some $n, k \in \mathbb{N}$ and $d_i \in \mathbb{N}$, $1 \leq i \leq n$. Then let $G(L) = (1, \{a\}, \#, P, A)$ be the 1-dimensional contextual array grammar with $A = \left\{arr\left(a^{d_i}\right) \mid 1 \leq i \leq n\right\}$ and $P = \left\{\boxed{a}a^k\right\}$. Obviously, $[L(G(L))] = [arr(L)]$. □

3. Controlled Contextual Array Grammars

Definition 3.1 *A d-dimensional matrix contextual array grammar is a pair $G_M = (G, M)$ where $G = (d, V, \#, P, A)$ is a d-dimensional contextual array grammar and M is a finite set of sequences, called* matrices*, of rules from P, i.e., each element of M is of the form $\langle p_1, \cdots, p_n \rangle$, $n \geq 1$, where $p_i \in P$ for $1 \leq i \leq n$. Derivations in a matrix contextual array grammar are defined as in a contextual array grammar except that a single derivation step now consists of the sequential application of the rules of one of the matrices in M, in the order in which the rules are given in the matrix. The array language generated by G_M is the set of all d-dimensional arrays which can be derived from any of the axioms in A. The family of d-dimensional array languages of arrays generated by d-dimensional matrix contextual array grammars (over a k-letter alphabet) is denoted by $\mathcal{L}(d\text{-}MContA)$ ($\mathcal{L}\left(d\text{-}MContA^k\right)$).*

Definition 3.2 *A d-dimensional contextual array grammar with regular control is a pair $G_C = (G, L)$ where $G = (d, V, \#, P, A)$ is a d-dimensional contextual array grammar and L is a regular string language over P. Derivations in a d-dimensional contextual array grammar with regular control are defined as in the contextual array grammar G except that in a successful derivation the sequence of applied rules has to be a word from L. The array language generated by G_C is the set of all d-dimensional arrays which can be derived from any of the axioms in A following a control word from L. The family of d-dimensional array languages of (equivalence classes of) arrays generated by d-dimensional contextual array grammars over a k-letter alphabet with regular control is denoted by $\mathcal{L}((d\text{-}ContA, REG))$ ($[\mathcal{L}((d\text{-}ContA, REG))]$).*

Example 3.3 *Consider the regular string language $L_r = \{ba^n b \mid n \geq 1\}$; we have $[arr(L_r)] \in \left[\mathcal{L}\left(1\text{-}REGA^2\right)\right] \setminus \left[\mathcal{L}\left(\left(1\text{-}ContA^2, REG\right)\right)\right]$. Yet if we take $L'_r = L_r \cup \{b\}$, then $[arr(L'_r)] \in \left[\mathcal{L}\left(\left(1\text{-}ContA^1, REG\right)\right)\right] \setminus \left[\mathcal{L}\left(1\text{-}MContA^2\right)\right]$: Consider $G''_r = (G'_r, C'_r)$ with $G'_r = (1, \{a, b\}, \#, P, \{arr(b)\})$ and $P = \{p_{ba}, p_{aa}, p_{ab}\}$ with $p_{ba} = \boxed{b}\, a$, $p_{aa} = \boxed{a}\, a$, and $p_{ab} = \boxed{a}\, b$ as well as $C'_r = \{p_{ba}\}\{p_{aa}\}^*\{p_{ab}\}$. Then $[L(G''_r)] = [arr(L'_r)]$.* □

Theorem 3.4 *For any $d \geq 1$ and any $k \geq 2$, we have:*

$$\left[\mathcal{L}\left(d\text{-}ContA^k\right)\right] \subsetneqq \left[\mathcal{L}\left(d\text{-}MContA^k\right)\right] \subsetneqq \left[\mathcal{L}\left(\left(d\text{-}ContA^k, REG\right)\right)\right].$$

Proof. The inclusions follow form the general results in [3]. The strictness of the first inclusion follows from taking the non-regular string language $L_n = \{a^n ba^n \mid n \geq 1\}$. Then $\left[i_{1,d}\left(arr\left(L_n\right)\right)\right] \in \left[\mathcal{L}\left(d\text{-}MContA^2\right)\right] \setminus \left[\mathcal{L}\left(d\text{-}ContA^k\right)\right]$. The strictness of the second inclusion follows from taking $\left[i_{1,d}\left(arr\left(L'_r\right)\right)\right]$. □

Theorem 3.5 *For any $d \geq 2$ and any $k \geq 2$, all three families $\left[\mathcal{L}\left(d\text{-}ContA^k\right)\right]$, $\left[\mathcal{L}\left(d\text{-}MContA^k\right)\right]$, and $\left[\mathcal{L}\left(\left(d\text{-}ContA^k, REG\right)\right)\right]$ are incomparable with $\left[\mathcal{L}\left(d\text{-}REGA^k\right)\right]$.*

Proof. For the singleton language $L_\perp$ from Example 2.4, we have $i_{2,d}\left(L_\perp\right) \in \left(\left[\mathcal{L}\left(d\text{-}ContA^1\right)\right] \cap \left[\mathcal{L}\left(d\text{-}MContA^1\right)\right] \cap \left[\mathcal{L}\left(\left(d\text{-}ContA^1, REG\right)\right)\right]\right) \setminus \left[\mathcal{L}\left(d\text{-}REGA^1\right)\right]$. On the other hand, for $L_r = \{ba^n b \mid n \geq 1\}$ we have $i_{2,d}\left(\left[arr\left(L_r\right)\right]\right) \in \left[\mathcal{L}\left(1\text{-}REGA^2\right)\right] \setminus \left(\left[\mathcal{L}\left(d\text{-}ContA^1\right)\right] \cup \left[\mathcal{L}\left(d\text{-}MContA^1\right)\right] \cup \left[\mathcal{L}\left(\left(d\text{-}ContA^1, REG\right)\right)\right]\right)$. □

Theorem 3.6
$\left[\mathcal{L}\left(1\text{-}REGA^1\right)\right] = \left[\mathcal{L}\left(\left(1\text{-}ContA^1, REG\right)\right)\right] = \left[\mathcal{L}\left(1\text{-}MContA^1\right)\right] = \left[\mathcal{L}\left(1\text{-}ContA^1\right)\right]$.

References

[1] J. DASSOW, GH. PĂUN, *Regulated Rewriting in Formal Language Theory*. EATCS Monographs in Theoretical Computer Science 18, Springer, 1989.

[2] R. FREUND, M. KOGLER, M. OSWALD, Control mechanisms on #−context-free array grammars. In: *Mathematical Aspects of Natural and Formal Languages*. World Scientific Publ., Singapore, 1994, 97–136.

[3] R. FREUND, M. KOGLER, M. OSWALD, A general framework for regulated rewriting based on the applicability of rules. In: *Computation, Cooperation, and Life - Essays Dedicated to Gheorghe Păun on the Occasion of His 60th Birthday*. Lecture Notes in Computer Science 6610, Springer, 2011, 35–53.

[4] K. KRITHIVASAN, M. BALAN, R. RAMA, Array contextual grammars. In: *Recent Topics in Mathematical and Computational Linguistics*. Editura Academiei Române, Bucureşti, 2000, 154–168.

[5] S. MARCUS, Contextual grammars. *Rev. Roum. Math. Pures Appl.* **14** (1969), 1525–1534.

[6] A. ROSENFELD, *Picture Languages*. Academic Press, Reading, MA, 1979.

[7] G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages*. 1-3, Springer, 1997.

[8] P.-P. WANG, Some new results on isotonic array grammars. *Information Processing Letters* **10** (1980), 129–131.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 85–88.

Scanning Pictures The Boustrophedon Way

Henning Fernau[(A)] Meenakshi Paramasivan[(A)] Markus L. Schmid[(A)]
D. Gnanaraj Thomas[(B)]

[(A)]Fachbereich 4 – Abteilung Informatik, Universität Trier, D-54286 Trier, Germany
{Fernau,Paramasivan,MSchmid}@uni-trier.de

[(B)]Department of Mathematics, Madras Christian College, Chennai - 600059, India
dgthomasmcc@yahoo.com

Abstract

We introduce and discuss finite automata working on rectangular-shaped arrays (i. e., pictures) in a boustrophedon reading mode, called BFAs. We prove close relationships with the well-established class of regular matrix (picture) languages (RML). We derive several combinatorial, algebraic and decidability results for BFAs. For instance, we show pumping and interchange lemmas, while the non-emptiness problem for BFAs turns out to be NP-complete.

