32. Theorietag
Automaten und Formale Sprachen

Caputh, 19.–21. September 2022

Universität Potsdam

Institut für Informatik und Computational Science
An der Bahn 2
14476 Potsdam
Herausgeber: Henning Bordihn

Vorwort

Der Theorietag ist die Jahrestagung der Fachgruppe Automaten und Formale Sprachen der Gesellschaft für Informatik und wird seit 1991 von Mitgliedern der Fachgruppe abwechselnd und an wechselnden Orten in Deutschland, Österreich und Tschechien veranstaltet. Seit dem Jahr 1996 wird der Theorietag von einem eintägigen Workshop mit eingeladenen Vorträgen begleitet. Im Laufe des Theorietags findet auch die jährliche Fachgruppensitzung statt.

Der diesjährige Theorietag wird nach 2004 und 2014 wieder von Mitgleidern der Fachgruppe organisiert, die der Universität Potsdam angehören. Er findet mit dem vorangehenden Workshop vom 23. bis 25. September 2014 im Hotel Märkisches Gildehaus in Caputh am Templiner See bei Potsdam statt. Auf dem Workshop tragen

- Nuria Brede vom Potsdam-Institut für Klimafolgenforschung,
- Pamela Fleischmann von der Christian-Albrechts-Universität zu Kiel,
- Detlef Groth von der Universität Potsdam und
- Markus Schmid von der Humboldt-Universität zu Berlin

vor. Auf dem Programm des Theorietags stehen 13 weitere Vorträge. Der vorliegende Tagungsband enthält Kurzfassungen aller 17 Beiträge.

Wir danken den eingeladenen Vortragenden für ihre Bereitschaft, den Workshop mit einem Beitrag zu unterstützen sowie allen Teilnehmern und Teilnehmerinnen, die einen Beitrag zum Tagungsprogramm eingereicht haben. Ferner sei der Gesellschaft für Informatik und der Universität Potsdam für die Unterstützung dieses Theorietags gedankt. Ein besonderer Dank gilt Tim Richter für die Unterstützung bei der Gestaltung der Webseite des Theorietags. Wir wünschen allen Teilnehmern eine interessante und anregende Tagung sowie einen angenehmen Aufenthalt in Caputh.

Potsdam, im September 2022 Henning Bordihn

Inhalt

THEORIE-TAG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 7–7.

Monadic Dynamical Systems – Generic Automata?

Nuria Brede[A]

[A]Potsdam Institute for Climate Impact Research (PIK)
RD4: Complexity Science, Potsdam, Germany
nuria.brede@pik-potsdam.de

Abstract

Monadic dynamical systems have been introduced by Ionescu as generalisations of ordinary (non-)deterministic or stochastic dynamical systems. In subsequent work, Botta, Ionescu and Jansson have proposed a framework for specifying and solving optimisation problems on top of such dynamical systems. The kind of optimisation problems that the framework can be applied to are relevant in many domains. Here, the development has been motivated by applications to climate policy advice and the wish to make such advice more accountable. The framework has been formalised in the dependently-typed programming language Idris – a language that allows to express mathematical problem specifications, implementations and their correctness proofs, thereby increasing transparency and reliability. In my talk, I will introduce the Botta-Ionescu-Jansson framework and recent extensions, and try to sketch an automata-theoretic perspective by considering monadic dynamical systems as generic automata.

[A]This talk is about joint work with Nicola Botta, Michel Crucifix, Marina Martínez Montero, Patrik Jansson, Cezar Ionescu and Tim Richter in the context of the project *TiPES – Tipping Points in the Earth System*. The TiPES project has received funding from the European Union´s Horizon 2020 research and innovation programme under grant agreement No 820970.

THEORIE-TAG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 9–9.

Scattered Factor Universality - an Approach to Investigate Simon's Congruence

Pamela Fleischmann[(A)]

[(A)]Kiel University, Germany
fpa@informatik.uni-kiel.de

Abstract

A word $u \in \Sigma^*$ is a scattered factor of a word $w \in \Sigma^*$ if u can be obtained from w by deleting some of w's letters while preserving the order of the remaining letters, e.g., the word `here` is a scattered factor of the word `theorietag` but `hero` is not, since the `o` is in front of the `r` and not behind (violating the order). In 1972, Imre Simon introduced the nowadays called *Simon congruence* to use scattered factors to measure the similarity of words. Let $\text{ScatFact}_k(w)$ denote the set of w's scattered factors of length k. Two words u and v are called *Simon congruent* ($u \sim_k v$) if we have $\text{ScatFact}_\ell(u) = \text{ScatFact}_\ell(v)$ for all $\ell \leq k$. For instance, $\text{ScatFact}_2(\texttt{abab}) = \text{ScatFact}_2(\texttt{baba})$ but $\text{ScatFact}_2(\texttt{aba}) = \{\texttt{aa}, \texttt{ab}, \texttt{ba}\} \neq \{\texttt{ab}, \texttt{ba}, \texttt{bb}\} = \text{ScatFact}_2(\texttt{bab})$. Since the introduction of this congruence by Simon, its index, i.e., $|\Sigma^*/\sim_k|$ for a fixed $k \in \mathbb{N}$ is unknown.

Motivated by the classical universality problem of languages, i.e., deciding whether a given language $L \subseteq \Sigma^*$ is completely Σ^*, we ask whether we have $\text{ScatFact}_k(w) = \Sigma^k$ for a given $k \in \mathbb{N}$ and $w \in \Sigma^*$. The words satisfying this condition are called *k-universal words*. In this work, we present a full characterisation of k-universal words.

In a second step, we relaxed the problem in the following way: a word is called *m-nearly k-universal* if we have $|\text{ScatFact}_k(w)| = |\Sigma|^k - m$, i.e., we are interested in words where exactly m scattered factors of length k are absent. Here, we have full characterisations for $m \in \{1, 2, 3\}$ and the more trivial cases like $|\Sigma|^k$ and $|\Sigma|^k - 1$ absent scattered factors.

Moreover, we present some further insights into k-universality based on the question about the universality of repetitions and palindromes. First, we can show that w is k-universal iff $\text{ScatFact}_k(w) = \text{ScatFact}_k(w^2)$ holds. Investigating the universality of w^2 revealed that for some words a conjugate (rotation) of w has a greater universality than w itself. Therefore, we defined the circular k-universality capturing the behaviour of repetitions of words.

THEORIE-AG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 11–15.

Der Scanner-Generator Re2c im Einsatz für Biologische Daten

Detlef Groth[(A)]

[(A)]Universität Potsdam
AG Bioinformatik
{dgroth}@uni-potsdam.de

Zusammenfassung

Die experimentelle moderne Biotechnologie produziert enorme Mengen an Primär-Daten, zum Beispiel von Sequenz-Daten oder Gen-Expressions-Daten und durch deren nachfolgenden Analyse mittels einer Vielzahl von an bioinformatischen und statistischen Tools von Sekundär-Daten. Die Speicherung dieser Information erfolgt zumeist in diversen Text-Datei-Formaten. Trotz aller Bemühungen einer Vereinheitlichung der Formate dieser Text-Dateien (XML, JSON, etc.) ist nach wie vor eine enorme Vielzahl an Formatier-Sprachen anzutreffen. Leicht zu programmierende Parser für diese Text-Dateien mit Hilfe von Scanner-Generatoren wie zum Beispiel Flex oder Re2c bieten eine Möglichkeit der schnellen Analyse und der Auswertung. In diesem Beitrag wird der Scanner-Generator Re2c vorgestellt welcher die Erzeugung schneller kompakter Programme für die Auswertung von Informationen in Text-Dateien und der Datei-Konvertierung ermöglicht.

1. Einleitung

Die Datenanalyse in der Bioinformatik ist gekennzeichnet von der Arbeit mit einer Vielzahl an Datenformaten und fehlenden Standardisierungen. Die diversen Datenbanken im Web bieten Primär- und Sekundär-Daten häufig in eigenen Text-Datei-Formaten zum Download an. Ähnlich ist es bei den bioinformatischen Programmen welche diese Daten für die Auwertung eigener Daten benutzen. So zum Beispiel durch den Vergleich eigener DNA oder Protein-Sequenzen mit denen der im Web vorhanden Daten. Auch bei den durch diese Tools erzeugten Sekundär-Daten existiert eine große Vielfalt an Text-Formaten. Logische Bezüge zwischen den unterschiedlichen Daten und Analysen herzustellen ist dadurch schwierig.

Versuche einer Vereinheitlichung und Vereinfachung des Datenaustausches und der Analyse haben sich als wenig erfolgversprechend erwiesen. Anläufe Dateiformate wie XML, JSON oder OBO als Austauschformate zu implementieren waren wenig erfolgversprechend. Für eine Übersicht zur Problematik empfehle ich den Artikel von Buchmann und Mitarbeitern welche ein weiteres standardisiertes (und offenbar ebenso erfolgloses) Austauschformat vorstellen [2]. Ursache fr̈ dieses Phänomen sind sicherlich die Jahr für Jahr neu in die Forschung kommenden Wissenschaftler welche in Folge des schnelllebigen Wissenschaftler-Daseins wenig Zeit

für das detaillierte Erlernen von technischen Hintergründen haben. Sie arbeiten im Rahmen ihrer Kurzzeit-Verträge sehr ergebnisorientiert. Die Weiterverwendung des von ihnen erarbeiteten Wissens mit Hilfe von strukturierten Datenformaten ist für den einzelnen Forscher hier leider eher zweitranging.

Aus diesem Grunde sind menschenlesbare Textformate von Vorteil. Ein weiterer Nachteil von Formaten mir erheblichen Markup-Aufwand wie z.B. XML ist die enorme Zunahme der Dateigröße durch die Markup-Anweisungen. Dies ist insbesondere bei den enormen Datenmengen die in der Bioinformatik anzutreffen ein erhebliches Problem. Eine neuere geeignete Markup-Sprache scheint hier YAML zu sein die sowohl gut strukturiert als auch lesbar für den Menschen ist.

Aus der soeben beschriebenen Problematik ergibt sich für den Bioinformatiker die Notwendigkeit für eine Vielzahl an Dateiformaten Programme zu schreiben welche die relevanten Informationen aus diesen Text-Dateien extrahieren und mit denen aus anderen Daten-Quellen oder Ausgabe-Dateien von bioinformatischen Tools verknüpfen. Auf Grund der teilweisen enormen Dateigrößen müssen diese Programme auch in hoher Geschwindigkeit arbeiten. Für die Erstellung solcher Programme bieten sich sogenannte Lexer bzw. Scanner-Generatoren wie Flex (C/C++), Re2c (C, C++, Go, Rust) oder Jflex (Java) an. Als Eingabe werden hier zumeist lediglich reguläre Ausdrücke und der dazugehörige Programmier-Code eingegeben. Diese Lexer erzeugen dann den entsprechenden Programmier-Code welcher von einem Compiler in ein Programm, den Scanner, übersetzt wird. Die so erzeugten Programme sind zumeist erheblich schneller als handgeschriebene Daten-Scanner, von Bioinformatikern oft auch als Parser bezeichnet.

Im folgenden Abschnitt wird der Scanner-Generator Re2c (`https://re2c.org/`) vorgestellt und seine Anwendung auf Ausgabedaten des häufig verwendete Bioinformatik-Tool BLAST illustriert. Als Ausgabe des Scanning-Prozesses wird Datenbank-Code erzeugt welcher direkt in eine Datenbank importiert werden kann. Diese Arbeitschritte ermöglichen das schnelle Verknüpfen von Informationen aus unterschiedlichen Quellen.

2. Re2c Scanner-Generator

Re2c [5] ist ein Scanner-Generator mit einem Schwerpunkt auf schnellem Ausführungs-Code. Der erzeugte Programmier-Code ist gekennzeichnet durch die Verwendung von Goto-Anweisungen. Gewöhnlich ist das daraus kompilierte Programm erheblich schneller als der von anderen Scanner-Generatoren erzeugte. Desweiteren sind die erzeugten binären Programm-Dateien extrem klein (Tabelle 2.) und benötigen nur sehr wenig Arbeitspeicher.

Das Grundprinzip beim Erstellen eines Parsers mit Hilfe eines Scanner-Generators ist das Erstellen einer Datei welche die regulären Ausrücke und den dazugehörigen Programm-Code enthält. Danach erfolgt die Übersetzung in eine C-Quellcode-Datei mit Hilfe von Re2c und anschließend die Übersetzung des C-Codes mit Hilfe des Compilers (Abbildung 1).

Neben der Dateigröße wurden außerdem noch der Speicher-Bedarf und die Ausführungs-Geschwindigkeit untersucht. Letzere ist für die praktische Anwendung sicherlich am wichtigsten. Dabei konnte festgestellt werden, dass in diesen Untersuchungen der Re2c Scanner-Generator durch enorm hohe Geschwindigkeit bei sehr geringem Speicher-Bedarf auffiel. Die Erzeugung des Datenbank-Codes verlief schneller als das Einlesen in die Datenbank. Demzu-

Scanner/ Sprache	Referenz	Compiler	WC Größe: dyn./stat (kb)
Flex / C	`https://github.com/westes/flex`	GNU gcc 4.1.2	16 / 489
Flexpp / C++	`https://github.com/westes/flex`	GNU g++ 4.1.2	21 / 1002
RE2C / C	`https://re2c.org`	GNU gcc 4.1.2	7 / 7
TPLex / Pascal	`https://github.com/martok/fpc-tply`	Free Pascal fpc 2.0.4	109 / 109
JFlex / Java	`https://www.jflex.de/`	SUN javac 1.6.0	7 / NA
GPLEX / C#	`https://github.com/k-john-gough/gplex`	Mono gmcs 2.0.1	13 / 6525

Table 1: Vergleich verschiedener Scanner-Generatoren und ihrer Dateigrößen.

folge waren weitere Geschwindigkeits-Optimierungen nicht notwendig.

Im Geschwindigkeits-Vergleich war das vom Java-Lexer Jflex erzeugte Code selbst im kompilierten Modus etwa 10 mal langsamer und der ebenfalls in C codierte Flex-Scanner etwa fünfmal langsamer als der Re2c Scanner. Überraschenderweise waren der C++ Scanner Flex++ wie auch der Mono-Scanner Code am wenigsten performant. Einschränkend ist zu sagen, dass die Vergleiche bereits vor einigen Jahren mit älteren Compilern durchgeführt wurden. Scanner in Script-Sprachen wie Perl oder Python sind zumeist weniger geeignet für das Prozessieren sehr großer Text-Dateien.