1. Introduction and Definitions

Syntactic considerations of digital images have a tradition of about five decades. They should (somehow) reflect methods applied to picture processing. However, one of the basic methods of scanning pictures in practice have not been thoroughly investigated from a more theoretical point of view: that of using space-filling curves [3, 5]. Here, we start such an investigation with what can be considered as the most simple way of defining space-filling curves: scanning line after line of an image, alternating the direction of movement every time when the image boundary is encountered. We consider finite automata that work this way. We show that they are (essentially) equivalent to regular matrix languages (RMLs) as introduced in a sequence of papers of Rani Siromoney and her co-authors already in the early 1970s. These two-dimensional picture languages have connection with generation of kolam patterns [4, 6]. Possibly surprisingly enough, we also present quite a number of new results for this class of picture languages, including a discussion of natural decidability questions lacking so far. This shows tighter connections between finite automata working on pictures and array grammars of different types; for overviews on this topic, we refer to [1, 2]. Let us briefly recall the standard definitions and notations regarding one- and two-dimensional words and languages.

Let $\mathbb{N} := \{1, 2, 3, \ldots\}$ be the set of natural numbers.

The notation $|w|$ stands for the length of a string w. A *two-dimensional word* (also called *picture*, *matrix* or an *array*) *over* Σ is a tuple

$$W := ((a_{1,1}, a_{1,2}, \ldots, a_{1,n}), (a_{2,1}, a_{2,2}, \ldots, a_{2,n}), \ldots, (a_{m,1}, a_{m,2}, \ldots, a_{m,n})),$$

where $m, n \in \mathbb{N}$ and, for every i, $1 \leq i \leq m$, and j, $1 \leq j \leq n$, $a_{i,j} \in \Sigma$. We define the *number of columns* (or *width*) and *number of rows* (or *height*) of W by $|W|_c := n$ and $|W|_r := m$, respectively.

For the sake of convenience, we also denote W by $[a_{i,j}]_{m,n}$ or by a matrix in a more pictorial form. If we want to refer to the j^{th} symbol in row i of the picture W, then we use $W[i,j] = a_{i,j}$. By Σ^{++}, we denote the set of all (non-empty) pictures over Σ. Every subset $L \subseteq \Sigma^{++}$ is a *picture language*. $L' = \Sigma^{++} - L$ is the *complement* of the picture language L.

Let $W := [a_{i,j}]_{m,n}$ and $W' := [b_{i,j}]_{m',n'}$ be two non-empty pictures over Σ. The *column concatenation* of W and W', denoted by $W \oplus W'$, is undefined if $m \neq m'$. The *row concatenation* of W and W', denoted by $W \ominus W'$, is undefined if $n \neq n'$.

1.1. Boustrophedon Finite Automaton

Definition 1.1 *A* boustrophedon finite automaton*, or BFA for short, can be specified as a 7-tuple $M = (Q, \Sigma, R, s, F, \#, \Box)$, where Q is a finite set of states, Σ is an input alphabet, $R \subseteq Q \times (\Sigma \cup \{\#\}) \times Q$ is a finite set of rules. A rule $(q, a, p) \in R$ is usually written as $qa \to p$. The special symbol $\# \notin \Sigma$ indicates the border of the rectangular picture that is processed, $s \in Q$ is the initial state, F is the set of final states.*

We are now going to discuss the notions of configurations, valid configurations and an according configuration transition to formalize the work of BFAs, based on snapshots of their work.

Let $\Box$ be a new symbol indicating an erased *position and let $\Sigma_+ := \Sigma \cup \{\#, \Box\}$. Then $C_M := Q \times \Sigma_+^{++} \times \mathbb{N}$ is the set of configurations of M. A configuration $(p, A, \mu) \in C_M$ is* valid *if $1 \leq \mu \leq |A|_r$ and, for every i, $1 \leq i \leq \mu - 1$, the i^{th} row equals $\#\Box^{|A|_c - 2}\#$, for every j, $\mu + 1 \leq j \leq |A|_r$, the j^{th} row equals $\#w\#$, $w \in \Sigma^{|A|_c - 2}$, and, for some ν, $0 \leq \nu \leq |A|_c - 2$, $w \in \Sigma^{|A|_c - \nu - 2}$, the μ^{th} row equals $\#\Box^{\nu} w\#$, if μ is odd and $\#w\Box^{\nu}\#$, if μ is even. Notice that valid configurations model the idea of observable snapshots of the work of the BFA.*

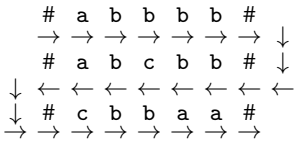
- *If (p, A, μ) and (q, A', μ) are two valid configurations such that A and A' are identical but for one position (i, j), where $A'[i, j] = \Box$ while $A[i, j] \in \Sigma$, then $(p, A, \mu) \vdash_M (q, A', \mu)$ if $pA[i, j] \to q \in R$.*
- *If (p, A, μ) and $(q, A, \mu + 1)$ are two valid configurations, then $(p, A, \mu) \vdash_M (q, A, \mu + 1)$ if the μ^{th} row contains only # and $\Box$ symbols, and if $p\# \to q \in R$.*

The reflexive transitive closure of the relation $\vdash_M$ is denoted by $\vdash_M^*$. The BFA M is deterministic, or a BDFA for short, if for all $p \in Q$ and $a \in \Sigma \cup \{\#\}$, there is at most one $q \in Q$ with $pa \to q \in R$.The language $L(M)$ accepted by M is then the set of all $m \times n$ pictures A over Σ such that

$$(s, \#_m \oplus A \oplus \#_m, 1) \vdash_M^* (f, \#_m \oplus \Box_m^n \oplus \#_m, m)$$

for some $f \in F$.

Note that the automaton works on a picture with a first and last column of only # symbols, but only the part in between these border columns is accepted. The following illustrates how a BFA scans some input picture and how a picture of a valid configuration looks like; it can be seen that the sequence of $\Box$ only indicate how far the input has been processed:



```
# a b b b b #
→ → → → → → ↓
# a b c b b # ↓
↓ ← ← ← ← ← ← ←
↓ # c b b a a #
→ → → → → → → →
```

```
# □ □ □ b b #
# a b c b b #
# c b b a a #
```

Example 1.2 *The* set K of tokens L *of all sizes and of all proportions, is accepted by the BFA* $M = (Q, \Sigma, R, s, F, \#, \square)$*, where* $Q = \{s, s_1, s_2, s_3, s_4\}$, $\Sigma = \{\mathtt{x}, \cdot\}$, $R = \{s\mathtt{x} \to s_1, s_1\cdot \to s_1, s_1\# \to s_2, s_1\# \to s_4, s_2\cdot \to s_2, s_2\mathtt{x} \to s_3, s_3\# \to s, s_3\# \to s_4, s_4\mathtt{x} \to s_4\}$*, and* $F = \{s_4\}$.

Here and in the following, if X is some mechanism (automata or grammars) for describing pictures, then $\mathcal{L}_\Sigma(X)$ is the family of non-empty picture languages over the alphabet Σ that can be described by X. If the alphabet does not matter, we can omit the subscript Σ.

A rotation by 90 degrees is also known as a *quarter turn*. The corresponding operator Q can be also applied (picture-wise) to picture languages and (language-wise) to families of picture languages.

2. Characterization Results

Theorem 2.1 *Let* Σ *be some alphabet. Then,* $\mathcal{L}_\Sigma(\mathrm{BDFA}) = \mathcal{L}_\Sigma(\mathrm{BFA})$.

Theorem 2.2 *Let* Σ *be some alphabet. Then,* $\mathcal{L}_\Sigma(\mathrm{BFA}) = \mathcal{L}_\Sigma(\mathrm{RFA})$.

Theorem 2.3 *Let* Σ *be some alphabet. Then,* $\mathcal{L}_\Sigma(\mathrm{BFA}) = Q(\mathcal{L}_\Sigma(\mathrm{RMG}))$.

3. Pumping and Interchange Lemmas

Lemma 3.1 *Let* M *be a BFA. Then there exists an* $n \in \mathbb{N}$*, such that, for every* $W \in L(M)$ *with* $|W|_r \geq n$*,* $W = X \ominus Y \ominus Z$*,* $|X \ominus Y|_r \leq n$*,* $|Y|_r \geq 1$ *and, for every* $k \geq 0$*,* $X \ominus Y_k \ominus Z \in L(M)$.

Lemma 3.2 *Let* M *be a BFA and let* $W \in L(M)$ *with* $|W|_r = m$*. Then there exist* $n, r_1, r_2, \ldots, r_m \in \mathbb{N}$*, such that, for every* $W \in L(M)$ *with* $|W|_c \geq n$*,*

$$W = (x_1 \oplus y_1 \oplus z_1) \ominus (x_2 \oplus y_2 \oplus z_2) \ominus \ldots \ominus (x_m \oplus y_m \oplus z_m)\,,$$

$|x_i \ominus y_i|_c \leq n$*,* $|y_i|_c \geq 1$*,* $1 \leq i \leq m$*, and, for every* $k \geq 1$*,*

$$W = (x_1 \oplus y_1^{(kt_1)} \oplus z_1) \ominus (x_2 \oplus y_2^{(kt_2)} \oplus z_2) \ominus \ldots \ominus (x_m \oplus y_m^{(kt_m)} \oplus z_m) \in L(M)\,,$$

where $t_i = \frac{lcm(r_1, r_2, \ldots, r_m)}{r_i}$*,* $1 \leq i \leq m$.

While the vertical pumping lemma has the nice property that a whole row-factor can be pumped, in the horizontal pumping lemma we can only pump factors of each individual row, that are independent from each other. As a result, this lemma does not guarantee the possibility of pumping by 0, i. e., removing a factor, which, for classical regular languages, often constitutes a particularly elegant way of showing the non-regularity of a language. However, it can be shown that also for BFA there exists a horizontal pumping lemma that pumps whole column-factors (which then also translates into a vertical pumping lemma for RML that pumps whole row-factors).

Lemma 3.3 *Let M be a BFA and let $m \in \mathbb{N}$. Then there exists an $n \in \mathbb{N}$, such that, for every $W \in L(M)$ with $|W|_r \leq m$ and $|W|_c \geq n$, $W = X \oplus Y \oplus Z$, $|X \oplus Y|_c \leq n$, $|Y|_c \geq 1$ and, for every $k \geq 0$, $X \oplus Y^k \oplus Z \in L(M)$.*

We wish to point out that in a similar way, we can also prove a row and a column interchange lemma (the only difference is that the number n has to be chosen large enough to enforce repeating pairs of states):

Lemma 3.4 *Let M be a BFA. Then there exists an $n \in \mathbb{N}$, such that, for every $W \in L(M)$ with $|W|_r \geq n$, there exists a factorization $W = V_1 \ominus X \ominus V_2 \ominus Y \ominus V_3$, $|X|_c \geq 1$, $|Y|_c \geq 1$, such that $V_1 \ominus Y \ominus V_2 \ominus X \ominus V_3 \in L(M)$.*

Lemma 3.5 *Let M be a BFA and let $m \in \mathbb{N}$. Then there exists an $n \in \mathbb{N}$, such that, for every $W \in L(M)$ with $|W|_r \leq m$ and $|W|_c \geq n$, there exists a factorization $W = V_1 \oplus X \oplus V_2 \oplus Y \oplus V_3$, $|X|_c \geq 1$, $|Y|_c \geq 1$, such that $V_1 \oplus Y \oplus V_2 \oplus X \oplus V_3 \in L(M)$.*

4. Complexity Results

Theorem 4.1 *The universal membership problem for BFAs is NL-complete.*

Theorem 4.2 *The non-emptiness problem for B(D)FAs is NP-complete, even for unary input alphabets.*

Theorem 4.3 *The inequivalence problem for BDFAs is NP-complete.*

References

[1] K. INOUE, I. TAKANAMI, A survey of two-dimensional automata theory. *Information Sciences* **55** (1991) 1-3, 99–121.

[2] J. KARI, V. SALO, A Survey on Picture-Walking Automata. In: W. KUICH, G. RAHONIS (eds.), *Algebraic Foundations in Computer Science - Essays Dedicated to Symeon Bozapalidis on the Occasion of His Retirement*. LNCS 7020, Springer, 2011, 183–213.

[3] H. SAGAN, *Space-Filling Curves*. Springer, 1994.

[4] G. SIROMONEY, R. SIROMONEY, K. KRITHIVASAN, Array grammars and kolam. *Computer Graphiccs and Image Processing* **3** (1974), 63–82.

[5] R. SIROMONEY, K. G. SUBRAMANIAN, Space-filling curves and infinite graphs. In: H. EHRIG, M. NAGL, G. ROZENBERG (eds.), *Graph grammars and their application to computer science*. LNCS 153, 1983, 380–391.

[6] K. YANAGISAWA, S. NAGATA, Fundamental Study on Design System of Kolam Pattern. *Forma* **22** (2007), 31–46.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 89–91.

Emptiness for Context-free Picture Languages is undecidable

Klaus Reinhardt[(A)]

[(A)]Universität Halle, Institut für Informatik
klaus.reinhardt@informatik.uni-halle.de

Zusammenfassung

A two-dimensional Kolam grammar as defined by Siromoney et al. in 1972 and independently by Matz in 1997 and Schlesinger in 1989 allows context-free rules of the form $A \to a$, $A \to BC$, $A \to \begin{smallmatrix} B \\ C \end{smallmatrix}$, and $S \to \lambda$ which can concatenate sub-pictures produced by B and C horizontally respectively vertically if their height respectively width fits. It was shown by Prusa in [2] that emptiness for the three-dimensional version is undecidable. We show this already for the two-dimensional version.

1. Definitions

A *picture* over an alphabet Σ is a two-dimensional array of elements of Σ. The set of pictures of size (m,n) is denoted by $\Sigma^{m,n}$. For examle $a^{m,n} \in \Sigma^{m,n}$ is a picture of width m and height n consisting only of a's. The empty picture λ is in $\Sigma^{m,0}$ and in $\Sigma^{0,n}$ for all $m,n \in \mathbb{N}$. A *picture language* is a subset of $\Sigma^{*,*} := \bigcup_{m,n\geq 0} \Sigma^{m,n}$. The *horizontal concatenation* of two picture languages $L_1, L_2 \in \Sigma^{*,*}$ is $L_1 L_2 := \{p \in \Sigma^{m+m',n} \mid \exists m, m', n \in \mathbb{N}, q \in L_1 \cap \Sigma^{m,n}, r \in L_2 \cap \Sigma^{m',n}\ \forall i \leq m, j \leq n\ q_{i,j} = p_{i,j}\ \wedge\ \forall i \leq m', j \leq n\ r_{i,j} = p_{i+m-1,j}\}$. Analogously, the *vertical concatenation* of two picture languages $L_1, L_2 \in \Sigma^{*,*}$ is $\begin{smallmatrix} L_1 \\ L_2 \end{smallmatrix} := \{p \in \Sigma^{m,n+n'} \mid \exists m, n, n' \in \mathbb{N}, q \in L_1 \cap \Sigma^{m,n}, r \in L_2 \cap \Sigma^{m,n'}\ \forall i \leq m, j \leq n\ q_{i,j} = p_{i,j}\ \wedge\ \forall i \leq m, j \leq n'\ r_{i,j} = p_{i,j+n-1}\}$.

A *two-dimensional Kolam grammar* (2KG) is a tuple $\mathcal{G} = (N, T, \mathcal{P}, S)$, where $S \in N$ is the initial nonterminal and $\mathcal{P}$ is a finite set of productions of the form $A \to a$, $A \to BC$, $A \to \begin{smallmatrix} B \\ C \end{smallmatrix}$, or $S \to \lambda$ with $A, B, C \in N$ and $a \in T$.

The produced picture language is $L(\mathcal{G}) = L(\mathcal{G}, S)$ where the picture languages $L(\mathcal{G}, A)$ for all $A \in N$ are the smallest sets with the following properties:

- If $A \to a$ is a production in $\mathcal{P}$ then $a \in L(\mathcal{G}, A)$,
- if $S \to \lambda$ is in $\mathcal{P}$ then $\lambda \in L(\mathcal{G}, S)$,
- if $A \to BC$ is in $\mathcal{P}$, then $L(G, A) \supseteq L(\mathcal{G}, B) L(\mathcal{G}, C)$, and

- if $A \to \begin{matrix} B \\ C \end{matrix}$ is in $\mathcal{P}$, then $L(\mathcal{G},A) \supseteq \begin{matrix} L(\mathcal{G},B) \\ L(\mathcal{G},C) \end{matrix}$.

Examples: The productions $R \to a,\ R \to RR$ generate the picture language $L(\mathcal{G},R) = a^{*,1}$ of rows. The productions $C \to a,\ C \to \begin{matrix} C \\ C \end{matrix}$ generate the picture language $L(\mathcal{G},C) = a^{1,*}$ of colulmns. Together with the productions $Q \to a,\ Q \to \begin{matrix} U \\ R \end{matrix},\ U \to QC$, the picture language $L(\mathcal{G},Q) = \{a^{n,n} \mid n \in \mathbb{N}\}$ of squares is produced, where $L(\mathcal{G},U) = \{a^{n+1,n} \mid n \in \mathbb{N}\}$.

2. Emptiness for 2KG is undecidable

Proof. We reduce the undecidable halting problem for a 2-counter Minski machine [1] to the emptiness problem for a 2KG. Depending on the state, such a machine can either increment one of the counters and change the state or decrement one of the counters and change the state if possible (not zero) and change to a different state otherwise.

Given such a 2-counter Minski machine M with n states and w.l.o.g. one halting state q_n, we will construct a 2KG

$$\mathcal{G} = (\{A_i,B_i,C_i,D_i,E_i,F_i,G_i,Q^{j,k},M^{j,k},N^{j,k},Q,C,R \mid i \le n,j,k \le 8\},\{a\},P,A_n)$$

such that by induction on the steps of M the following holds:

The counter machine M can reach the configuration with state q_i and the counter values $c,d \in \mathbb{N}$ if and only if there is an $e \in \mathbb{N}$ with the sqare picture $a^{2^e3^c5^d,2^e3^c5^d} \in L(\mathcal{G},A_i)$; furthermore $L(\mathcal{G},A_i)$ consists only of square pictures of this form. (The extra exponent e allows a growing number even if counter values decrease.) Thus $L(\mathcal{G}) = L(\mathcal{G},A_n)$ is non empty if and only is M halts.