Für unsere Zwecke, hohe Ausführungs-Geschwindigkeit bei minimalem Bedarf an Arbeitsspeicher, erwies sich damit Re2c als am geeignetsten.

3. Übersetzung in Datenbank-Code

Die diversen Parser für die unterschiedlichen bioinformatischen Daten-Formate erzeugen wiederum variierende Ausgabe-Formate. Damit stellt sich erneut das Problem wie Informationen aus diesen Ergebnis-Dateien miteinander zu verknüpfen sind. Zur Lösung des Problems wurde deswegen Datenbank SQL-Code (Structured Query Language) erzeugt welcher aus den üblichen Befehlen zum Erzeugen und Befüllen von Tabellen besteht. Dieser Code wurde möglichst Datenbank unspezifisch gehalten und erlaubt es Datenbanken wie SQLite, MySQL oder PostgreSQL direkt mit den Daten zu befüllen. Besonders geeignet war hier die SQLite-Datenbank. Deren gute Unterstützung für Pipes im Terminal und der “eine Datenbank ist eine Datei” Ansatz integriert sich bestens in den normalen Arbeitsprozeß von Bioinformatikern und Biologen.

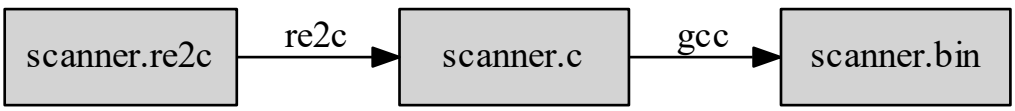

Figure 1: Übersetzung der Re2c-Deklarationen in eine Programm-Datei

4. Re2c BLAST-Scanner

Eines der am häufigsten verwendeten Tools in der Bioinformatik zur Sequenz-Analyse ist BLAST [1]. Mit BLAST werden Sequenz-Vergleiche und Sequenz-Suchen in großen Datenbanken durchgeführt. Über Ähnlichkeiten werden funktionelle Bestimmungen vorgenommen oder evolutionäre Zusammenhänge hergestellt. Bei größeren Genomen, z.B. von Säugern, können dabei erhebliche Datenmengen entstehen. Mit dem Re2c BLAST-Scanner konnten solche Sequenz-Vergleiche und funktionelle Untersuchungen sehr schnell durchgeführt werden, siehe zum Beispiel Schedina et al. [3, 4]. Selbst gigabyte-große Dateien können mit dem BLAST-Scanner in wenigen Sekunden in Datenbank-Code übersetzt und dann in Datenbanken exportiert werden. Die nachfolgende Analyse mit Hilfe der Datenbank ermöglichte dann die strukturierte Auswertung in einer standardisierten Abfrage-Sprache in dem hochperformanten System einer SQL-Datenbank.

5. Schlussfolgerungen

Das Projekt war eines meiner ersten und wenigen C Programmier-Projekte. Die Möglichkeit einer extrem schnellen Datenanalyse mit nachfolgender Nutzung einer etablierten Abfrage-Sprache (SQL) schien mir der geeignete Weg für bioinformatische Analysen. Leider ist jedoch die Kenntnis von SQL in der Wissenschaft eher marginal. Die wenigen Wissenschaftler, welche dieses Tool in Kooperationen mit mir nutzten sind mittlerweile, wie im Wissenschaftsbereich nicht unüblich, in andere Arbeitsbereiche abgewandert. Die teilweise extreme Kurzzeitigkeit von Zielstellungen in der heutigen Wissenschaft steht der Durchsetzung von etablierten, älteren Techniken leider entgegen. Ich selbst verwende diesen Ansatz immer wieder in kleineren Projekten. Mein persönlicher Arbeitsbereich ist jedoch heute mehr im Bereich der Lehre und statistischen Datenanalyse angesiedelt, weswegen der hoch-performante BLAST-Scanner zumeist seinen wohlverdienten Dornrösschen-Schlaf genießt.

Literatur

[1] S. ALTSCHUL, W. GISH, W. MILLER, E. MYERS, D. LIPMAN, Sequence homology searches done using BLAST. *J Mol Biol* **215** (1990), 404–415.

[2] J. P. BUCHMANN, M. FOURMENT, E. C. HOLMES, The Biological Object Notation (BON): a structured file format for biological data. *Scientific reports* **8** (2018) 1, 1–8.

[3] D. GROTH, S. HARTMANN, G. PANOPOULOU, A. J. POUSTKA, S. HENNIG, GOblet: annotation of anonymous sequence data with gene ontology and pathway terms. *Journal of Integrative Bioinformatics* **5** (2008) 2, 208–220.

[4] I. M. SCHEDINA, D. GROTH, I. SCHLUPP, R. TIEDEMANN, The gonadal transcriptome of the unisexual Amazon molly Poecilia formosa in comparison to its sexual ancestors, Poecilia mexicana and Poecilia latipinna. *BMC genomics* **19** (2018) 1, 1–18.

[5] U. TROFIMOVICH, RE2C: A lexer generator based on lookahead-TDFA. *Software Impacts* **6** (2020), 100027.

THEORIE-AG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 17–20.

Regular Languages in Database Theory – Document Spanners and Regular Path Queries

Markus L. Schmid[(A)]

[(A)]Humboldt-Universität zu Berlin
MLSchmid@MLSchmid.de

In the field of database theory, the term *query evaluation* usually refers to the following algorithmic setting: We consider a class of possible databases (e. g., relational databases, documents, data graphs) and a class of queries for such databases (e. g., relational algebra (for relational data), path queries (for data graphs), document spanners (for documents)). While this is a classical field of research in database theory (see, e. g., [11]), the contemporary interest has shifted from classical complexity analyses (e. g., NP-completeness vs. polynomial time algorithms for decision problems) to a more fine-grained setting, which is mainly entailed by the following three particularities (see [4, 12, 8, 2] for some recent publications).

Enumeration: After some preprocessing phase, enumerate all elements of the solution set (without repetition) with a worst-case upper bound on the delay (i. e., the time between the output of two consecutive elements of the result set). This point of view is justified whenever the solution set may be large, since if in this case we measure the overall computation time, then the size of the solution set (which, in most settings, is potentially of exponential size) is always a trivial lower bound.

Data Complexity: We usually assume that the size of the query is rather small, while the data is very large. This assumption is justified by practical scenarios: that the size of the data is large is obviously to be expected (especially in the big-data era), and the assumption that queries are not excessively large follows from the fact that they are usually formulated by (and have to be understood by) human users. The so-called *data complexity* perspective therefore completely neglects the query-size (or, equivalently, assumes the query to be fixed and thus of constant size), and measures complexity only in terms of the size of the data. For example, in this setting, we would prefer a running time $O(|q|^8|D|)$ over a running time $O(|q| + |D|\log(|D|))$, where q is the query and D is the database.

Dynamic Setting: Instead of treating every evaluation of a query over a database as a new problem instance, we assume that the same few queries are evaluated over a single database, which may slightly change over time by suitable updates. In particular, we are interested in algorithms that are optimised for this setting, i. e., algorithms that exploit the fact that we have already evaluated our query over a database that only slightly differs from the current one. More precisely, in the dynamic setting, we have one database, which may be updated (e. g., adding or deleting data). For any given query q, we may compute in a preprocessing phase some data structures that allow us to evaluate this query over our database (where evaluation often means enumeration of the solution set, as described above). However, when the database is updated,

instead of re-running the whole preprocessing for q, we wish to update our data structures for the evaluation of q directly and significantly faster than computing them from scratch.

These three objectives are also investigated in other branches of algorithmic research, but it is a unique feature of query evaluation in database theory that we are interested in evaluation algorithms that are tailored to all three of these aspects at the same time, i. e., algorithms that enumerate the solution set, that support certain updates to the data, and are optimised with respect to the data complexity perspective. The optimum here are enumeration algorithms that have a linear preprocessing (in data complexity, meaning a running time that only depends linearly on the data size), a constant delay (again in data complexity, meaning a delay bound that *only* depends on the size of the query), and that can perform update-procedures with constant or logarithmic running time.

It can be easily seen that this algorithmic paradigm leads to rather challenging and new problems, even for data domains that are heavily investigated for a long time. In fact, contemporary questions from database theory also identify new and relevant algorithmic questions about rather classical concepts from formal languages, like regular languages and the corresponding description mechanisms. In this talk, I will give a brief overview of two query evaluation settings that are based on regular languages: (1) Document spanners, which is a query class for textual data (i. e., strings), and (2) regular path queries, which is an interpretation of regular expressions as queries for graph databases (i. e., directed, arc-labelled graphs).

Document spanners have been introduced quite recently and are heavily investigated in database theory (see, e. g., [6, 1, 9] and [10] for a survey). The main idea is to use regular expressions or finite automata in order to compile a table of so-called span-tuples from a string (a span is just an interval of the string, or, equivalently, a factor of the string represented by its start and end position), and then further manipulate this relational data by operators of relational algebra. A special such operation is the string-equality selection, which selects certain span-tuples according to whether some spans refer to different occurrences of the same factor.

It is known that spanners that are strictly regular (in the sense that they are completely described by a finite automaton) have excellent algorithmic properties [6, 1], while the addition of the string equality selection moves most relevant computational problems into the realm of intractability or even undecidability [7]. We show in [9] that by slightly modifying the model, we can obtain a class of spanners that in terms of expressive power are rather close to the most powerful class of spanners that can use the string equality selection, while in terms of algorithmic properties they are still rather close to the purely regular spanners.

The concept of regular path queries (RPQs) dates back to the beginnings of graph databases and since then they are of continuous interest in applied and theoretical research about graph databases (see [3] for a survey). In their most common abstraction, graph databases are just arc-labelled and directed graphs. In their most simple formulation, an RPQ is just a regular expression, and applied to a graph database it returns all pairs of nodes that are connected by a path labelled with a word that matches the regular expression.

It is conceivable (and well-documented in the literature) that a general technique for evaluating an RPQ q is to apply the cross-product construction with respect to an automaton for q and the graph database (interpreted as an automaton). All relevant RPQ evaluation problems can then be solved by basic graph searching techniques on this structure. In [5], we investigate the question whether this classical *cross-product approach* also yields optimal algorithms for RPQ evaluation problems.

References

[1] A. AMARILLI, P. BOURHIS, S. MENGEL, M. NIEWERTH, Constant-Delay Enumeration for Nondeterministic Document Spanners. *ACM Trans. Database Syst.* **46** (2021) 1, 2:1–2:30.
https://doi.org/10.1145/3436487

[2] A. AMARILLI, L. JACHIET, M. MUÑOZ, C. RIVEROS, Efficient Enumeration for Annotated Grammars. In: *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. 2022, 291–300.
https://doi.org/10.1145/3517804.3526232

[3] P. BARCELÓ, Querying graph databases. In: *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*. 2013, 175–188.

[4] C. BERKHOLZ, F. GERHARDT, N. SCHWEIKARDT, Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News* **7** (2020) 1, 4–33.
https://doi.org/10.1145/3385634.3385636

[5] K. CASEL, M. L. SCHMID, Fine-Grained Complexity of Regular Path Queries. In: *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus*. 2021, 19:1–19:20.
https://doi.org/10.4230/LIPIcs.ICDT.2021.19

[6] R. FAGIN, B. KIMELFELD, F. REISS, S. VANSUMMEREN, Document Spanners: A Formal Approach to Information Extraction. *J. ACM* **62** (2015) 2, 12:1–12:51.
https://doi.org/10.1145/2699442

[7] D. D. FREYDENBERGER, M. HOLLDACK, Document Spanners: From Expressive Power to Decision Problems. *Theory of Computing Systems (ToCS)* (2017). https://doi.org/10.1007/s00224-017-9770-0.

[8] M. NIEWERTH, L. SEGOUFIN, Enumeration of MSO Queries on Strings with Constant Delay and Logarithmic Updates. In: *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*. 2018, 179–191.
https://doi.org/10.1145/3196959.3196961

[9] M. L. SCHMID, N. SCHWEIKARDT, A Purely Regular Approach to Non-Regular Core Spanners. In: *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus*. 2021, 4:1–4:19.
https://doi.org/10.4230/LIPIcs.ICDT.2021.4

[10] M. L. SCHMID, N. SCHWEIKARDT, Document Spanners - A Brief Overview of Concepts, Results, and Recent Developments. In: *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. 2022, 139–150.
https://doi.org/10.1145/3517804.3526069

[11] N. SCHWEIKARDT, T. SCHWENTICK, L. SEGOUFIN, *Database Theory: Query Languages*. 2 edition, Chapman & Hall/CRC, 2010, 19.

[12] N. SCHWEIKARDT, L. SEGOUFIN, A. VIGNY, Enumeration for FO Queries over Nowhere Dense Graphs. *J. ACM* **69** (2022) 3, 22:1–22:37.
https://doi.org/10.1145/3517035

THEORIE-TAG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 21–24.

Prescribed Teams of Rules Working in Parallel on Different Objects

Artiom Alhazov[(A)] Rudolf Freund[(B)] Sergiu Ivanov[(C)]
Sergey Verlan[(D)]

[(A)]Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[(B)]Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

[(C)]IBISC, Univ. Évry, Paris-Saclay University
23, boulevard de France 91034 Évry, France
sergiu.ivanov@ibisc.univ-evry.fr

[(D)]Univ. Paris Est Creteil, LACL, 94010, Creteil, France
verlan@u-pec.fr

Abstract

In this paper we consider prescribed sets of rules working in parallel on several objects. We show that systems of degree three and prescribed teams of size two are sufficient to obtain computational completeness for strings with the simple rules being of the form $aI_R(b)$, meaning that a symbol b can be inserted on the right-hand side of a string ending with a, and $D_R(b)$, meaning that a symbol b is erased on the right-hand side of a string.

1. Preliminaries

Many control mechanisms for guiding the application of rules have been studied in the previous decades, see [2], e.g., matrix grammars forcing sequences of rules to be applied. In a less strict variant, the rules of a group must be applied together, but the order of their application is not imposed; this control mechanism is known as *prescribed teams* and was introduced in [1].