We start the induction by the rule $A_0 \to a \in P$ for the initial state q_0 of M placing $a = a^{2^03^05^0,2^03^05^0} \in L(\mathcal{G},A_0)$ for the initial configuration with state q_0 and both counters empty.

In the following, we consider the induction step for all six possible types of transitions of M and, at the same time, mention all other productions in P. It does not matter if M is deterministic or nondeterministic.

1. If M increments the first counter going from q_i to q_j then the productions $B_i \to A_iQ^{1,2}$ and $Q^{1,2} \to QQ$ allow $a^{2^e3^c5^d,2^e3^{c+1}5^d} \in L(\mathcal{G},B_i)$ for $a^{2^e3^c5^d,2^e3^c5^d} \in L(\mathcal{G},A_i)$. The variable Q as from Example 3 produces squares which both must have the same height thus the width is multiplied by 3. Then the productions $A_j \to \begin{matrix} B_i \\ Q^{2,3} \end{matrix}$ and $Q^{2,3} \to \begin{matrix} Q^{1,3} \\ Q^{1,3} \end{matrix}$ and $Q^{1,3} \to Q^{1,2}Q$ allow $a^{2^e3^{c+1}5^d,2^e3^{c+1}5^d} \in L(\mathcal{G},A_j)$ which completes the induction step to the following configuration in this case.

2. If M increments the second counter going from q_i to q_j then the productions $C_i \to A_iQ^{1,4}$ and $Q^{1,4} \to Q^{1,2}Q^{1,2}$ allow $a^{2^e3^c5^d,2^e3^c5^{d+1}} \in L(\mathcal{G},C_i)$ for $a^{2^e3^c5^d,2^e3^c5^d} \in L(\mathcal{G},A_i)$. Here the width is multiplied by 5. Then the productions $A_j \to \begin{matrix} C_i \\ Q^{4,5} \end{matrix}$ and $Q^{4,5} \to \begin{matrix} Q^{2,5} \\ Q^{2,5} \end{matrix}$ and $Q^{2,5} \to Q^{2,3}Q$ allow $a^{2^e3^c5^{d+1},2^e3^c5^{d+1}} \in L(\mathcal{G},A_j)$ analogously.

3. If M decrements the first counter going from q_i to q_j then the productions $D_i \to A_i Q^{3,1}$ and $Q^{3,1} \to \begin{smallmatrix} Q^{2,1} \\ Q \end{smallmatrix}$ and $Q^{2,1} \to \begin{smallmatrix} Q \\ Q \end{smallmatrix}$ allow $a^{2^e 3^c 5^d, 2^{e+2} 3^{c-1} 5^d} \in L(\mathcal{G}, D_i)$ for $a^{2^e 3^c 5^d, 2^e 3^c 5^d} \in L(\mathcal{G}, A_i)$. Here the width is multiplied by 4/3 (by adding 1/3 of the height), which is possible if and only if the first counter is not zero. Then the productions $A_j \to \begin{smallmatrix} D_i \\ Q^{1,4} \end{smallmatrix}$ and $Q^{1,4} \to Q^{1,3} Q$ allow $a^{2^{e+2} 3^{c-1} 5^d, 2^{e+2} 3^{c-1} 5^d} \in L(\mathcal{G}, A_j)$.

4. If M decrements the second counter going from q_i to q_j then the productions $E_i \to A_i Q^{5,3}$ and $Q^{5,3} \to \begin{smallmatrix} Q^{2,3} \\ Q \end{smallmatrix}$ allow $a^{2^e 3^c 5^d, 2^{e+3} 3^c 5^{d-1}} \in L(\mathcal{G}, E_i)$ for $a^{2^e 3^c 5^d, 2^e 3^c 5^d} \in L(\mathcal{G}, A_i)$. Here the width is multiplied by 8/5, which is possible if and only if the second counter is not zero. Then the productions $A_j \to \begin{smallmatrix} E_i \\ Q^{3,8} \end{smallmatrix}$ and $Q^{3,8} \to Q^{3,4} Q^{3,4}$ and $Q^{3,4} \to Q^{3,1} Q$ allow $a^{2^{e+3} 3^c 5^{d-1}, 2^{e+3} 3^c 5^{d-1}} \in L(\mathcal{G}, A_j)$ analogously.

5. If M zero-tests the first counter going from q_i to q_j then the productions $F_i \to A_i N^{3,1}$ and $F_i \to A_i N^{3,2}$ and $N^{3,2} \to \begin{smallmatrix} M^{3,2} \\ R \end{smallmatrix}$ and $M^{3,2} \to N^{3,1} C$ and $N^{3,1} \to \begin{smallmatrix} M^{3,1} \\ R \end{smallmatrix}$ and $M^{3,1} \to Q^{3,3} C$ and $Q^{3,3} \to \begin{smallmatrix} Q^{1,3} \\ Q^{2,3} \end{smallmatrix}$ allow $a^{2^e 3^0 5^d, 2^{e+1} 3^0 5^d} \in L(\mathcal{G}, F_i)$ for $a^{2^e 3^0 5^d, 2^e 3^0 5^d} \in L(\mathcal{G}, A_i)$. Here the width is multiplied by 2 and the first counter is zero which corresponds to width and height of a picture in $L(\mathcal{G}, N^{3,1})$ respectively $L(\mathcal{G}, N^{3,2})$ being congruent 1 respectively 2 modulo 3. Then the production $A_j \to \begin{smallmatrix} F_i \\ Q^{1,2} \end{smallmatrix}$ allows $a^{2^{e+1} 3^0 5^d, 2^{e+1} 3^0 5^d} \in L(\mathcal{G}, A_j)$.

6. If M zero-tests the second counter going from q_i to q_j then the productions $G_i \to A_i N^{5,1}$ and $G_i \to A_i N^{5,2}$ and $G_i \to A_i N^{5,3}$ and $G_i \to A_i N^{5,4}$ and $N^{5,4} \to \begin{smallmatrix} M^{5,4} \\ R \end{smallmatrix}$ and $M^{5,4} \to N^{5,3} C$ and $N^{5,3} \to \begin{smallmatrix} M^{5,3} \\ R \end{smallmatrix}$ and $M^{5,3} \to N^{5,2} C$ and $N^{5,2} \to \begin{smallmatrix} M^{5,2} \\ R \end{smallmatrix}$ and $M^{5,2} \to N^{5,1} C$ and $N^{5,1} \to \begin{smallmatrix} M^{5,1} \\ R \end{smallmatrix}$ and $M^{5,1} \to Q^{5,5} C$ and $Q^{5,5} \to \begin{smallmatrix} Q^{1,5} \\ Q^{4,5} \end{smallmatrix}$ allow $a^{2^e 3^c 5^0, 2^{e+1} 3^c 5^0} \in L(\mathcal{G}, G_i)$ for $a^{2^e 3^c 5^0, 2^e 3^c 5^0} \in L(\mathcal{G}, A_i)$. Here the width is multiplied by 2 and the second counter is zero which corresponds to width and height of a picture in $L(\mathcal{G}, N^{5,m})$ being congruent $0 < m < 5$ modulo 5. Then the production $A_j \to \begin{smallmatrix} G_i \\ Q^{1,2} \end{smallmatrix}$ allows $a^{2^{e+1} 3^c 5^0, 2^{e+1} 3^c 5^0} \in L(\mathcal{G}, A_j)$.

In each case the production from A_j takes care to complete again to a square picture thus exacly those pictures representing an encoding of a reachable configuration with state q_j are produced. Since $L(\mathcal{G}) = L(\mathcal{G}, A_n)$ is non empty if and only is M halts, this concludes the proof. □

Literatur

[1] M. L. MINSKY, *Computation: Finite and Infinite Machines*. Prentice-Hall, 1971.

[2] D. PRUSA, (Un)decidability of the Emptiness Problem for Multi-dimensional Context-Free Grammars. In: *Implementation and Application of Automata - 20th International Conference, CIAA 2015, Umeå, Sweden, August 18-21, 2015, Proceedings*. 2015, 250–262. http://dx.doi.org/10.1007/978-3-319-22360-5_21

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 92–93.

Weighted Restarting Automata and Pushdown Relations

Qichao Wang[(A)] Norbert Hundeshagen[(A)] Friedrich Otto[(A)]

[(A)]Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
{wang,hundeshagen,otto}@theory.informatik.uni-kassel.de

Zusammenfassung

Weighted restarting automata have been introduced to study quantitative aspects of computations of restarting automata. Here we study the special case of assigning words as weights from the semiring of formal languages over a given (output) alphabet, in this way generalizing the restarting transducers introduced by Hundeshagen (2013). We obtain several new classes of word relations in terms of restarting automata, which we relate to various types of pushdown relations.

1. Introduction

Analysis by reduction is a linguistic technique that is used to check the correctness of sentences of natural languages through sequences of local simplifications. The *restarting automaton* (or RRWW-automaton for short) was invented as a formal model for the analysis by reduction [2]. Just as a finite automaton, also a restarting automaton accepts or rejects its input. Therefore, we can say that such an automaton computes a Boolean function. In order to study quantitative aspects of computations of restarting automata, *weighted restarting automata* were introduced in [4]. These automata are obtained by assigning an element of a given semiring S as a weight to each transition of a restarting automaton. Then the product (in S) of the weights of all transitions that are used in a computation yields a weight for that computation, and by forming the sum over the weights of all accepting computations for a given input $w \in \Sigma^*$, a value from S is assigned to w. Thus, a partial function $f : \Sigma^* \dashrightarrow S$ is obtained. If S is the semiring of formal languages over some finite (output) alphabet Δ, then f is a transformation from Σ^* into the languages over Δ. Hence, in this way we obtain a generalization of the notion of a *restarting transducer* (or RRWW-transducer for short) as it was introduced in [1]. In analogy to finite transducers and pushdown transducers, a restarting transducer is a restarting automaton that is equipped with an additional output function which gives an output word for each restart and each accept transition. Therefore, restarting transducers are a special type of word-weighted restarting automata.

2. Results

It is well known (see, e.g., [3]) that the class of languages that are accepted by monotone RRWW-automata and monotone RWW-automata (a restricted type of RRWW-automata) coincides with the class of context-free languages. Accordingly, we are interested in the classes of transformations that are computed by various types of weighted restarting automata that are *monotone*. We prove that monotone weighted RRWW-automata compute strictly more transformations than monotone weighted RWW-automata, which is the first result that establishes a difference in the computational power between a model of the monotone RWW-automaton and the corresponding model of the monotone RRWW-automaton.

In this work we relate the classes of transformations that are computed by (monotone) weighted RWW- and RRWW-automata and transducers to the class of *pushdown relations* and some of its subclasses, including *linearly bounded pushdown relations*, *realtime pushdown relations* and *quasi-realtime pushdown relations*. In particular, we obtain that monotone RWW- and RRWW-transducers are equally powerful, and they characterize the class of quasi-realtime pushdown relations. Further, it is shown that any linearly bounded pushdown relation is computable by a non-monotone RRWW-transducers. Finally, we prove that the classes of transformations that are computed by monotone weighted RWW- and RRWW-automata are incomparable to the classes of pushdown relations and linearly bounded pushdown relations.

Literatur

[1] N. HUNDESHAGEN, F. OTTO, Characterizing the Rational Functions by Restarting Transducers. In: A. H. DEDIU, C. MARTÍN-VIDE (eds.), *LATA 2012*. Lecture Notes in Computer Science 7183, Springer, Heidelberg, 2012, 325–336.

[2] P. JANCAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Restarting Automata. In: H. REICHEL (ed.), *FCT'95*. Lecture Notes in Computer Science 965, Springer, Heidelberg, 1995, 283–292.

[3] P. JANCAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, On Monotonic Automata with a Restart Operation. *J. Auto. Lang. Comb.* **4** (1999) 4, 287–312.

[4] F. OTTO, Q. WANG, *Weighted Restarting Automata*, 2014. Submitted. The results of this paper have been announced at WATA 2014 in Leipzig.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 94–97.

On Nondeterminism in Ordered Restarting Automata

Kent Kwee[(A)] Friedrich Otto[(A)]

[(A)]Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
{kwee,otto}@theory.informatik.uni-kassel.de

Abstract

While (stateless) deterministic ordered restarting automata accept exactly the regular languages, it is known that nondeterministic ordered restarting automata accept some languages that are not context-free. Here we show that, in fact, the class of languages accepted by these automata is an abstract family of languages that is incomparable to the linear languages, the context-free languages, and the growing context-sensitive languages with respect to inclusion, and that the emptiness problem is decidable for these languages. In addition, it is shown that stateless ordered restarting automata just accept regular languages, and we present an infinite family of regular languages C_n such that C_n is accepted by a stateless ordered restarting automaton with an alphabet of size $O(n)$, but each stateless deterministic ordered restarting automaton for C_n needs $2^{O(n)}$ letters.

1. Introduction

The *ordered restarting automaton* (ORWW-automaton for short) was introduced in [3], where it was extended into a device for recognizing picture languages. An ORWW-automaton (for strings) has a finite-state control, a tape with end markers that initially contains the input, and a window of size three. Based on its state and the content of its window, the automaton can either perform a *move-right step*, a *rewrite/restart step*, or an *accept step*. While the deterministic variant of the ORWW-automaton characterizes the regular languages, it has been observed that the nondeterministic variant accepts some languages that are not context-free. However, the nondeterministic ORWW-automaton and the languages it accepts has not yet been studied in detail.

Here we present such a study. First we prove that the class of languages accepted by the ORWW-automaton forms an abstract family of languages, that is, it is closed under union, intersection (with regular sets), product, Kleene star, inverse morphisms, and non-erasing morphisms. However, it is neither closed under complementation nor under reversal. Further, it is incomparable to the linear, the context-free, and the growing context-sensitive languages with respect to inclusion, as it contains a language that is not even growing context-sensitive, while on the other hand, it does not even include all linear languages. In addition, we show that the emptiness problem is decidable for ORWW-automata. Several of these proofs are based on a Cut-and-Paste Lemma for ORWW-automata that is derived from Higman's Theorem [1].

In [4] an investigation of the descriptional complexity of the det-ORWW-automaton was initiated. Each det-ORWW-automaton can be simulated by an automaton of the same type that has only a single state, which means that for these automata, states are actually not needed. Accordingly, such an automaton is called a *stateless* det-ORWW-automaton. For these automata, the size of their working alphabets can be taken as a measure for their descriptional complexity. For $n \geq 1$, there exists a regular language that is accepted by a stateless det-ORWW-automaton of size $O(n)$ such that each deterministic finite-state acceptor (DFA) for this language has at least 2^{2^n} many states. On the other hand, each stateless det-ORWW-automaton of size n can be simulated by an unambiguous nondeterministic finite-state acceptor (NFA) with $2^{O(n)}$ states [2], and therewith by a DFA with $2^{2^{O(n)}}$ states. Thus, the stateless det-ORWW-automaton is exponentially more succinct than NFAs.

Here we are also interested in the computational capacity of stateless ORWW-automata and in their descriptional complexity. We show that stateless ORWW-automata only accept regular languages, which shows that states are actually useful for nondeterministic ORWW-automata. Finally, we present a family of example language $(C_n)_{n\geq 1}$ such that the language C_n is accepted by a stateless ORWW-automaton with an alphabet of size $O(n)$, while each stateless deterministic ORWW-automaton for C_n needs an alphabet of size 2^n.

2. Ordered Restarting Automata

An *ordered restarting automaton* (ORWW-automaton) is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$, where Q is a finite set of states containing the initial state q_0, Σ is a finite input alphabet, Γ is a finite tape alphabet such that $\Sigma \subseteq \Gamma$, the symbols $\rhd, \lhd \notin \Gamma$ serve as markers for the left and right border of the work space, respectively,

$$\delta : (Q \times ((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\}) \cup \{\rhd\lhd\})) \to 2^{(Q \times \mathsf{MVR}) \cup \Gamma \cup \{\mathsf{Accept}\}}$$

is the *transition relation*, and $>$ is a *partial ordering* on Γ. The transition relation describes three different types of transition steps:

(1) A *move-right step* has the form $(q', \mathsf{MVR}) \in \delta(q, a_1a_2a_3)$, where $q, q' \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, and $a_2, a_3 \in \Gamma$. It causes M to shift the window one position to the right and to change from state q into state q'.

(2) A *rewrite/restart step* has the form $b \in \delta(q, a_1a_2a_3)$, where $q \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, $a_2, b \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$ such that $a_2 > b$ holds. It causes M to replace the symbol a_2 in the middle of its window by the symbol b and to restart, that is, the window is moved back to the left end of the tape, and M reenters its initial state q_0.

(3) An *accept step* has the form $\mathsf{Accept} \in \delta(q, a_1a_2a_3)$, where $q \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, $a_2 \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$. It causes M to halt and accept. In addition, we allow an accept step of the form $\delta(q_0, \rhd\lhd) = \{\mathsf{Accept}\}$.

If $\delta(q, u) = \emptyset$ for some state q and a word u, then M necessarily halts, when it is in state q seeing u in its window, and we say that M *rejects* in this situation. Further, the letters in $\Gamma \setminus \Sigma$ are called *auxiliary symbols*.

If $Q = \{q_0\}$, that is, if the initial state is the only state of M, then we call M a *stateless* ORWW-automaton (stl-ORWW) or a *stateless deterministic* ORWW-automaton (stl-det-ORWW), as in this case the state is actually not needed. Accordingly, for stateless ORWW-automata, we simplify the notation by dropping the components referring to states.