In this paper[1], we define prescribed teams in a general framework for rewriting as a control mechanism over abstract rules, but with the rules in a prescribed team to be applied to several

[(A)] Artiom Alhazov acknowledges project 20.80009.5007.22 "Intelligent information systems for solving ill-structured problems, processing knowledge and big data" by the National Agency for Research and Development.

[1] A longer version has been accepted for MCU 2022 taking place in Debrecen at he beginning of September.

objects in parallel. For the results on string rewriting we will only use certain insertion and deletion rules.

For notions and results in formal language theory we refer to textbooks like [2] and [3].

The main model we consider in this paper is a system of arbitrary objects which starts on a set of d such objects and has prescribed teams of size $s \leq d$ with the rules of each team working on the d objects derived so far until no such team can be applied any more.

Definition 1.1 *A* (homogenous) system with prescribed teams of rules of size s and degree d *is a construct* $G = (O, O_T, P, R, A)$ *where* O *is a set of objects;* $O_T \subseteq O$ *is a set of* terminal objects*;* P *is a finite set of* rules*, i.e.,* $P = \{p_i \mid 1 \leq i \leq m\}$*, for some* $m \geq 0$*, and* $p_i \subseteq O \times O$*;* $R = \{T_1, \ldots, T_n\}$ *is a finite set of sets of rules from* P *called* prescribed teams*, i.e.,* $T_i \subseteq P$*,* $1 \leq i \leq n$*, with* $|T_i| = s$*;* A *is a finite set of* d *initial objects in* O*.*

A rule $p \in P$ is called *applicable* to an object $x \in O$ if and only if there exists at least one object $y \in O$ such that $(x, y) \in p$; in this case we also write $x \Longrightarrow_p y$.

A computation in G starts with the initial object being the set A. A computation step means the application of a prescribed team of rules to the current configuration being a set of d objects. At the beginning of a computation step we first have to choose a suitable team T_k; each rule in T_k is applied to a different object in the current set of objects; T_k can only be applied if every rule in T_k can be applied. A *derivation step* in G using T_k can be written as $\{O_1, \ldots, O_d\} \Longrightarrow_{T_k} \{O'_1, \ldots, O'_d\}$ where $\{O_1, \ldots, O_d\}$ is the current set of objects and $\{O'_1, \ldots, O'_d\}$ is the set of objects obtained after the application of T_k. The derivation relation $\Longrightarrow_G$ of the system G then is the union of all derivation relations $\Longrightarrow_{T_k}$, $1 \leq k \leq n$. The reflexive and transitive closure of $\Longrightarrow_G$ is denoted by $\Longrightarrow^*_G$.

The computational model we will simulate for showing computational completeness for the systems with prescribed teams of rules defined above are Turing machines with one tape with left boundary marker Z_0:

Definition 1.2 *A Turing machine is a construct* $M = (Q, V, T_1, T_2, \delta, q_0, q_1, Z_0, B)$ *where* Q *is a finite set of* states*,* V *is the* tape alphabet*,* $T_1 \subseteq V \setminus (\{Z_0, B\})$ *is the* input alphabet*,* $T_2 \subseteq V \setminus (\{Z_0, B\})$ *is the* output alphabet*,* $\delta \subseteq (Q \times V) \to (Q \times V \times \{L, R\})$ *is the* transition function*,* q_0 *is the* initial state*,* q_1 *is the* final state*,* $Z_0 \in V$ *is the* left boundary marker*,* $B \in V$ *is the* blank symbol*. A* transition *between configurations is carried out according to the transition function* δ *in the following way for* $(q, X; p, Y, D) \in \delta$*: the state changes from* q *to* p*; the symbol* X *currently read is replaced by the symbol* Y*; for* $D = R$ *the read-write head goes one step to the right and for* $D = L$ *the read-write head goes one step to the left; observe that the read-write head can never go to the left of* Z_0*.*

A *configuration* of the Turing machine M can be written as Z_0uqvB^ω, where $u, v \in (V \setminus (\{Z_0\}))^*$ and B^ω indicates the remaining infinite empty part of the tape, offering an unbounded number of tape cells initially carrying the blank symbol B; moreover, the current state $q \in Q$ is written to the right of the tape cell on which the read-write head of the Turing machine currently stands.

2. Results

In this section we consider the objects to be strings. Moreover, we will restrict ourselves to special variants of insertion an deletion rules to be applied to strings: $aI_R(b)$ means that a symbol b can be inserted on the right-hand side of a string ending with a and $D_R(b)$ means that a symbol b is erased on the right-hand side of a string.

Theorem 2.1 *The computations of a Turing machine M can be simulated by a homogenous string system with prescribed teams of size* 2 *and degree* 3 *using only rules of the form $aI_R(b)$ and $D_R(b)$.*

Proof. Let $M = (Q, V, T_1, T_2, \delta, q_0, q_1, Z_0, B)$ be a Turing machine. In order to represent the configurations of M as finite strings, we use a right end marker Z_1 to mark the end of a finite representation $Z_0uqvB^mZ_1$ of the configuration Z_0uqvB^ω, where m may be any natural number ≥ 0; m depends on how far on the tape the read-write head has already proceeded during a computation.

We now construct a system G with prescribed teams using only rules of the form $aI_R(b)$ and $D_R(b)$ which can simulate the computations of the given Turing machine M. The basic idea is folklore – a configuration $Z_0uqvB^mZ_1$ is represented by two strings Z_0uq and $(vB^mZ_1)^R = Z_1B^mv^R$, which like stacks are only affected at the end of the strings. A special technical detail is that when we reach a situation where the second string is Z_1, no transition to the right is possible immediately, we first have to insert an additional blank B to then continue with the second string being Z_1B. Moreover, in order to allow the rules to distinguish between the two strings, the second string is written in the primed alphabet $V' = \{X' \mid X \in V\}$.

For simulating the computations of M we construct the system

$$G = ((V \cup Q) \cup (V \cup Q)' \cup Q'')^*, \{Z_0\}T_2^*, P, R, A),$$
$$A = \{Z_0w_{input}q_0, Z_1', q_0\}.$$

The first two strings in a configuration represent the left and the right part of the tape of M, whereas the third string collects the labels of rules applied during a computation and thus guides the application of rules with insertion rules of the form $qI_R(p)$. At the end of a successful computation, the result of a successful computation w_{output} is given by the first string as the string Z_0w_{output}. The third string can be interpreted as the Szilard word of the computation.

The main idea for constructing the prescribed teams is that the rule $qI_R(p)$ is applied to the third string whereas the other one acts on one of the first two strings and can be looked at as the rule labeled by p. In that way, a kind of graph control can be implemented, where for every rule labeled by q the successor rules labeled by p can be specified. In which order the two rules are applied does not matter, as they have to be applied to different strings in the configuration.

The set of rules P can be collected from the prescribed teams of rules described in the following, and the intermediate states defined below are collected in Q'':

$(\mathbf{q}, \mathbf{X}; \mathbf{p}, \mathbf{Y}, \mathbf{L})$ is simulated by guessing the symbol $W \in V$ to the left of X and the symbol $U \in V$ at the end of the second string in a nondeterministic way and by applying $\{D_R(X), qI_R([W; q, X; p, Y, L; U])\}$ and $\{U'I_R(Y'), [W; q, X; p, Y, L; U]I_R(p)\}$.

With these two teams, we obtain the following derivation:

$\{uWX, Z_1{'}B'^m v'^R U', zq\} \Longrightarrow$

$\{uW, Z_1{'}B'^m v'^R U', zq[W;q,X;p,Y,L;U]\}\ Longrightarrow$

$\{uW, Z_1{'}B'^m v'^R U'Y', zq[W;q,X;p,Y,L;U]p\}$

$(\mathbf{q},\mathbf{X};\mathbf{p},\mathbf{Y},\mathbf{R})$ is simulated by guessing the symbol $W \in V$ to the left of X and the symbol $U \in V$ at the end of the second string in a nondeterministic way and by applying the following four teams in the indicated sequence:

$\{D_R(X), qI_R([W;q,X;p,Y,L;U])\}$;

$\{WI_R(Y), [W;q,X;p,Y,L;U]I_R([q,X;p,Y,L;U'])\}$;

$\{D_R(U'), [q,X;p,Y,L;U']I_R([q,X;p,Y,L;U''])\}$;

$\{YI_R(U), [q,X;p,Y,L;U'']I_R(p)\}$.

With these four teams, we obtain the following derivation:

$\{uWX, Z_1{'}B'^m v'^R U', zq\} \Longrightarrow$

$\{uW, Z_1{'}B'^m v'^R U', zq[W;q,X;p,Y,L;U]\} \Longrightarrow$

$\{uWY, Z_1{'}B'^m v'^R U', zq[W;q,X;p,Y,L;U][q,X;p,Y,L;U']\} \Longrightarrow$

$\{uWY, Z_1{'}B'^m v'^R, zq[W;q,X;p,Y,L;U][q,X;p,Y,L;U'][q,X;p,Y,L;U'']\} \Longrightarrow$

$\{uWYU, Z_1{'}B'^m v'^R, zq[W;q,X;p,Y,L;U][q,X;p,Y,L;U'][q,X;p,Y,L;U'']p\}$

insertion of B If one more blank is needed in front of the right end marker $Z_1{'}$, an intermediate step for any state q being part of a rule $(q,B;p,Y,R)$ to be applied is carried out:

$\{Z_1{'}I_R(B'), qI_R(q)\}$

Derivation:

$\{u, Z_1{'}, zq\} \Longrightarrow \{u, Z_1{'}B', zqq\}$

In sum, for any halting computation of M

$Z_0 w_{input} q_0 B^\omega \Longrightarrow_M^* Z_0 w_{output} q_1 B^\omega$

we obtain a corresponding halting computation in G

$\{Z_0 w_{input}, Z_1{'}, q_0\} \Longrightarrow_G^* \{Z_0 w_{output}, Z_1{'}B'^m, zq_1\}$.

A result $Z_0 w_{output}$ obtained in G represents the string w_{output}. Observe that Z_0 cannot be avoided as only non-empty strings can be handled by the insertion rules in G. □

References

[1] E. CSUHAJ-VARJÚ, J. DASSOW, J. KELEMEN, *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Topics in computer mathematics, Gordon and Breach, 1994.

[2] J. DASSOW, GH. PĂUN, *Regulated Rewriting in Formal Language Theory*. Springer, 1989.

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

THEORIE-AG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 25–28.

Compositions with Constant Weighted Extended Tree Transducers

Malte Blattmann[(A)] Andreas Maletti[(B)]

[(A)]Universität Leipzig, Innovation Center Computer Assisted Surgery
Semmelweisstraße 14, 04103 Leipzig, Germany
malteblattmann@gmx.de

[(B)]Universität Leipzig, Faculty of Mathematics and Computer Science
PO Box 100 920, 04009 Leipzig, Germany
andreas.maletti@uni-leipzig.de

Abstract

Conjecture 11 of [Lagoutte, Maletti: Survey — Weighted extended top-down tree transducers — Part III: Composition. Proc. AFCS, LNCS 7020, p. 272–308, Springer 2011] is confirmed. It is demonstrated that the composition of a constant weighted extended tree transducer with a linear weighted top-down tree transducer can be computed by a single weighted extended tree transducer. This research was published as [Blattmann, Maletti: Compositions of constant weighted extended tree transducers. Proc. DLT, LNCS 12811, p. 66–77, Springer 2021].

1. Introduction

In this contribution we investigate compositions of certain weighted extended tree transducers [5]. Given weighted relations $\tau\colon T_\Gamma \times T_\Sigma \to S$ and $\tau'\colon T_\Sigma \times T_\Delta \to S$ with weights in a commutative semiring $(S, +, \cdot, 0, 1)$ [3], their composition is the weighted relation $\tau;\tau'\colon T_\Gamma \times T_\Delta \to S$ given for every $g \in T_\Gamma$ and $u \in T_\Delta$ by

$$(\tau;\tau')(g,u) = \sum_{t \in T_\Sigma} \tau(g,t) \cdot \tau'(t,u) \ , \tag{1}$$

where we note that, in general, the sum might be infinite. It is well-known that a representation of the composed weighted relation (both for speech recognition as well as for machine translation) by just a single transducer offers significant advantages [6]. We continue the investigation into compositions of weighted extended tree transducers started in [4] and confirm its Conjecture 11. To this end, we show how to compose an arbitrary constant weighted extended tree

[(B)]The author gratefully acknowledges the financial support of the DFG graduate research training group 1763: "Quantitative Logics and Automata".

transducer with a linear weighted top-down tree transducer. The constant property requires that for each state p there exists a constant c_p that is the total weight of all translations in p for any input tree. In particular, c_p needs to be independent of the input tree, so the total weight of all translations needs to coincide for all input trees. Despite its somewhat complicated description, the constant property is very natural and all stochastic tree transducers (over the rational numbers) and all total tree transducers (over the BOOLEAN semiring) are constant (in both cases with $c_p = 1$ for every state p). The second tree transducer needs to be linear (i.e., is not allowed to process the same input subtree multiple times) and top-down (i.e., non-extended, so it processes only a single input symbol in each rule). However, we place no additional constraints on the utilized weight structure, so our construction works for any commutative semiring. Both main features of commutative semirings, namely commutativity and distributivity, are heavily utilized in our construction. The history and corresponding unweighted composition results are discussed at length in [4].

2. Weighted extended tree transducers

We use the variables $X = \{x_1, x_2, \dots\}$, its subsets $X_k = \{x_1, \dots, x_k\}$ for every $k \in \mathbb{N}$ as well as the set $T_\Sigma(X)$ of all trees over Σ and X, and its subset $\widehat{T}_\Sigma(X)$, which contains those trees, in which the variables of X occur in the order $x_1, \dots, x_n$, when read left-to-right, for some $n \in \mathbb{N}$. As usual, $\mathrm{var}(t)$ denotes the set of variables that actually occur in t.