Let $\Sigma = \{a, b, \$\}$, and let

$$L'_{\text{copy}} = \{\, w\$u \mid w, u \in \{a,b\}^*, |w|, |u| \geq 2, u \text{ is a scattered subsequence of } w \,\}.$$

Lemma 2.1 *The language L'_{copy} is not growing context-sensitive, but there exists an ORWW-automaton M such that $L(M) = L'_{\text{copy}}$.*

Lemma 2.2 (Cut-and-Paste Lemma)
For each ORWW-automaton M, there exists a constant $N(M) > 0$ such that each word $w \in L(M)$, $|w| \geq N(M)$, has a factorization $w = xyz$ satisfying all of the following conditions:

$$\text{(a) } |yz| \leq N(M), \text{ (b) } |y| > 0, \text{ and (c) } xz \in L(M).$$

3. Closure Properties

Theorem 3.1 *$\mathcal{L}(\mathsf{ORWW})$ is closed under union, intersection, product, Kleene star, inverse morphisms, and non-erasing morphisms.*

In order to establish some non-closure properties we consider the example language $L_{\leq} = \{\, a^m b^n \mid 1 \leq m \leq n \,\}$. Clearly, $L_{\leq}$ is a linear language.

Theorem 3.2 $L_{\leq} \notin \mathcal{L}(\mathsf{ORWW})$.

Corollary 3.3 *The language class $\mathcal{L}(\mathsf{ORWW})$ is incomparable to the language classes* LIN, CFL, *and* GCSL *with respect to inclusion.*

Finally, Theorem 3.2 allows us to derive the following non-closure properties.

Theorem 3.4 *The language class $\mathcal{L}(\mathsf{ORWW})$ is neither closed under the operation of reversal nor under complementation.*

Theorem 3.5 *The emptiness problem for ORWW-automata is decidable.*

4. Stateless ORWW-Automata

In [2] it is shown that each stateless det-ORWW-automaton can be turned into an unambiguous NFA that accepts the same language. This construction can easily carried over to stateless ORWW-automata.

Theorem 4.1 *Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta_M, >)$ be a stl-ORWW-automaton. Then an NFA $A = (Q, \Sigma, \Delta_A, q_0, F)$ can be constructed from M such that $L(A) = L(M)$ and $|Q| \in 2^{O(|\Gamma|)}$.*

Corollary 4.2 $\mathcal{L}(\mathsf{stl\text{-}ORWW}) = \mathsf{REG}$.

Finally, we show that stateless ORWW-automata can describe some (regular) languages much more succinctly than stateless det-ORWW-automata.

Let $\Sigma = \{a, b, \#\}$, and let, for $n \geq 1$, C_n denote the following language over Σ:

$$C_n = \{\, u_1 \# u_2 \# \ldots \# u_m \mid m \geq 2, u_1, \ldots, u_m \in \{a,b\}^n, \exists i < j : u_i = u_j \,\}.$$

Lemma 4.3 *The language C_n is accepted by a stl-ORWW-automaton M_n with a tape alphabet of size $70n - 20$.*

Proposition 4.4 *Let $s : \mathbb{N} \to \mathbb{N}$ be a function such that, for each $n \geq 1$, there exists a stl-det-ORWW-automaton $D_n = (\Sigma, \Gamma_n, \rhd, \lhd, \delta_n, >)$ such that $L(D_n) = C_n$ and $|\Gamma_n| \leq s(n)$. Then $s(n) \notin o(2^n)$.*

Corollary 4.5 *Let $s : \mathbb{N} \to \mathbb{N}$ be a function such that, for each $n \geq 1$, there exists a stl-ORWW-automaton $E_n = (\Sigma, \Gamma_n, \rhd, \lhd, \delta_n, >)$ such that $L(E_n) = C_n^c$ and $|\Gamma_n| \leq s(n)$. Then $s(n) \notin o(2^n)$.*

5. Concluding Remarks

The class of languages accepted by nondeterministic ORWW-automata forms an abstract class of languages that is incomparable to the linear, context-free, and growing context-sensitive languages with respect to inclusion. However, we don't know yet whether this class is closed under arbitrary morphisms. In addition, we have shown that the emptiness problem is decidable for these languages, but it remains to find an efficient algorithm for this problem. Also it is still open whether finiteness, inclusion, and equivalence are decidable. Further, we have seen that stateless ORWW-automata provide another characterization for the regular languages, which provides exponentially more succinct representations than stateless deterministic ORWW-automata. However, we are still missing an algorithm for turning a stl-ORWW-automaton into an equivalent stl-det-ORWW-automaton without constructing an equivalent NFA as an intermediate step.

References

[1] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 2:326–336, 1952.

[2] K. Kwee and F. Otto. On some decision problems for stateless deterministic ordered restarting automata. In J. Shallit and A. Okhotin, editors, *DCFS 2015, Proc.*, *LNCS 9118*, pages 165–176. Springer, Heidelberg, 2015.

[3] F. Mráz and F. Otto. Ordered restarting automata for picture languages. In V. Geffert, B. Preneel, B. Rovan, J. Štuller, and A. Min Tjoa, editors, *SOFSEM 2014, Proc.*, *LNCS 8327*, pages 431–442. Springer, Heidelberg, 2014.

[4] F. Otto. On the descriptional complexity of deterministic ordered restarting automata. In H. Jürgensen, J. Karhumäki, and A. Okhotin, editors, *DCFS 2014, Proc.*, *LNCS 8614*, pages 318–329. Springer, Heidelberg, 2014.

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 98–101.

Tinput-Driven Pushdown Automata

Martin Kutrib[(A)] Andreas Malcher[(A)] Matthias Wendlandt[(A)]

[(A)]Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
{kutrib,malcher,matthias.wendlandt}@informatik.uni-giessen.de

Abstract

In input-driven pushdown automata (IDPDA) the input alphabet is divided into three distinct classes and the actions on the pushdown store (push, pop, nothing) are solely governed by the input symbols. Here, this model is extended in such a way that the input of an IDPDA is preprocessed by a deterministic sequential transducer. These automata are called tinput-driven pushdown automata (TDPDA) and it turns out that TDPDAs are more powerful than IDPDAs but still not as powerful as real-time deterministic pushdown automata. Nevertheless, even this stronger model has still good closure and decidability properties. In detail, it is shown that TDPDAs are closed under the Boolean operations union, intersection, and complementation. Furthermore, decidability procedures for the inclusion problem as well as for the questions of whether a given automaton is a TDPDA or an IDPDA are developed. Finally, representation theorems for the context-free languages using IDPDAs and TDPDAs are established.

1. Introduction

An interesting subclass of pushdown automata (PDA) is represented by *input-driven* PDAs. The essential idea here is that for such devices the operations on the storage medium are dictated by the input symbols. The first references of input-driven PDAs may be found in [5, 14], where input-driven PDAs are introduced as classical PDAs in which the input symbols define whether a push operation, a pop operation, or no operation on the pushdown store has to be performed. The main results obtained there show that the membership problem for input-driven PDAs can be solved in logarithmic space, and that the nondeterministic model can be determinized. More on the membership problem has been shown in [8] where the problem is classified to belong to the parallel complexity class NC^1.

The investigation of input-driven PDAs has been revisited in [1, 2], where such devices are called visibly PDA or nested word automata. Some of the results are the classification of the language family described by input-driven PDAs to lie properly in between the regular and the deterministic context-free languages, the investigation of closure properties and decidable questions which turn out to be similar to those of regular languages, and descriptional complexity results for the trade-off occurring when nondeterminism is removed from input-driven PDAs. A recent survey with many valuable references on complexity aspects of input-driven PDAs may

be found in [16]. Further aspects such as the minimization of input-driven PDAs and a comparison with other subclasses of deterministic context-free languages have been studied in [6, 7] while extensions of the model with respect to multiple pushdown stores or more general auxiliary storages are introduced in [12, 13]. Recently, the computational power of input-driven automata using the storage medium of a stack and a queue, respectively, have been investigated in [3, 11].

The edge between deterministic context-free languages that are accepted by an IDPDA or not is very small. For example, language $\{ a^n\$b^n \mid n \geq 1 \}$ is accepted by an IDPDA where an a means a push-operation, b means a pop-operation, and a \$ leaves the pushdown store unchanged. On the other hand, the very similar language $\{ a^n\$a^n \mid n \geq 1 \}$ is not accepted by any IDPDA. Similarly, the language $\{ w\$w^R \mid w \in \{a,b\}^+ \}$, where w^R denotes the reversal of w, is not accepted by any IDPDA, but if w^R is written down with some marked alphabet $\{\hat{a},\hat{b}\}$, then language $\{ w\$\hat{w}^R \mid w \in \{a,b\}^+ \}$ is accepted by an IDPDA. To overcome these obstacles we consider a sequential transducer that translates some input to some output which in turn is the input for an IDPDA. In the above first example such a transducer translates every a before reading \$ to a and after reading \$ to b. In the second example a and b are translated to a, b or $\hat{a}$, $\hat{b}$, respectively, depending on whether or not \$ has been read. We call such a pair of a sequential transducer and an IDPDA *tinput-driven* PDA (TDPDA). To implement the idea without giving the transducers too much power for the overall computation, essentially, we will consider only deterministic injective and length-preserving transducers.