Definition 2.1 (see Definition 3.1 of [2]) *A* weighted extended tree transducer *(for short: wxtt) is a tuple $(Q, \Sigma, \Delta, Q_0, R)$, in which*

- *Q is a finite set of* states
- *Σ and Δ are ranked alphabets of* input *and* output symbols*, respectively,*
- *$Q_0 \subseteq Q$ is a set of* initial states*, and*
- *$R \subseteq Q \times \widehat{T}_\Sigma(X) \times S \times \widehat{T}_\Delta(X) \times (Q \times X)^*$ is a finite set of* weighted rules *such that $\ell \neq x_1$ and $w \in \big(Q \times \mathrm{var}(\ell)\big)^{\mathrm{var}(r)}$ for every $\langle q, \ell, s, r, w\rangle \in R$.*

Let $M = (Q, \Sigma, \Delta, Q_0, R)$ be a wxtt. We present a rule $\big\langle q, \ell, s, r, (q_1, y_1) \cdots (q_n, y_n)\big\rangle \in R$ as $q(\ell) \xrightarrow{s} r\theta$, where the substitution $\theta\colon \mathrm{var}(r) \to Q\big(\mathrm{var}(\ell)\big)$ is given by $\theta(x_i) = q_i(y_i)$ for all $1 \leq i \leq n$. The weighted rule $\big\langle q, \gamma(x_1), 1, \nu(x_1), (q', x_1)\big\rangle$ is thus also $q\big(\gamma(x_1)\big) \xrightarrow{1} \nu\big(q'(x_1)\big)$. The wxtt M is *linear* if $|\{y_1, \dots, y_n\}| = n$ for every rule $\big\langle q, \ell, s, r, (q_1, y_1) \cdots (q_n, y_n)\big\rangle \in R$. In other words, for every weighted rule $\langle q, \ell, s, r, w\rangle \in R$ the variables in the range of w may not repeat and thus $|\mathrm{var}(r)| \leq |\mathrm{var}(\ell)|$. Finally, M is a *weighted top-down tree transducer* (for short: wtdtt), if for every rule $\langle q, \ell, s, r, w\rangle \in R$ there exist $k \in \mathbb{N}$ and symbol $\sigma \in \Sigma_k$ such that $\ell = \sigma(x_1, \dots, x_k)$. Finally, given $c \in \widehat{T}_\Sigma(X)$ and $t \in T_\Sigma(X)$ we let

$$\mathrm{match}(c, t) = \big\{\theta\colon \mathrm{var}(c) \to T_\Sigma(X) \mid c\theta = t\big\}$$

be the set of substitutions that matches c to t. Note that $|\mathrm{match}(c, t)| \leq 1$.

Definition 2.2 (see Definition 3.2 of [2]) *For every $V \subseteq X$, state $q \in Q$, and $\varphi\colon V \to Q \times X$*

we define the mapping $\mathrm{h}_M^{q,\varphi} \colon T_\Sigma(X) \times T_\Delta(X) \to S$ *for every* $t \in T_\Sigma(X)$ *and* $u \in T_\Delta(X)$ *by*

$$\mathrm{h}_M^{q,\varphi}(t,u) = \begin{cases} 1 & \text{if } u \in V \text{ and } \varphi(u) = (q,t) \\ \displaystyle\sum_{\substack{\langle q,\ell,s,r,w\rangle \in R \\ \theta \in \mathrm{match}(\ell,t) \\ \theta' \in \mathrm{match}(r,u)}} s \cdot \prod_{\substack{x \in \mathrm{var}(r) \\ w(x)=(z,y)}} \mathrm{h}_M^{z,\varphi}(y\theta, x\theta') & \text{otherwise} \end{cases} .$$

The semantics of M *is the* weighted relation $M \colon T_\Sigma \times T_\Delta \to S$ *with* $M(t,u) = \sum_{q \in Q_0} \mathrm{h}_M^{q,\emptyset}(t,u)$ *for every tree* $t \in T_\Sigma$ *and* $u \in T_\Delta$.

Definition 2.3 (see Definition 9 of [4]) *Let* $q \in Q$ *be a state and* $c \in S$ *be a constant. The state* q *is* c-constant *if* $c = \sum_{u \in T_\Delta} \mathrm{h}_M^{q,\emptyset}(t,u)$ *for every* $t \in T_\Sigma$. *The wxtt* M *is* constant *if for every state* $q \in Q$ *there exists* $c_q \in S$ *such that* q *is* c_q*-constant.*

3. Composition

Our goal is to settle Conjecture 11 of [4]. Given a constant wxtt M with states P and a linear wtdtt N with states Q, Conjecture 11 of [4] claims that the composition $M\,;N$ can be computed by a wxtt. The standard composition construction fails since it does not charge some weights, which can be addressed with the following approach. Whenever N deletes a subtree, we charge the weight c_p for every occurrence of state $p \in P$ in the deleted part. Since the wtdtt M' is linear, it will not copy subtrees and thus not duplicate occurrences of states $p \in P$. This motivates the following composition of the given wxtt. To facilitate notation, we define the set $\mathrm{decomp}(w)$ for every word $w \in (P \times Q \times X_n)^m$ with $m, n \in \mathbb{N}$.

$$\begin{aligned} \mathrm{decomp}(w) = \Big\{ \langle w', w'' \rangle \mid\ & w' = (p_1, y_1) \cdots (p_k, y_k) \in (P \times X_n)^k, \\ & w'' = (q_1, x_{j_1}) \cdots (q_m, x_{j_m}) \in (Q \times X_{|w'|})^m, \\ & \forall i \in \{1, \ldots, m\} \colon w_i = (p_{j_i}, q_i, y_{j_i}) \Big\} \end{aligned}$$

For $w = (p, q, x_1)$, $w' = (p, x_1)(p, x_1)$, and $w'' = (q, x_2)$ we have $\langle w', w'' \rangle \in \mathrm{decomp}(w)$.

Definition 3.1 (see Definition 6 of [4]) *Let* $M = (P, \Gamma, \Sigma, P_0, R')$ *be a constant wxtt as well as* $N = (Q, \Sigma, \Delta, Q_0, R'')$ *be a linear wtdtt. Additionally, let*

- $a = \max\{\mathrm{size}(\ell) \mid \langle p, \ell, s, r, w\rangle \in R'\}$ *be the maximal size of the left-hand sides of* M,
- $b = \max\{\mathrm{size}(r) \mid \langle p, \ell, s, r, w\rangle \in R'\}$ *be the maximal size of the right-hand sides of* M,
- $c_p \in S$ *be such that* p *is* c_p*-constant (in* M*) for every* $p \in P$, *and*
- $\mathrm{mx} \in \mathbb{N}$ *be the maximal size of an output tree of* N *for any input tree of size at most* b

$$\mathrm{mx} = \max\left\{ \mathrm{size}(u) \;\middle|\; (t,u) \in \bigcup_{\substack{q \in Q, V \subseteq X \\ \varphi \colon V \to Q \times X}} \mathrm{supp}(\mathrm{h}_N^{q,\varphi}), \mathrm{size}(t) \le b \right\} .$$

We define the composed wxtt $\underline{M\,;N} = (P \times Q, \Gamma, \Delta, P_0 \times Q_0, R)$ *with the following set* R *of weighted rules. For every state* $p \in P$, *state* $q \in Q$, *left-hand side* $\ell \in \widehat{T}_\Gamma(X)$ *of size* $\mathrm{size}(\ell) \le a$,

$r \in \widehat{T}_\Delta(X)$ *of size* $\mathrm{size}(r) \leq \mathrm{mx}$*, and word* $w = (p_1, q_1, y_1) \cdots (p_m, q_m, y_m) \in (P \times Q \times X_n)^m$ *with* $m = |\mathrm{var}(r)|$ *and* $n = |\mathrm{var}(\ell)|$ *we have* $\big\langle (p, q), \ell, s, r, w \big\rangle \in R$ *with*

$$s = \sum_{\substack{\langle p,\ell,s',r',w'\rangle \in R' \\ \langle w',w''\rangle \in \mathrm{decomp}(w) \\ w''=(q_1,x_{j_1})\cdots(q_m,x_{j_m})}} s' \cdot \mathrm{h}_N^{q,w''}(r', r) \cdot \prod_{\substack{x \in \mathrm{var}(r') \\ w'(x)=(p',z) \\ x \notin \{x_{j_1},\ldots,x_{j_m}\}}} c_{p'} \quad . \tag{2}$$

No other weighted rules are in R*.*

Theorem 3.2 *Let* M *be a constant wxtt and* N *be a linear wtdtt. Then the composed wxtt* $\underline{M ; N}$ *computes the weighted relation* $M ; N$*.*

Let us conclude with some final remarks. Obviously the relevant sums in (1) and (2) are finite in our contexts and thus well-defined. Moreover, it is straightforward to avoid the construction of many useless rules (i.e., rules with weight 0) in Definition 3.1. Indeed Definition 5 of [1] shows a much more direct construction. Overall, we have thus successfully confirmed Conjecture 11 of [4].

References

[1] M. BLATTMANN, A. MALETTI, Compositions of Constant Weighted Extended Tree Transducers. In: N. MOREIRA, R. REIS (eds.), *Proc. 25th Int. Conf. Developments in Language Theory*. Lecture Notes in Computer Science 12811, Springer, 2021, 66–77.

[2] Z. FÜLÖP, A. MALETTI, H. VOGLER, Weighted Extended Tree Transducers. *Fundamenta Informaticae* **111** (2011) 2, 163–202.

[3] J. S. GOLAN, *Semirings and their Applications*. Kluwer Academic Publishers, 1999.

[4] A. LAGOUTTE, A. MALETTI, Survey: Weighted extended top-down tree transducers — Part III: Composition. In: W. KUICH, G. RAHONIS (eds.), *Proc. Workshop Algebraic Foundations in Computer Science*. Lecture Notes in Computer Science 7020, Springer, 2011, 272–308.

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

[6] J. MAY, K. KNIGHT, H. VOGLER, Efficient inference through cascades of weighted tree transducers. In: J. HAJIČ, S. CARBERRY, S. CLARK, J. NIVRE (eds.), *Proc. 48th Ann. Meeting Association for Computational Linguistics*. ACL, 2010, 1058–1066.

THEORIE-AG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 29–34.

On the Generative Capacity of Contextual Grammars with Strictly Locally Testable Selection Languages

Jürgen Dassow[(A)] Bianca Truthe[(B)]

[(A)]Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg
Postfach 4120, 39106 Magdeburg, Germany
dassow@iws.cs.uni-magdeburg.de

[(B)]Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
bianca.truthe@informatik.uni-giessen.de

Abstract

We continue the research on the generative capacity of contextual grammars where contexts are adjoined around whole words (externally) or around subwords (internally) which belong to special regular selection languages. All languages generated by contextual grammars where all selection languages are elements of a certain subregular language family form again a language family. We investigate contextual grammars with strictly locally testable selection languages and compare those families to families which are based on finite, monoidal, nilpotent, combinational, definite, suffix-closed, ordered, commutative, circular, non-counting, power-separating, or union-free languages.

1. Introduction

Contextual grammars were introduced by Solomon Marcus in [8] as a formal model that might be used for the generation of natural languages. The derivation steps consist of adjoining contexts to given sentences starting from a finite set. A context is given by a pair (u, v) of words. The external adjoining to a word x gives the word uxv and the internal adjoining gives all words $x_1ux_2vx_3$ with $x_1x_2x_3 = x$. Following the linguistic motivation, conditions are given for each context which have to be met by the word in order to allow the context to be adjoined. Contextual grammars where the contexts are adjoined ex- or internally are called external or internal contextual grammars, respectively. If conditions are given to the subword where a context is to be adjoined, we speak about external or internal contextual grammars with selection. Contextual grammars with ex- or internal derivation and selection in a certain family F of languages were defined where it is required that the word where a context is wrapped around belongs to a language of the family F.

This is an extended abstract of a paper presented at NCMA 2022 ([6]).

The study of external contextual grammars with selection in special regular sets was started by Jürgen Dassow in [1]. The research was continued by Jürgen Dassow, Florin Manea, and Bianca Truthe (see [3, 4, 5, 7]) where further subregular families of selection languages were considered and the effect of subregular selection languages on the generative power of external and internal contextual grammars was investigated. A recent survey can be found in [9].

In [6], the impact of strictly locally testable selection languages in contextual grammars on the generative capacity is investigated. External contextual grammars with such selection languages have been studied already in [2].

2. Preliminaries

We consider the following restrictions for regular languages. Let L be a language over an alphabet V. We say that the language L – with respect to the alphabet V – is

- *monoidal* if and only if $L = V^*$,
- *nilpotent* if and only if it is finite or its complement $V^* \setminus L$ is finite,
- *combinational* if and only if it has the form $L = V^*X$ for some subset $X \subseteq V$,
- *definite* if and only if it can be represented in the form $L = A \cup V^*B$ where A and B are finite subsets of V^*,
- *suffix-closed* (or *fully initial* or *multiple-entry* language) if and only if, for any two words x and y over V, the relation $xy \in L$ implies the relation $y \in L$,
- *ordered* if and only if the language is accepted by some deterministic finite automaton

  $$\mathcal{A} = (V, Z, z_0, F, \delta)$$

  with an input alphabet V, a finite set Z of states, a start state $z_0 \in Z$, a set $F \subseteq Z$ of accepting states and a transition mapping δ where $(Z, \preceq)$ is a totally ordered set and, for any input symbol $a \in V$, the relation $z \preceq z'$ implies $\delta(z, a) \preceq \delta(z', a)$,
- *commutative* if and only if it contains with each word also all permutations of this word,
- *circular* if and only if it contains with each word also all circular shifts of this word,
- *non-counting* (or *star-free*) if and only if there is a natural number $k \geq 1$ such that, for any three words $x \in V^*$, $y \in V^*$, and $z \in V^*$, it holds $xy^kz \in L$ if and only if $xy^{k+1}z \in L$,
- *power-separating* if and only if, there is a natural number $m \geq 1$ such that for any word $x \in V^*$, either $J_x^m \cap L = \emptyset$ or $J_x^m \subseteq L$ where $J_x^m = \{\, x^n \mid n \geq m \,\}$,
- *union-free* if and only if L can be described by a regular expression which is only built by product and star,
- *strictly locally k-testable* if and only if there are three subsets B, I, and E of V^k such that any word $a_1a_2 \ldots a_n$ with $n \geq k$ and $a_i \in V$ for $1 \leq i \leq n$ belongs to the language L if and only if $a_1a_2 \ldots a_k \in B$, $a_{j+1}a_{j+2} \ldots a_{j+k} \in I$ for $1 \leq j \leq n-k-1$, and $a_{n-k+1}a_{n-k+2} \ldots a_n \in E$,
- *strictly locally testable* if and only if it is strictly locally k-testable for some natural number k.

Among the commutative, circular, suffix-closed, non-counting, and power-separating languages, we only consider those which are also regular.

By *FIN*, *MON*, *NIL*, *COMB*, *DEF*, *SUF*, *ORD*, *COMM*, *CIRC*, *NC*, *PS*, *UF*, SLT_k (for any natural number $k \geq 1$), *SLT*, and *REG*, we denote the families of all finite, monoidal, nilpotent,

combinational, definite, regular suffix-closed, ordered, regular commutative, regular circular, regular non-counting, regular power-separating, union-free, strictly locally k-testable, strictly locally testable, and regular languages, respectively.

Let F be a family of languages. A contextual grammar with selection in F is a triple $G = (V, \mathcal{P}, A)$ where

- V denotes an alphabet,
- $\mathcal{P}$ is a finite set of pairs (S, C) with a language S over some subset U of the alphabet V which belongs to the family F with respect to the alphabet U and a finite set $C \subset V^* \times V^*$,
- A denotes a finite subset of V^*.

The set V is called the basic alphabet; for a selection pair $(S, C) \in \mathcal{P}$, the language S is called a selection language and the set C is called a set of contexts of the grammar G; the elements of A are called axioms.

We now define the derivation modes for contextual grammars with selection.

Let $G = (V, \mathcal{P}, A)$ be a contextual grammar with selection. The external derivation relation $\underset{\text{ex}}{\Longrightarrow}$ is defined as follows: a word x derives a word y if and only if there is a pair $(S, C) \in \mathcal{P}$ such that $x \in S$ and $y = uxv$ for some pair $(u, v) \in C$. The internal derivation relation $\underset{\text{in}}{\Longrightarrow}$ is defined as follows: a word x derives a word y if and only if there are words $x_1, x_2, x_3 \in V^*$ such that $x = x_1x_2x_3$ and a pair $(S, C) \in \mathcal{P}$ such that $x_2 \in S$ and $y = x_1ux_2vx_3$ for some pair $(u, v) \in C$.

By $\overset{*}{\underset{\alpha}{\Longrightarrow}}$ we denote the reflexive and transitive closure of the derivation relation $\underset{\alpha}{\Longrightarrow}$ for $\alpha \in \{\text{ex}, \text{in}\}$. The language generated externally or internally by the grammar G is defined as

$$L_\alpha(G) = \left\{ z \mid x \overset{*}{\underset{\alpha}{\Longrightarrow}} z \text{ for some } x \in A \right\}$$

for $\alpha \in \{\text{ex}, \text{in}\}$. If the derivation mode is known from the context, we omit the index α. For a family $\mathfrak{L}$ of languages, we denote by $\mathcal{EC}(\mathfrak{L})$ and $\mathcal{IC}(\mathfrak{L})$ the family of all languages generated externally and internally, respectively, by contextual grammars with selection in $\mathfrak{L}$ (where all selection languages belong to the family $\mathfrak{L}$).

3. Results

For external contextual grammars, we have the following result (for proofs, we refer to [2, 6] and earlier literature cited there).

Theorem 3.1 *The inclusion relations presented in Figure 1 hold. An arrow from an entry X to an entry Y depicts the proper inclusion $X \subset Y$; the dashed arrow from $\mathcal{EC}(ORD)$ to $\mathcal{EC}(NC)$ indicates that it is not known so far whether the inclusion is proper or whether equality holds. With exception of the pairs $(\mathcal{EC}(ORD), \mathcal{EC}(SLT))$ and $(\mathcal{EC}(ORD), \mathcal{EC}(SLT_k))$ for $k \geq 2$, if two families X and Y are not connected by a directed path, then X and Y are incomparable; in the exceptional cases, $\mathcal{EC}(ORD) \not\subseteq \mathcal{EC}(SLT))$ and $\mathcal{EC}(ORD) \not\subseteq \mathcal{EC}(SLT_k))$ for $k \geq 2$ hold.*

For internal contextual grammars, we have the following result (for proofs, we refer to [6]

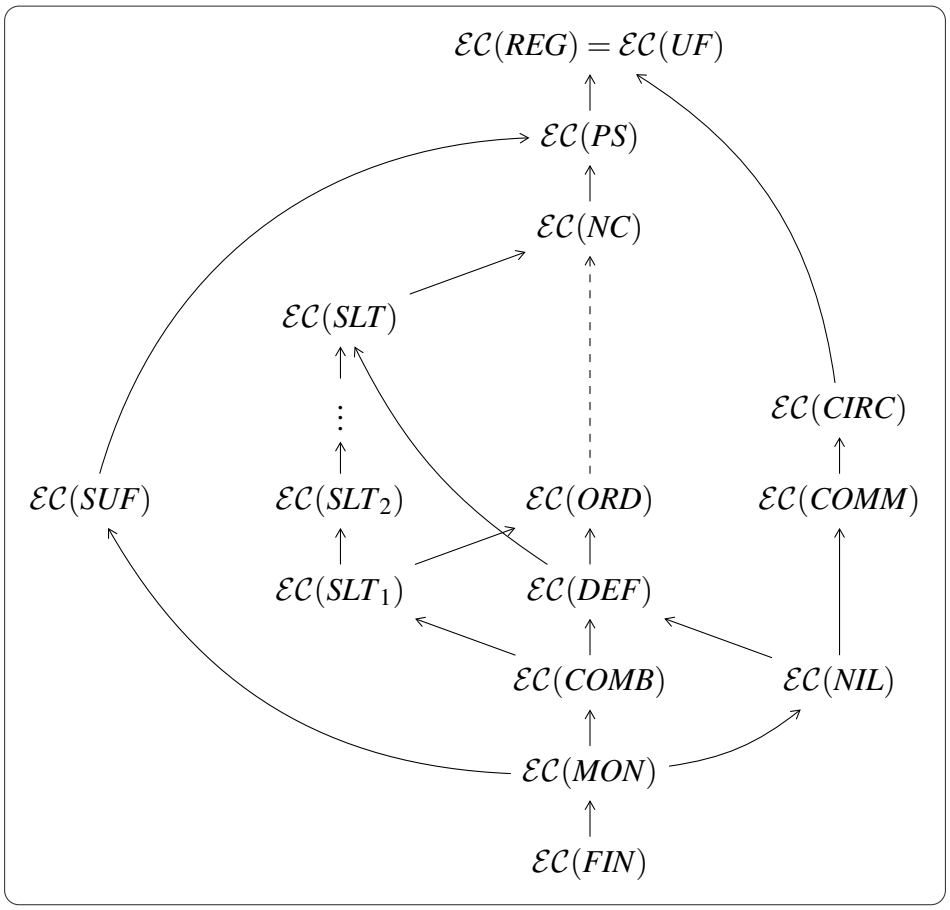

Figure 1: Hierarchy of language families by external contextual grammars with selection languages defined by structural properties.

and earlier literature cited there).

Theorem 3.2 *The inclusion relations presented in Figure 2 hold. An arrow from an entry X to an entry Y depicts the proper inclusion $X \subset Y$; the dashed arrow from $\mathcal{IC}(ORD)$ to $\mathcal{IC}(NC)$ indicates that it is not known so far whether the inclusion is proper or whether equality holds. If two families are not connected by a directed path, then they are incomparable with the exception of the family $\mathcal{IC}(SUF)$ and the families $\mathcal{IC}(ORD)$, $\mathcal{IC}(NC)$, and $\mathcal{IC}(SLT)$ where*

$$\mathcal{IC}(ORD) \not\subseteq \mathcal{IC}(SUF), \quad \mathcal{IC}(NC) \not\subseteq \mathcal{IC}(SUF), \quad \text{and} \quad \mathcal{IC}(SLT) \not\subseteq \mathcal{IC}(SUF)$$

hold, and with exception of the family $\mathcal{IC}(ORD)$ and the families $\mathcal{IC}(SLT)$ and $\mathcal{IC}(SLT_k)$ for $k \geq 2$, where

$$\mathcal{IC}(ORD) \not\subseteq \mathcal{IC}(SLT_k) \text{ for } k \geq 2 \text{ and } \mathcal{IC}(ORD) \not\subseteq \mathcal{IC}(SLT)$$

hold.

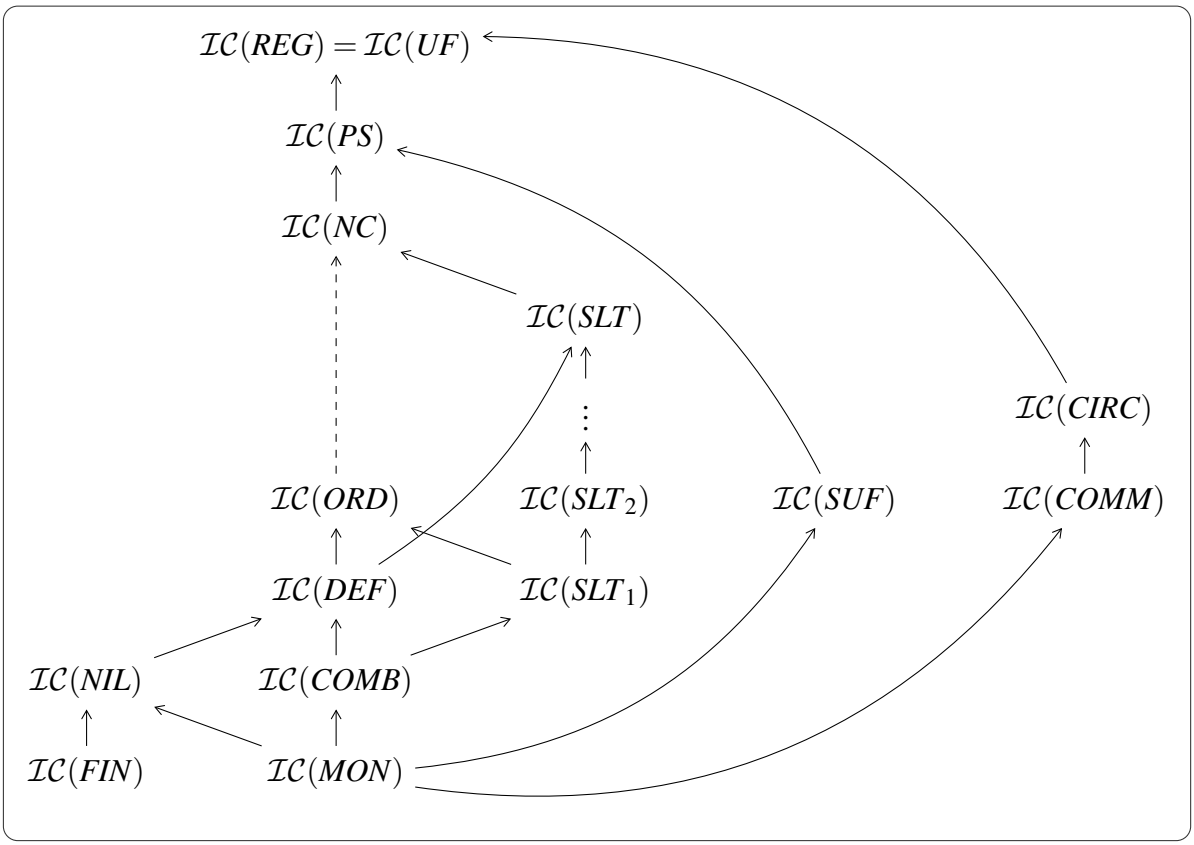

Figure 2: Hierarchy of language families by internal contextual grammars with selection languages defined by structural properties.

References

[1] J. DASSOW, Contextual grammars with subregular choice. *Fundamenta Informaticae* **64** (2005) 1–4, 109–118.

[2] J. DASSOW, Contextual languages with strictly locally testable and star free selection languages. *Analele Universității București* **62** (2015), 25–36.

[3] J. DASSOW, F. MANEA, B. TRUTHE, On Contextual Grammars with Subregular Selection Languages. In: M. HOLZER, M. KUTRIB, G. PIGHIZZINI (eds.), *Descriptional Complexity of Formal Systems – 13th International Workshop, DCFS 2011, Gießen/Limburg, Germany, July 25 – 27, 2011. Proceedings*. LNCS 6808, Springer-Verlag, 2011, 135–146.

[4] J. DASSOW, F. MANEA, B. TRUTHE, On External Contextual Grammars with Subregular Selection Languages. *Theoretical Computer Science* **449** (2012) 1, 64–73.

[5] J. DASSOW, F. MANEA, B. TRUTHE, On Subregular Selection Languages in Internal Contextual Grammars. *Journal of Automata, Languages, and Combinatorics* **17** (2012) 2–4, 145–164.

[6] J. DASSOW, B. TRUTHE, On the Generative Capacity of Contextual Grammars with Strictly Locally Testable Selection Languages. In: *Non-conventional Models of Automata, NCMA 2022, Debrecen, Hungary, August 26 – 27, 2022. Proceedings*. 2022.

[7] F. MANEA, B. TRUTHE, On Internal Contextual Grammars with Subregular Selection Languages. In: M. KUTRIB, N. MOREIRA, R. REIS (eds.), *Descriptional Complexity of Formal Systems – 14th International Workshop, DCFS 2012, Braga, Portugal, July 23 – 25, 2012. Proceedings*. LNCS 7386, Springer-Verlag, 2012, 222–235.

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

[9] B. TRUTHE, Generative Capacity of Contextual Grammars with Subregular Selection Languages. *Fundamenta Informaticae* **180** (2021), 1–28.

THEORIE-AG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 35–37.