2. Definition

A classical deterministic pushdown automaton is called input-driven if the next input symbol defines the next action on the pushdown store, that is, pushing a symbol onto the pushdown store, popping a symbol from the pushdown store, or changing the state without modifying the pushdown store. To this end, we assume the input alphabet Σ to be partitioned into the sets Σ_N, Σ_D, and Σ_R, that control the actions state change only (N), push (D), and pop (R). A formal definition is:

Definition 2.1 *A* deterministic input-driven pushdown automaton (IDPDA) *is a system* $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$*, where,* Q *is the finite set of* internal states,Σ *is the finite set of* input symbols *partitioned into the sets* Σ_D*,* Σ_R*, and* Σ_N*,* Γ *is the finite set of* pushdown symbols*,* $q_0 \in Q$ *is the* initial state*,* $F \subseteq Q$ *is the set of* accepting states*,* $\bot \notin \Gamma$ *is the* empty pushdown symbol*,* δ_D *is the partial transition function mapping from* $Q \times \Sigma_D \times (\Gamma \cup \{\bot\})$ *to* $Q \times \Gamma$*,* δ_R *is the partial transition function mapping from* $Q \times \Sigma_R \times (\Gamma \cup \{\bot\})$ *to* Q*,* δ_N *is the partial transition function mapping from* $Q \times \Sigma_N \times (\Gamma \cup \{\bot\})$ *to* Q*.*

A *configuration* of an IDPDA $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ is a triple (q, w, s), where $q \in Q$ is the current state, $w \in \Sigma^*$ is the unread part of the input, and $s \in \Gamma^*$ denotes the current pushdown content, where the leftmost symbol is at the top of the pushdown store. The *initial configuration* for an input string w is set to (q_0, w, λ). During the course of its computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $a \in \Sigma$, $w \in \Sigma^*$, $z, z' \in \Gamma$, and $s \in \Gamma^*$. We set

1. $(q, aw, zs) \vdash (q', w, z'zs)$, if $a \in \Sigma_D$ and $(q', z') \in \delta_D(q, a, z)$,
2. $(q, aw, \lambda) \vdash (q', w, z')$, if $a \in \Sigma_D$ and $(q', z') \in \delta_D(q, a, \bot)$,
3. $(q, aw, zs) \vdash (q', w, s)$, if $a \in \Sigma_R$ and $q' \in \delta_R(q, a, z)$,
4. $(q, aw, \lambda) \vdash (q', w, \lambda)$, if $a \in \Sigma_R$ and $q' \in \delta_R(q, a, \bot)$,
5. $(q, aw, zs) \vdash (q', w, zs)$, if $a \in \Sigma_N$ and $q' \in \delta_N(q, a, z)$,
6. $(q, aw, \lambda) \vdash (q', w, \lambda)$, if $a \in \Sigma_N$ and $q' \in \delta_N(q, a, \bot)$.

The language accepted by the IDPDA M is the set $L(M)$ of words for which there exists some computation beginning in the initial configuration and ending in a configuration in which the whole input is read and an accepting state is entered. Formally:

$$L(M) = \{\, w \in \Sigma^* \mid (q_0, w, \lambda) \vdash^* (q, \lambda, s) \text{ with } q \in F, s \in \Gamma^* \,\}.$$

For the definition of tinput-driven pushdown automata we need the notion of *deterministic one-way sequential transducers* (DST) which are basically deterministic finite automata equipped with an initially empty output tape. In every transition a DST appends a string over the output alphabet to the output tape. The transduction defined by a DST is the set of all pairs (w, v), where w is the input and v is the output produced after having read w completely. Formally, a DST is a system $T = \langle Q, \Sigma, \Delta, q_0, \delta \rangle$, where Q is the finite set of internal states, Σ is the finite set of input symbols, Δ is the finite set of output symbols, $q_0 \in Q$ is the initial state, and δ is the partial transition function mapping $Q \times \Sigma$ to $Q \times \Delta^*$. By $T(w) \in \Delta^*$ we denote the output produced by T on input $w \in \Sigma^*$. In the following, we will consider only injective and length-preserving DSTs.

Let M be an IDPDA and T be an injective and length-preserving DST. Furthermore, the output alphabet of T is the input alphabet of M. Then, the pair (M, T) is called a *tinput-driven pushdown automaton* (TDPDA) and the language accepted by (M, T) is defined as $L(M, T) = \{\, w \in \Sigma^* \mid T(w) \in L(M) \,\}$.

In order to clarify this notion we continue with an example.

Example 2.2 *Language $L_1 = \{\, a^n \$ a^n \mid n \geq 1 \,\}$ is accepted by a TDPDA. Before reading symbol \$ the transducer maps an a to an a, and after reading \$ it maps an a to a b. Thus, L_1 is translated to $\{\, a^n \$ b^n \mid n \geq 1 \,\}$ which is accepted by some IDPDA.*

Consider $L_2 = \{\, a^n b^{2n} \mid n \geq 1 \,\}$. Here, the transducer maps an a to a and every b alternately to b and c. This gives language $\{\, a^n (bc)^n \mid n \geq 1 \,\}$ which is accepted by some IDPDA: every a implies a push-operation, every b implies a pop, and every c leaves the pushdown store unchanged.

References

[1] Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Babai, L. (ed.) Symposium on Theory of Computing (STOC 2004). pp. 202–211. ACM (2004)

[2] Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM 56 (2009)

[3] Bensch, S., Holzer, M., Kutrib, M., Malcher, A.: Input-driven stack automata. In: Baeten, J.C.M., Ball, T., de Boer, F.S. (eds.) Theoretical Computer Science (TCS 2012). LNCS, vol. 7604, pp. 28–42. Springer (2012)

[4] Bordihn, H., Holzer, M., Kutrib, M.: Economy of description for basic constructions on rational transductions. J. Autom. Lang. Comb. 9, 175–188 (2004)

[5] von Braunmühl, B., Verbeek, R.: Input-driven languages are recognized in $\log n$ space. In: Karpinski, M., van Leeuwen, J. (eds.) Topics in the Theory of Computation, Mathematics Studies, vol. 102, pp. 1–19. North-Holland (1985)

[6] Chervet, P., Walukiewicz, I.: Minimizing variants of visibly pushdown automata. In: Kucera, L., Kucera, A. (eds.) Mathematical Foundations of Computer Science (MFCS 2007). LNCS, vol. 4708, pp. 135–146. Springer (2007)

[7] Crespi-Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. J. Comput. System Sci. 78, 1837–1867 (2012)

[8] Dymond, P.W.: Input-driven languages are in $\log n$ depth. Inform. Process. Lett. 26, 247–250 (1988)

[9] Hartmanis, J.: Context-free languages and Turing machine computations. Proc. Symposia in Applied Mathematics 19, 42–51 (1967)

[10] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)

[11] Kutrib, M., Malcher, A., Mereghetti, C., Palano, B., Wendlandt, M.: Deterministic input-driven queue automata: Finite turns, decidability, and closure properties. Theor. Comput. Sci. 578, 58–71 (2015)

[12] La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: Logic in Computer Science (LICS 2007). pp. 161–170. IEEE Computer Society (2007)

[13] Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: Ball, T., Sagiv, M. (eds.) Principles of Programming Languages, (POPL 2011). pp. 283–294. ACM (2011)

[14] Mehlhorn, K.: Pebbling moutain ranges and its application of DCFL-recognition. In: de Bakker, J.W., van Leeuwen, J. (eds.) Int. Coll. on Automata, Languages and Programming (ICALP 1980). LNCS, vol. 85, pp. 422–435. Springer (1980)

[15] Myhill, J.: Finite automata and the representation of events. Tech. Rep. TR 57-624, WADC (1957)

[16] Okhotin, A., Salomaa, K.: Complexity of input-driven pushdown automata. SIGACT News 45, 47–67 (2014)

[17] Schützenberger, M.-P.: Sur les relations rationelles. In: Braklage, H. (ed.) Automata Theory and Formal Languages. LNCS, vol. 33, pp. 209–213. Springer (1975)

[18] Sheng, Y.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, Vol. 1. pp. 41–110. Springer (1997)

THEORIE-TAGE 2015
H. Fernau (Herausgeber): Theorietage 2015, Speyer, 29.09. – 02.10.2015
Universität Trier, Technischer Bericht, S. 102–105.