Formal Languages via Theories over Strings

Joel Day[(A)] Vijay Ganesh[(B)] Nathan Grewal[(B)] Florin Manea[(C)]

[(A)]Loughborough University, United Kingdom
J.Day@lboro.ac.uk

[(B)]University of Waterloo, Canada
{vijay.ganesh,negrewal}@uwaterloo.ca

[(C)]University of Göttingen, Germany
florin.manea@cs.uni-goettingen.de

1. Theories over Strings

In this work, motivated by the ongoing work in the area of string solving, we investigate the properties of formal languages expressible in terms of formulas over quantifier-free theories over strings with concatenation (CON), word equations (WE), arithmetic over length constraints (LEN), and language membership predicates for the classes of regular (REG), visibly push down (VPL), and deterministic context-free languages (DCF). In total, we consider 20 theories, as well as decidability questions for problems such as emptiness or universality for formal languages over them.

Firstly, we discuss their relative expressive power and observe a rough division into two hierarchies based on whether or not word equations are present. Based on this, we derive the decidability status of several decision problems, such as emptiness, finiteness, and universality. Note that the emptiness problem is equivalent to the satisfiability problem over the corresponding theory. Secondly, we consider the problem of whether a language in one theory is expressible in another and show several undecidability results. These results seem particularly relevant in the context of normal forms in both practical and theoretical aspects of string solving.

The landscape resulting from the literature and our novel results is succinctly presented in Figures 1,2, and 3. For more details, please see the Arxiv version of our work at https://doi.org/10.48550/arXiv.2205.00475.

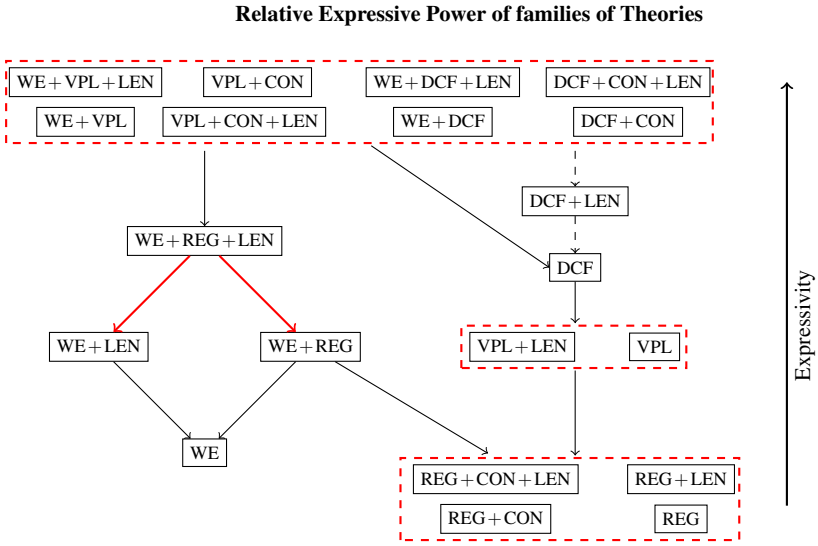

Figure 1: Visual representation of all 20 families of string-based logical theories considered in our work. Theory-families are depicted in solid square boxes containing the name of that theory. Dashed red square-boxes around multiple theory-families show equivalence w.r.t. the class of expressible languages: these are all novel results. For instance, we show that the most expressive group of theories (i.e. those equivalent to VPL + CON) are able to express all recursively enumerable languages. The arrows between theory-families and their transitive closure represent inclusion w.r.t. the class of expressible languages. For example, the arrow between WE + REG and WE indicates that all languages expressible in WE are expressible in WE + REG. Solid arrows indicate that the inclusion is known to be strict (the red arrows highlight novel results), while dashed arrows indicate that we do not know whether the inclusion is strict or not.

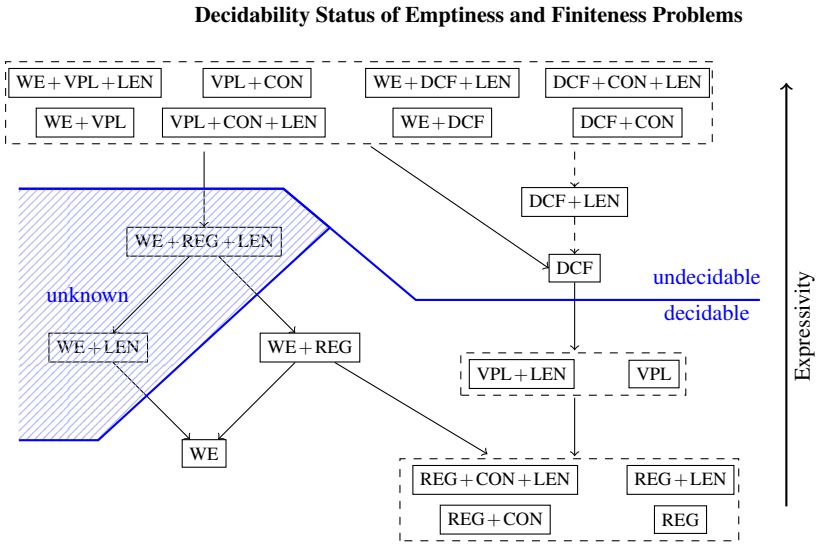

Figure 2: Indication for which (families of) theories the emptiness and finiteness problems are decidable or undecidable. Based on our results on the classes of expressible languages in each theory, we obtain, for instance, undecidability for VPL + CON. This is a particularly interesting case because it is only a slight generalisation of important cases which remain open, namely WE + REG + LEN and WE + LEN (shaded). We also show decidability for VPL + LEN, REG + CON + LEN etc. by showing equivalence to families for which decidability follows directly from known results (REG and VPL).

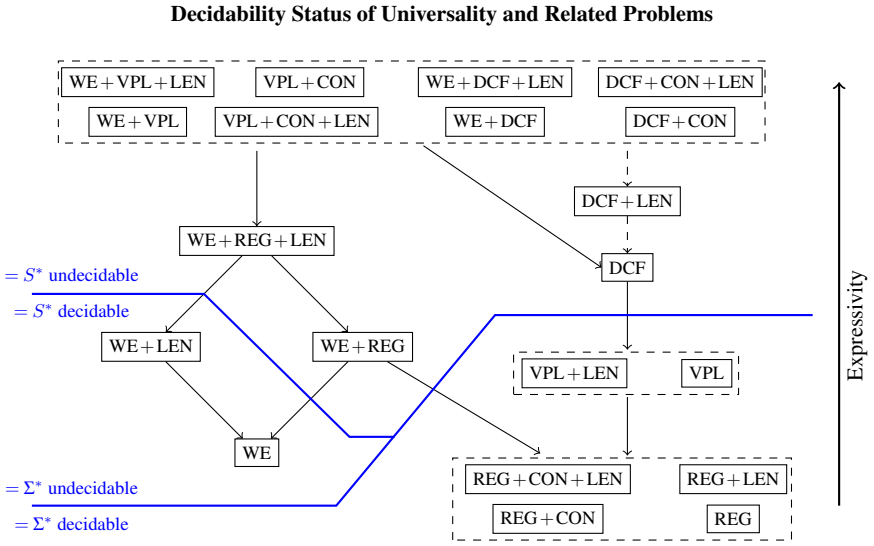

Figure 3: Depiction of (families) of theories for which the universality ($= \Sigma^*$) and subset-universality ($= S^*$) problems are decidable/undecidable. Note that undecidability of universality immediately implies undecidability of equivalence and inclusion for any theory in which the universal language Σ^* is expressible (which is true in any string-based theory containing at least one tautology). Also, an undecidable universality problem is the foundation for Greibach's theorem, which is helpful for proving that many other problems are undecidable, in particular problems concerning when a language expressed in one theory may be expressed in another are undecidable. As far as the reported results are concerned: we see a small difference between the two problems due to the fact that subsets of the alphabet are not expressible in WE and WE + LEN, rendering the subset-universality problem trivial in those cases. Known decidability results for regular and visibly pushdown languages extend to families of theories we have shown to be express exactly these classes. Equivalence and inclusion (where the two languages might come from different theories) are decidable if and only if both theories fall into cases where universality ($= \Sigma^*$) decidable.

THEORIE-AG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 39–39.

Subsequences With Gap Constraints: Complexity Bounds for Matching and Analysis Problems

Joel D. Day[(A)] Maria Kosche[(B)] Florin Manea[(B)]
Markus L. Schmid[(C)]

[(A)]Loughborough University, UK
J.Day@lboro.ac.uk

[(B)]Universität Göttingen, Germany
{maria.kosche,florin.manea}@cs.uni-goettingen.de

[(C)]Humboldt-Universität zu Berlin
MLSchmid@MLSchmid.de

Abstract

We consider subsequences with gap constraints, i. e., length-k subsequences p that can be embedded into a string w such that the induced gaps (i. e., the factors of w between the positions to which p is mapped to) satisfy given gap constraints $gc = (C_1, C_2, \ldots, C_{k-1})$; we call p a gc-subsequence of w. In the case where the gap constraints gc are defined by lower and upper length bounds $C_i = (L_i^-, L_i^+) \in \mathbb{N}^2$ and/or regular languages $C_i \in$ REG, we prove tight (conditional on the orthogonal vectors (OV) hypothesis) complexity bounds for checking whether a given p is a gc-subsequence of a string w. We also consider the whole set of all gc-subsequences of a string, and investigate the complexity of the universality, equivalence and containment problems for these sets of gc-subsequences.

THEORIE-TAG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 41–44.

Neue Anwendungen braucht das Land

Henning Fernau[(A)]

[(A)]Universität Trier, FB IV, Informatikwissenschaften
fernau@uni-trier.de

Auf den Theorietagen wurde in der Vergangenheit auch immer wieder einmal das Thema „Lehre“ aufgegriffen. Beispielhaft seien hier Bonn 2017 und Baunatal 2010 genannt.

Wir wollen in diesem Vortrag einige konkrete Ideen vorstellen, wie wir als Lehrende insbesondere im Bachelorbereich das Thema „Automaten und Formale Sprachen“ ansprechender gestalten können und so mithelfen können, dieses Thema auch in Zukunft fest im Curriculum des Informatikstudiums zu verankern. Ein gewisser Fokus der folgenden Darstellungen liegt auf dem Bereich „Künstliche Intelligenz“, da dieser immer weiteren Einfluss auch auf die Gestaltung von Informatik-Curricula nehmen wird.

Für die kurz vorstellten Ideen möchten wir nun auf einige Arbeiten verweisen.

- Eine Darstellung automatentheoretischer Konstruktionen und formalsprachlicher Methoden in spielerischer Form kann helfen, manche abstrakte Sachverhalte zu veranschaulichen, siehe [6, 12]. Der Entwurf geeigneter interessanter, ansprechender Spiele sollte als ernsthafter Forschungsgegenstand begriffen werden, um für unser Thema zu begeistern.
- Wir kommen jetzt auf den Bereich der Künstlichen Intelligenz im engeren Sinne zu sprechen.[1] Wir sollten dieses Thema aktiv in der Lehre aufgreifen und integrieren. Tatsächlich ist dies möglich, da es immer wieder Arbeiten gibt, die in typischen KI-Konferenzen oder auch -Zeitschriften (oder auch auf allgemeinen Theoriekonferenzen) erscheinen.
  - So wird in [23] beschrieben, wie eine Landkarte (Umgebung) für einen Roboter als endlicher Automat begriffen werden kann. Die möglichen Aktionen des Roboters sind dann elementare Bewegungen, die wiederum den Zustand (also den Ort) des Roboters ändern.
  - In [20] wird dargestellt, wie eine mechanische Orientierungsvorrichtung in einer Fließbandfertigung als endlicher Automat gesehen werden kann und hier auch der Begriff eines synchronisierenden Wortes als Vorschrift dieser Orientierungsvorrichtung aufgefasst werden kann. Für ähnliche Anwendungen siehe [1, 21].
  - Als Verallgemeinerung der vorigen Aufgabe kann das sogenannte *konforme Planen* [8, 9] betrachtet werden. Hierbei geht es darum, ein vorgegebenes Ziel zu erreichen, unabhängig vom Startzustand und auch unabhängig von den später beschrittenen Wegen im Falle von Wahlmöglichkeiten. Die beschriebenen Formalisierungen lehnen sich schon stark an die Sprechweise nichtdeterministischer Automaten an. Der

[(A)]Die Arbeit fasst einige Beispiele und Beobachtungen zusammen, die in Zusammenarbeit in unterschiedlichen Autorenteams entstanden sind.

[1]Wir hätten nämlich auch den Aspekt „Spiele“ aus dem vorigen Punkt hier einordnen können.

Hintergrund zu konformem Planen [14] impliziert auch eine Form der Interpretation nichtdeterministischer Maschinen, die nicht dem Standard entspricht, aber gerade deshalb interessant ist. Es ist oft motivierend zu wissen, dass diese Ideen auch im Sinne von Erweiterungen von Programmiersprachen umgesetzt wurden [14, 25]. Außerdem spielt in der algorithmischen Behandlung eine Potenzautomatenkonstruktion eine Rolle, die ähnlich zu (aber nicht identisch mit) der klassischen Lehrbuchvariante ist [4].

– Ein ganz anderer Bereich betrifft das Maschinelle Lernen, konkreter das Lernen von Automaten [11]. Auch hier lassen sich Grundideen im Grundstudium behandeln, beispielsweise das Textlernen oder auch das Anfragelernen endlicher Automaten, wie bei Angluin beschrieben [2, 3]. Hieran kann auch die Wichtigkeit des Begriffs des Minimalautomaten im Sinne einer kanonischen Darstellung regulärer Sprachen besser verstanden werden. In diesem Zusammenhang könnte man auch andere kanonische Darstellungen regulärer Sprachen diskutieren, wie beispielsweise in [5, 10] beschrieben. Für weitere Anwendungen, siehe [7, 13, 15, 16, 17, 18, 19].

Keines dieser Beispiele gehört in den Standardkanon der Anwendungsbeispiele, die typischerweise Studierenden im Bachelorbereich vermittelt werden, um ihr Interesse am Gebiet „Automaten und Formale Sprachen“ zu wecken. Hiermit möchten wir eben dazu anregen, derlei Beispiele im Unterricht zu erläutern. Das muss nicht notwendigerweise in der Vorlesung geschehen, sondern kann selbstverständlich auch im Übungsbetrieb untergebracht werden.

Natürlich sollte auch eine Einführung in Richtung Compilerbau erfolgen (da oft dieses Gebiet nicht mehr zum Kanon eines Informatikstudiums gehört), da es zum Einen wichtig ist, dass im Informatikstudium hierzu mal an irgendeiner Stelle diese Dinge erläutert werden, zum Anderen aber diese Verbindung auch eine Urmotivation war, „Automaten und Formale Sprachen“ überhaupt in den Kanon der Informatikveranstaltungen im Grundstudium aufzunehmen. Des Weiteren wollen wir nicht die Verbindungen zur (Computer-)Linguistik unerwähnt lassen, was beispielsweise auch eine Darstellung der Arbeitsweise von Baumautomaten innerhalb der Behandlung kontextfreier Sprachen nahelegt.

Sehr schön wäre sicherlich auch, Ansätze der automatentheoretischen Anwendungen im Bereich des Model Checking zu erläutern, das wird jedoch erfahrungsgemäß im Bachelorstudium eher schwierig. Es gibt aber eine weitere Verbindung zur Logik, die sich leichter integrieren lässt, nämlich das Parsen mit Hilfe logischer Kalküle. Oftmals ist ein Kurs in Logik doch einem Kurs in „Automaten und Formale Sprachen“ zeitlich vorgelagert, sodass beispielsweise die Hornlogik bekannt ist. Da sich Grammatiken im Sinne des Parsens als hornlogische Formeln verstehen lassen, ergibt sich so die Gelegenheit, ein weiteres kubisches Parsingverfahren für kontextfreie Grammatiken darzustellen, in Anlehnung an [24]. Es sollte auch nicht vergessen werden, dass Hornlogik die Basis für Prolog ist, eine Programmiersprache, die weiterhin gerne in Einführungskursen zur Künstlichen Intelligenz verwendet wird.

Sollte ein Kurs zu Berechenbarkeit und Komplexität dem zu Automaten und Formalen Sprachen vorangehen, können auch verschiedene Entscheidungsprobleme aus der Automatentheorie behandelt werden, um eben auch im Bachelorstudium oft „getrennt“ gelernte Dinge miteinander zu verbinden und so auch zu festigen. Ebenso könnten möglicherweise in Ergänzung eines typischen Einführungskurses in die Algorithmik beispielsweise Patternmatching-Algorithmen oder andere automatenbasierte Textalgorithmen besprochen werden. Aber diese beiden Dinge sind natürlich eher klassisches Terrain, weshalb sie nicht im Fokus unserer Darstellungen stehen.

Natürlich gibt es auch noch zahlreiche andere Anwendungen formalsprachlicher Methoden, beispielsweise im Bereich der Datenbanken oder auch zur Formalisierung alternativer Computermodelle. Ob und in welchem Umfang diese Anwendungen besprochen werden können, liegt wiederum am jeweiligen Curriculum. Wann ist zum Beispiel ein Kurs in Datenbanken angesetzt und was wird dort gelehrt? Je nach Curriculum bieten sich auch noch Bezüge zu Rechnerstrukturen an, beispielsweise gewissen Fragestellungen bei Schaltkreisen. Hier sei exemplarisch auf [22] verwiesen.

Literatur

[1] L. V. R. ALVES, P. N. PENA, Reconfiguration of Discrete Event Systems using Synchronizing Words. *IFAC-PapersOnLine* **53** (2020) 4, 453–458. 15th IFAC Workshop on Discrete Event Systems WODES.

[2] D. ANGLUIN, Inference of Reversible Languages. *Journal of the ACM* **29** (1982) 3, 741–765.

[3] D. ANGLUIN, Learning Regular Sets from Queries and Counterexamples. *Information and Computation* **75** (1987), 87–106.

[4] E. ARRIGHI, H. FERNAU, M. DE OLIVEIRA OLIVEIRA, P. WOLF, *Synchronization and Diversity of Solutions*, 2022.

[5] J. BJÖRKLUND, H. FERNAU, A. KASPRZIK, Polynomial Inference of Universal Automata from Membership and Equivalence Queries. *Information and Computation* **246** (2016), 3–19.

[6] W. BRAUER, M. HOLZER, B. KÖNIG, S. SCHWOON, The Theory of Finite-State Adventures. *Bulletin of the EATCS* **79** (2003), 230–237.

[7] S. CASSEL, F. HOWAR, B. JONSSON, B. STEFFEN, Extending Automata Learning to Extended Finite State Machines. In: A. BENNACEUR, R. HÄHNLE, K. MEINKE (eds.), *Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, 2016, Revised Papers*. LNCS 11026, Springer, 2018, 149–177.

[8] A. CIMATTI, M. ROVERI, Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research* **13** (2000) 1, 305–338.

[9] A. CIMATTI, M. ROVERI, P. BERTOLI, Conformant Planning via Symbolic Model Checking and Heuristic Search. *Artificial Intelligence* **159** (2004) 1, 127–206.

[10] H. FERNAU, Identification of Function Distinguishable Languages. *Theoretical Computer Science* **290** (2003), 1679–1711.

[11] H. FERNAU, C. DE LA HIGUERA, Grammar Induction: An Invitation to Formal Language Theorists. *GRAMMARS* **7** (2004), 45–55.

[12] H. FERNAU, C. HAASE, S. HOFFMANN, The Synchronization Game on Subclasses of Automata. In: P. FRAIGNIAUD, Y. UNO (eds.), *11th International Conference on Fun with Algorithms, FUN*. LIPIcs 226, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 14:1–14:17.

[13] B. GARHEWAL, F. W. VAANDRAGER, F. HOWAR, T. SCHRIJVERS, T. LENAERTS, R. SMITS, Grey-Box Learning of Register Automata. In: B. DONGOL, E. TROUBITSYNA (eds.), *Integrated Formal Methods - 16th International Conference, IFM*. LNCS 12546, Springer, 2020, 22–40.

[14] R. P. GOLDMAN, M. S. BODDY, Expressive Planning and Explicit Knowledge. In: B. DRABBLE (ed.), *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, AIPS*. AAAI, 1996, 110–117.

[15] R. GOPINATH, B. MATHIS, A. ZELLER, Mining Input Grammars from Dynamic Control Flow. In: P. DEVANBU, M. B. COHEN, T. ZIMMERMANN (eds.), *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2020, 172–183.

[16] M. HÖSCHELE, A. ZELLER, Mining Input Grammars from Dynamic Taints. In: D. LO, S. APEL, S. KHURSHID (eds.), *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE*. ACM, 2016, 720–725.

[17] M. HÖSCHELE, A. ZELLER, Mining Input Grammars with AUTOGRAM. In: S. UCHITEL, A. ORSO, M. P. ROBILLARD (eds.), *Proceedings of the 39th International Conference on Software Engineering, ICSE - Companion Volume*. IEEE Computer Society, 2017, 31–34.

[18] F. HOWAR, B. STEFFEN, Active Automata Learning in Practice - An Annotated Bibliography of the Years 2011 to 2016. In: A. BENNACEUR, R. HÄHNLE, K. MEINKE (eds.), *Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, 2016, Revised Papers*. LNCS 11026, Springer, 2018, 123–148.

[19] M. ISBERNER, F. HOWAR, B. STEFFEN, Learning Register Automata: from Languages to Program Structures. *Machine Learning* **96** (2014) 1-2, 65–98.

[20] B. K. NATARAJAN, An Algorithmic Approach to the Automated Design of Parts Orienters. In: *27th Annual Symposium on Foundations of Computer Science, FOCS*. IEEE Computer Society, 1986, 132–142.

[21] B. K. NATARAJAN, Some paradigms for the Automated Design of Parts Feeders. *The International Journal of Robotics Research* **8** (1989) 6, 98–109.

[22] I. POMERANZ, S. M. REDDY, On Synchronizing Sequences and Test Sequence Partitioning. In: *Proceedings. 16th IEEE VLSI Test Symposium*. IEEE, 1998, 158–167.

[23] R. L. RIVEST, R. E. SCHAPIRE, Inference of Finite Automata Using Homing Sequences (Extended Abstract). In: D. S. JOHNSON (ed.), *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, STOC*. ACM, 1989, 411–420.

[24] S. M. SHIEBER, Y. SCHABES, F. C. N. PEREIRA, Principles and Implementation of Deductive Parsing. *The Journal of Logic Programming* **24** (1995) 1, 3–36. Computational Linguistics and Logic Programming.

[25] D. E. SMITH, D. S. WELD, Conformant graphplan. In: J. MOSTOW, C. RICH (eds.), *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI & IAAI*. AAAI Press / The MIT Press, 1998, 889–896.

THEORIE-TAG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 45–45.

Expressiveness of Subword Constraints

Moses Ganardi[(A)]

[(A)]Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany
ganardi@mpi-sws.org

Zusammenfassung

A word u is a *subword* of a word v if u can be obtained from v by deleting letters. We study subword constraints expressed by existential first-order formulas $\varphi(\bar{x})$ over the structure $(A^*, \preccurlyeq, (w)_{w \in A^*})$ where A is an alphabet, $\preccurlyeq$ is the subword ordering, and every word $w \in A^*$ is available as a constant. While it has been known that the *truth problem* is undecidable ("Given a subword constraint φ without free variables, does φ hold?") [2], little was known on definability, i.e. which languages and relations over words are definable by subword constraints. It is easy to see that all definable relations are recursively enumerable; however, it is unclear whether all recursively enumerable relations can be defined using subword constraints, assuming $|A| \geq 2$. The undecidability proof for the truth problem relies on Diophantine equations and only shows that recursively enumerable languages over a singleton alphabet (and some auxiliary predicates) are definable. As a first step towards understanding the expressiveness of subword constraints, we prove that, if $|A| \geq 3$ then a relation $R \subseteq (A^*)^k$ is definable by subwords constraints if and only if it is recursively enumerable. Whether the same characterization holds for binary alphabets remains an open problem.

This presentation is based on joint work with Pascal Baumann, Ramanathan S. Thinniyam, and Georg Zetzsche (MPI-SWS), which has been presented at STACS 2022 [1].

Literatur

[1] P. BAUMANN, M. GANARDI, R. S. THINNIYAM, G. ZETZSCHE, Existential Definability over the Subword Ordering. In: P. BERENBRINK, B. MONMEGE (eds.), *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*. LIPIcs 219, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 7:1–7:15. https://doi.org/10.4230/LIPIcs.STACS.2022.7

[2] S. HALFON, P. SCHNOEBELEN, G. ZETZSCHE, Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 2017, 1–12.
https://doi.org/10.1109/LICS.2017.8005141

THEORIE-TAG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 47–50.

Matching Patterns with Variables Under Edit Distance

Paweł Gawrychowski[(A)] Florin Manea[(B)] Stefan Siemer[(B)]

[(A)]University of Wrocław, Faculty of Mathematics and Computer Science, Poland
gawry@cs.uni.wroc.pl

[(B)]Göttingen University, Computer Science Department and CIDAS, Germany
{florin.manea,stefan.siemer}@cs.uni-goettingen.de

Abstract

A pattern α is a string of variables and terminal letters. We say that α matches a word w, consisting only of terminal letters, if w can be obtained by replacing the variables of α by terminal words. The matching problem, i.e., deciding whether a given pattern matches a given word, was heavily investigated: it is NP-complete in general, but can be solved efficiently for classes of patterns with restricted structure. If we are interested in what is the minimum Hamming distance between w and any word u obtained by replacing the variables of α by terminal words (so matching under Hamming distance), one can devise efficient algorithms and matching conditional lower bounds for the class of regular patterns (in which no variable occurs twice), as well as for classes of patterns where we allow unbounded repetitions of variables, but restrict the structure of the pattern, i.e., the way the occurrences of different variables can be interleaved. Moreover, under Hamming distance, if a variable occurs more than once and its occurrences can be interleaved arbitrarily with those of other variables, even if each of these occurs just once, the matching problem is intractable. In this paper, we consider the problem of matching patterns with variables under edit distance. We still obtain efficient algorithms and matching conditional lower bounds for the class of regular patterns, but show that the problem becomes, in this case, intractable already for unary patterns, consisting of repeated occurrences of a single variable interleaved with terminals.

A preprint of the paper is available on arXiv, and it is available under this link:
https://arxiv.org/abs/2207.07477

1. Introduction

A *pattern with variables* is a string consisting of *constant* or *terminal letters* from a finite alphabet Σ (e.g., $\mathtt{a}, \mathtt{b}, \mathtt{c}$), and *variables* (e.g., x, y, x_1, x_2) from a potentially infinite set $\mathcal{X}$, with $\Sigma \cap \mathcal{X} = \emptyset$. In other words, a pattern α is an element of $PAT_\Sigma = (\mathcal{X} \cup \Sigma)^+$. A pattern α is mapped (by a function h called substitution) to a word by substituting the variables by arbitrary strings of terminal letters; as such, h simply maps the variables occurring in α to words over Σ. For example, $xx\mathtt{bbb}yy$ can be mapped to $\mathtt{aaaabbbbb}$ by the substitution h defined by $(x \to \mathtt{aa}, y \to \mathtt{b})$. In this framework, $h(\alpha)$ denotes the word obtained by substituting every occurrence of a variable x in α by $h(x)$ and leaving all the terminals unchanged. If a pattern α can be mapped to a string

of terminals w, we say that α matches w; the problem of deciding whether there exists a substitution which maps a given pattern α to a given word w is called the *(exact) matching problem.*

Exact Matching Problem: `Match`	
Input:	A pattern α, with $\|\alpha\| = m$, a word w, with $\|w\| = n$.
Question:	Is there a substitution h with $h(\alpha) = w$?

`Match` is NP-complete [1], in general. In fact, a detailed analysis [17, 19, 9, 10, 8, 18] of the matching problem has provided a better understanding of the parameterized complexity of this problem, highlighting, in particular, several subclasses of patterns for which the matching problem is polynomial, when various structural parameters of patterns are bounded by constants. Prominent examples in this direction are patterns with a bounded number of repeated variables, patterns with bounded scope coincidence degree [17], patterns with bounded locality [7], or patterns with a bounded treewidth [17]. See [8, 7, 17] for efficient algorithms solving $\texttt{Match}_P$ restricted to (or, in other words, parameterized by) to such classes P of patterns. In general, each of the structural parameters defining such classes P is a number k characterizing in some way the structure of the patterns of the class P and the matching algorithms for the respective class of patterns runs in $O(n^{ck})$ for some constant c. Moreover, these restricted matching problems are usually shown to be $W[1]$-hard w.r.t. the respective parameters.