Visibly Counter Languages and the Structure of NC^1

Michael Hahn[(A)] Andreas Krebs[(A)] Klaus-Jörn Lange[(A)]
Michael Ludwig[(A)]

[(A)]WSI - University of Tübingen
Sand 13, 72076 Tübingen, Germany
{hahnm,krebs,lange,ludwigm}@informatik.uni-tuebingen.de

Abstract

There are well-known and deep connections between circuit complexity classes and algebraic classes of regular languages. We extend this correspondence beyond the realm of regular languages to Visibly Counter Languages, which like the regular languages are NC^1- complete. We investigate what the visibly counter languages in certain constant depth circuit complexity classes are and present characterizations and decidability results for various logics and circuit classes. In particular, our approach yields a way to understand TC^0, which is not known to have complete regular languages.

1. Introduction

Connections between circuit complexity and algebraic classes of regular languages, based on the Furst-Saxe-Sipser theorem [6, 8] and Barrington's theorem [3] were already uncovered in the 1980s [5]. These results showed that circuit classes have effective characterizations in terms of algebraic properties of finite monoids, which has led to the characterization of class separations in terms of properties of regular sets. For instance, the classes ACC^0 and NC^1 differ if and only if no syntactic monoid of a regular set in ACC^0 contains a nonsolvable group. On the other hand, very few decidable characterizations of complexity classes are known for nonregular language classes. Our contribution consists in such a characterization: we lift the correspondence between circuit classes and algebraic properties of regular languages to *visibly counter languages*.

Visibly pushdown languages.

Visibly counter languages [2] are a restricted class of *visibly pushdown languages*, which were originally introduced by Mehlhorn [11] in 1980 and later popularized by Alur and Madhusudan [1] in 2004. These are context-free languages recognized by visibly pushdown automata, i.e., pushdown automata for which the input symbol determines whether a symbol is pushed or popped. This leads to a partitioning $\Sigma = \Sigma_{call} \cup \Sigma_{neutral} \cup \Sigma_{return}$. Visibly counter languages are visibly pushdown languages recognized by visibly pushdown automata whose stack alphabet is a singleton set.

Our Results.

We examine the intersection of the visibly counter languages with the circuit classes AC^0, $CC^0[q]$, CC^0, $ACC^0[q]$, ACC^0, TC^0, and NC^1. As our main theorem, we unconditionally prove criteria for visibly counter languages to be complete for these classes. The results are summarized in Figure 1, which compares the known families of regular languages complete for various circuit classes with corresponding complete classes of visibly counter languages. Furthermore, given a visibly counter automaton, we can effectively compute the smallest of these complexity classes that contains the language recognized by this automaton. This also provides an effective method to decide whether a language, given by a visibly counter automaton, is in one of these complexity classes. Decidability results for nonregular languages classes corresponding to circuit complexity classes are rather sparse. We expect that our more general method will help to lift various decidabilty results about regular languages to visibly counter languages.

2. Proof Sketch

In this section, we sketch the proof. For the details, we refer to [9] and [7]. We obtain our results by splitting the computation of visibly counter automata into two parts. Each position in a word over a visibly alphabet can be assigned a height. The first computation step is determining this height. Assuming the height computation is already performed, what is left is the computation performed by the states of the automaton. We call this the *regular part*. After we have established those two parts, we derive results on complexity classes.

2.1. Height Computation for Visibly Counter Languages

The *height* $\Delta(w)$ of a word $w = w_1...w_n$ is defined to be $|w|_{\Sigma_{call}} - |w|_{\Sigma_{return}}$. Visibly-counter automata automatically reject words having a prefix of negative height. We defined a notion of *simple height behavior* [9] which guarantees that the computation of the heights of all words can be approximated in AC^0. Informally, a VCL has simple height behavior if whenever a recognizing automaton loops through a state q reading a subword x, the height difference $\Delta(x)$ is determined by q and the length $|w|$. In the case of simple height behavior, the height is computable in FO[+] and thus in AC^0. On the other hand, if a VCL does not have simple height

Circuit Class	Regular Languages	Visibly Counter Languages	
	Syntactic Monoids	Stack	Regular Part
CC^0	solvable groups	no stack	& solvable groups
AC^0	quasi aperiodic	well behaved	& quasi aperiodic
ACC^0	solvable monoids	well behaved	& solvable monoids
TC^0	—	any	& solvable monoids
NC^1	finite monoids	any	& finite monoids

Figure 1: Circuit classes with complete classes of regular languages [12], and complete visibly counter classes.

behavior, it is TC^0-hard by quantifier-free Turing reductions [9]. So in conclusion we get a dichotomy: Either the height is computable in AC^0, or the language TC^0- hard.

2.2. The Regular Part of Visibly Counter Languages

In the previous section we studied the compelxity of computing the height of words. In this section we want to look at what is left if we presuppose that the heights of all prefixes have already been computed. We will do so in coding the height up to a threshold m into the input by a function τ_m labeling symbols with their height up to the threshold m. For instance, if a and b are push and pop letters respectively, then $\tau_2(aaaab) = (a,0)(a,1)(a,2)(a,2)(b,2)$. What is left when the height has been computed is the finite-state control of the visibly counter automaton. Its action is modeled by a regular language, the *regular part* [9], which is defined by a finite automaton simulating the finite control of the visibly counter automaton, with the stack being replaced by the height labeling τ_m.

The regular languages in some constant-depth circuit class C are known to be determined already by the class of simple monoids whose word problem is in C [12]. Broadly speaking, a regular language L is in a circuit class if and only if, for all simple monoids 'contained in' L, their word problems are also in C. In the regular case, the appropriate notion of a monoid M 'being contained' in a language L with syntactic morphism η_L is that there a $t > 0$ such that $\eta_L(\Sigma^t)$ contains a monoid divided by M [4]. In the case of VCLs, the precise meaning of 'being contained' in the language is similar, if slightly more involved [7]. We refer to the monoids 'contained' in L in this way by Q_L. With this definition we get results similar to the regular case – in particular, the word problems of monoids in Q_L are reducible to the membership problem of L by quantifier-free Turing reductions. Putting these hardness results together with the results on height computation, we obtain the classification in Figure 1. As in the case of regular languages, the intersections of the circuit classes with the visibly counter langzages are decidable. However, it has to be noted that for most classes, the concrete decision algorithm depends on open questions about the separation of classes, as applying it to suitable languages would immediately settle these questions.

Making assumptions about separations, we also obtain uniformity results. For instance, assuming that $ACC^0 \neq TC^0 \neq NC^1$, we can show that the visibly counter languages intersected with these languages are contained in logic classes using only semilinear predicates. This observation has been explored by [10], who introduced the notion of *extensional uniformity*, which is based on the study of the intersection of a complexity class with a family of formal languages. For AC^0, $CC^0[p]$, $ACC^0[p]$, p being prime, we obtain similar but unconditional results.

References

[1] R. ALUR, P. MADHUSUDAN, Visibly pushdown languages. In: L. BABAI (ed.), *STOC*. ACM, 2004, 202–211.

[2] V. BÁRÁNY, C. LÖDING, O. SERRE, Regularity Problems for Visibly Pushdown Languages. In: B. DURAND, W. THOMAS (eds.), *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*. Lecture Notes in

Computer Science 3884, Springer, 2006, 420–431.
http://dx.doi.org/10.1007/11672142_34

[3] D. A. M. BARRINGTON, Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. *J. Comput. Syst. Sci.* **38** (1989) 1, 150–164.

[4] D. A. M. BARRINGTON, K. J. COMPTON, H. STRAUBING, D. THÉRIEN, Regular Languages in NC^1. *J. Comput. Syst. Sci.* **44** (1992) 3, 478–499.

[5] D. A. M. BARRINGTON, D. THÉRIEN, Finite monoids and the fine structure of NC^1. *J. ACM* **35** (1988) 4, 941–952.

[6] M. L. FURST, J. B. SAXE, M. SIPSER, Parity, Circuits, and the Polynomial-Time Hierarchy. In: *FOCS*. 1981, 260–270.

[7] M. HAHN, A. KREBS, K.-J. LANGE, M. LUDWIG, Visibly Counter Languages and the Structure of NC1. In: *MFCS 2015*. 2015.

[8] J. HÅSTAD, Almost Optimal Lower Bounds for Small Depth Circuits. In: *STOC*. ACM, 1986, 6–20.

[9] A. KREBS, K. LANGE, M. LUDWIG, Visibly Counter Languages and Constant Depth Circuits. In: E. W. MAYR, N. OLLINGER (eds.), *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*. LIPIcs 30, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, 594–607.
http://dx.doi.org/10.4230/LIPIcs.STACS.2015.594

[10] P. MCKENZIE, M. THOMAS, H. VOLLMER, Extensional Uniformity for Boolean Circuits. *SIAM J. Comput.* **39** (2010) 7, 3186–3206.
http://dx.doi.org/10.1137/080741811

[11] K. MEHLHORN, Pebbling mountain ranges and its application to DCFL-recognition. In: J. DE BAKKER, J. VAN LEEUWEN (eds.), *Automata, Languages and Programming*. Lecture Notes in Computer Science 85, Springer Berlin Heidelberg, 1980, 422–435.
http://dx.doi.org/10.1007/3-540-10003-2_89

[12] H. STRAUBING, *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, 1994.