In [11], the study of efficient matching algorithms for patterns with variables was extended to an approximate setting. More precisely, the problem of deciding, for a pattern α from a class of patterns P (defined by structural restrictions), a word w, and a non-negative integer Δ, whether there exists a substitution h such that the Hamming distance $d_{\texttt{HAM}}(h(\alpha), w)$ between $h(\alpha)$ and w is at most Δ was investigated. The corresponding minimization problem of computing $d_{\texttt{HAM}}(\alpha, w) = \min\{d_{\texttt{HAM}}(h(\alpha), w) \mid h \text{ is a substitution of the variables of } \alpha\}$ was also considered. The main results of [11] were rectangular time algorithms and matching conditional lower bounds for the class of regular patterns `Reg` (which contain at most one occurrence of any variable). Moreover, polynomial time algorithms were obtained for unary patterns (also known as one-variable patterns, which consist in one or more occurrences of a single variable, potentially interleaved with terminal strings) or non-cross patterns (which consist in concatenations of unary patterns, whose variables are pairwise distinct). However, as soon as the patterns may contain multiple variables, whose occurrences are interleaved, the problems became NP-hard, even if only one of the variables occurs more than once. As such, unlike the case of exact matching, the approximate matching problem under Hamming distance is NP-hard even if some of the aforementioned parameters (number of repeated variables, scope coincidence degree, treewidth, but, interestingly, not locality) were upper bounded by small constants.

Our Contribution. In this paper, inspired by, e.g., [12, 6, 3, 15, 2, 5, 4, 16] where various stringology patterns are considered in an approximate setting under *edit distance* [14, 13], and as a natural extension of the results of [11], we consider the aforementioned approximate matching problems (parameterized by a class of patterns P) for the edit distance $d_{\texttt{ED}}(\cdot, \cdot)$, instead of Hamming Distance:

Approximate Matching Decision Problem for $\texttt{MisMatch}_P$	
Input:	A pattern $\alpha \in P$, with $\|\alpha\| = m$, a word w, with $\|w\| = n$, an integer $\Delta \leq m$.
Question:	Is $d_{\texttt{ED}}(\alpha, w) \leq \Delta$?

2. Results

In the table below we can see a comparison of the complexities for the various classes of pattern, when considered without approximation, under Hamming distance as well as Edit distance, respectively, from left to right.

Table 1: Our new results are listed in column 4. The results overviewed in column 3 were all shown in [11]. We assume $|w| = n$ and $|\alpha| = m$.

Class	$\texttt{Match}(w, \alpha)$	$\texttt{MisMatch}(w, \alpha, \Delta)$ for $d_{\texttt{HAM}}(\cdot, \cdot)$	$\texttt{MisMatch}(w, \alpha, \Delta)$ for $d_{\texttt{ED}}(\cdot, \cdot)$
`Reg`	$O(n)$ [folklore]	$O(n\Delta)$, matching cond. lower bound	$O(n\Delta)$, matching cond. lower bound
`1Var` ($\texttt{var}(\alpha) = \{x\}$)	$O(n)$ [folklore]	$O(n)$	$O(n^{3\lvert\alpha\rvert_x})$ W[1]-hard w.r.t. $\lvert\alpha\rvert_x$
`NonCross`	$O(nm \log n)$ [8]	$O(n^3 p)$	NP-hard
`1RepVar` k=# x-blocks	$O(n^2)$ [8]	$O(n^{k+2}m)$ W[1]-hard w.r.t. k	NP-hard for $k \geq 1$
`kLOC`	$O(mkn^{2k+1})$ [7] W[1]-hard w.r.t. k	$O(n^{2k+2}m)$ W[1]-hard w.r.t. k	NP-hard for $k \geq 1$
`kSCD`	$O(m^2 n^{2k})$ [8] W[1]-hard w.r.t. k	NP-hard for $k \geq 2$	NP-hard for $k \geq 1$
`kRepVar`	$O(n^{2k})$ [8] W[1]-hard w.r.t. k	NP-hard for $k \geq 1$	NP-hard for $k \geq 1$
k-bounded treewidth	$O(n^{2k+4})$ [17] W[1]-hard w.r.t. k	NP-hard for $k \geq 3$	NP-hard for $k \geq 1$

References

[1] D. ANGLUIN, Finding Patterns Common to a Set of Strings. *J. Comput. Syst. Sci.* **21** (1980) 1, 46–62.
https://doi.org/10.1016/0022-0000(80)90041-0

[2] G. BERNARDINI, H. CHEN, G. LOUKIDES, N. PISANTI, S. P. PISSIS, L. STOUGIE, M. SWEERING, String Sanitization Under Edit Distance. In: *31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020)*. LIPIcs 161, 2020, 7:1–7:14.

[3] G. BERNARDINI, N. PISANTI, S. P. PISSIS, G. ROSONE, Approximate pattern matching on elastic-degenerate text. *Theor. Comput. Sci.* **812** (2020), 109–122.
https://doi.org/10.1016/j.tcs.2019.08.012

[4] P. CHARALAMPOPOULOS, T. KOCIUMAKA, S. MOZES, Dynamic String Alignment. In: *31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020)*. LIPIcs 161, 2020, 9:1–9:13.

[5] P. CHARALAMPOPOULOS, T. KOCIUMAKA, P. WELLNITZ, Faster Approximate Pattern Matching: A Unified Approach. In: S. IRANI (ed.), *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 2020, 978–989.
https://doi.org/10.1109/FOCS46700.2020.00095

[6] P. CHARALAMPOPOULOS, T. KOCIUMAKA, P. WELLNITZ, Faster Pattern Matching under Edit Distance. *CoRR* **abs/2204.03087** (2022).
https://doi.org/10.48550/arXiv.2204.03087

[7] J. D. DAY, P. FLEISCHMANN, F. MANEA, D. NOWOTKA, Local Patterns. In: *Proc. 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017*. LIPIcs 93, 2017, 24:1–24:14.

[8] H. FERNAU, F. MANEA, R. MERCAS, M. L. SCHMID, Pattern Matching with Variables: Efficient Algorithms and Complexity Results. *ACM Trans. Comput. Theory* **12** (2020) 1, 6:1–6:37. https://doi.org/10.1145/3369935

[9] H. FERNAU, M. L. SCHMID, Pattern matching with variables: A multivariate complexity analysis. *Inf. Comput.* **242** (2015), 287–305.

[10] H. FERNAU, M. L. SCHMID, Y. VILLANGER, On the Parameterised Complexity of String Morphism Problems. *Theory Comput. Syst.* **59** (2016) 1, 24–51. https://doi.org/10.1007/s00224-015-9635-3

[11] P. GAWRYCHOWSKI, F. MANEA, S. SIEMER, Matching Patterns with Variables Under Hamming Distance. In: *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021*. LIPIcs 202, 2021, 48:1–48:24. https://doi.org/10.4230/LIPIcs.MFCS.2021.48

[12] G. M. LANDAU, U. VISHKIN, Fast parallel and serial approximate string matching. *Journal of Algorithms* **10** (1989) 2, 157–169. https://www.sciencedirect.com/science/article/pii/0196677489900102

[13] V. LEVENSHTEIN, Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission* **1** (1965), 8–17.

[14] V. I. LEVENSHTEIN, Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady* **10** (1966), 707.

[15] T. MIENO, S. P. PISSIS, L. STOUGIE, M. SWEERING, String Sanitization Under Edit Distance: Improved and Generalized. In: *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021*. LIPIcs 191, 2021, 19:1–19:18.

[16] G. NAVARRO, A guided tour to approximate string matching. *ACM Comput. Surv.* **33** (2001) 1, 31–88. https://doi.org/10.1145/375360.375365

[17] D. REIDENBACH, M. L. SCHMID, Patterns with bounded treewidth. *Inf. Comput.* **239** (2014), 87–99. https://doi.org/10.1016/j.ic.2014.08.010

[18] M. L. SCHMID, A Note on the Complexity of Matching Patterns with Variables. *Inf. Process. Lett.* **113** (2013) 19, 729–733.

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

THEORIE-AG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 51–51.

The Constrained Synchronization Problem

Stefan Hoffmann[(A)]

[(A)]Informatikwissenschaften, FB IV, Universität Trier, Germany, hoffmanns.tcs@gmail.com

Zusammenfassung

I will present some unpublished work about synchronizing automata and the constrained synchronization problem touching such diverse topics as coding theory, computational complexity and descriptional complexity.

The constrained synchronization problem [1] asks for a synchronizing word of a given input automaton contained in a given (fixed) regular constraint language.

A central question concerning us is the following:

> Classify the computational complexity for different regular constraint languages.

This problem is still open. We survey the results so far, establish a connection to solid codes and show that the following computational complexities arise: AC^0, L, NL, P, NP and PSPACE.

The set of synchronizing words controls the computational complexity in the sense that an upper bound to compute it for a given class of input automata yields an upper bound, up to polynomial time reducibility, for the constrained synchronization problem.

This naturally leads to questions of descriptional complexity for the set of synchronizing words. We present results in that direction, point to connections to existing work from the area of state complexity of regular languages and present open problems and questions for further research.

Literatur

[1] H. FERNAU, V. V. GUSEV, S. HOFFMANN, M. HOLZER, M. V. VOLKOV, P. WOLF, Computational Complexity of Synchronization under Regular Constraints. In: P. ROSSMANITH, P. HEGGERNES, J. KATOEN (eds.), *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*. LIPIcs 138, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 63:1–63:14.
https://doi.org/10.4230/LIPIcs.MFCS.2019.63

THEORIE-TAG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 53–56.

Extension Acceptance and Regular ω-languages

Christopher Hugenroth[(A)]

[(A)] christopher.hugenroth@tu-ilmenau.de

Abstract

We introduce a new acceptance type for regular ω-languages called extension acceptance and study it, in particular deterministic extension automata (DEA). DEA are the first known automaton type to have the following properties:

1. Emptiness can be decided in NL.
2. Boolean operations on DEA cause at most polynomial blowup.
3. DEA capture exactly the ω-regular languages.

We compare extension acceptance to many other known acceptance types. For example, we show that extension acceptance is PTime-equivalent to Zielonka-DAG acceptance.

1. Introduction

Deterministic finite automata on finite words are fundamental in theoretical computer science. This is, at least partly, because they have many good properties. First, decision problems, like the emptiness problem, can be decided efficiently, even in non-deterministic logarithmic space. Second, Boolean operations cause at most quadratic blowup. Third, they capture exactly the class of regular languages over finite words. Fourth, they are the smallest known automaton type with these properties.

In contrast, there is no standard deterministic automaton type for regular languages over *infinite* words. The key difference between deterministic finite automata on finite and on infinite words is the way that acceptance is defined. For automata on finite words a run ends in some state, so acceptance can be defined based on states. For infinite words, however, a run does not end. Instead, the set of states visited infinitely often by a run, called a loop, is used to define acceptance. So, acceptance is defined by a set of loops. Different kinds of representations of such sets give rise to different acceptance types which in turn give rise to different automata types.

The classical acceptance types are loop, Büchi, Co-Büchi, parity, Rabin, Streett and Muller acceptance. They form the foundation for research on infinite words. For all classical types, except Muller acceptance, an optimal way of representing the set of acceptance loops is known. For Muller acceptance many representations have been proposed and each leads to a special kind of Muller acceptance. They differ significantly with respect to the blowup of Boolean operations and the complexity of associated decision problems, see table 1 row 7-13.

Acceptance Type	Non-emptiness is complete for	Blowup			Complete
		Union	Intersection	Complement	
Weak	NL	Quadratic		No blowup	No
Büchi	NL	Quadratic		No blowup (if possible)	No
Co-Büchi	NL	Quadratic		No blowup (if possible)	No
Parity	NL	Exponential		No blowup	Yes
Rabin	NL	Quad.	Exp.	Exponential	Yes
Streett	P	Exp.	Quad.	Exponential	Yes
Explicit-Muller	NL	Exponential		Exponential	Yes
Generalized Rabin	NL	Quadratic		Exponential	Yes
Generalized Streett	NP	Quadratic		Exponential	Yes
Hyper Rabin	P	Quadratic		Exponential	Yes
Hyper Streett	NP	Quadratic		Exponential	Yes
Emmerson-Lei	NP	Quadratic		No blowup	Yes
Hyper-Dual	P	Quadratic		No blowup	Yes
$\forall$-Extension	NP	Quadratic		Exponential	Yes
$\exists$-Extension	NL	Quadratic		Exponential	Yes
Extension, Zielonka DAG	NL	Quadratic		No blowup	Yes

Table 1: Overview of properties (1), (2), (3) for some deterministic automaton types. The results of the three bottom rows are established in this paper, all other results can be found in [1], [2].

Notice that for none of the aforementioned acceptance types the corresponding deterministic automaton model has all the good properties of a DFA. And to the best of the author's knowledge this hasn't been shown for any acceptance type. So, none of these deterministic automata has been known to satisfy all of the following conditions:

1. Decidability: The emptiness problem of the automaton type is in NL.
2. Closure: The automaton type is closed under Boolean operations and they cause at most polynomial blowup.
3. Completeness: The automaton type is complete, i.e. for every regular ω-language there is an automaton of the type that recognizes the language.
4. Succinctness: The automaton type is minimal in a robust sense.

Extension Acceptance

We introduce extension automata and argue that they satisfy all of the conditions (1)-(4). It is easy to define a preliminary automaton type that satisfies only conditions (1)-(3). Consider Muller automata where both the set of accepting sets $\mathcal{F}_e$ and the set of rejecting sets $\mathcal{G}_e$ are explicitly specified. This new automaton type, call it huge automaton, satisfies conditions (1)-(3) but not the succinctness condition (4) as the size of the acceptance condition is always exponential in the number of states.

We can adapt huge acceptance such that condition (4) is satisfied as well by representing $\mathcal{F}_e$ and $\mathcal{G}_e$ succinctly. For example we could use Zielonka DAGs as a succinct representation. These are Zielonka trees where some nodes are merged. This representation can be exponentially more succinct than an explicit representation and it has already been briefly considered for games in [3]. Later we will see that this Zielonka DAG acceptance does indeed satisfy all of the conditions (i)-(iv). However, Zielonka DAGs are complicated objects and we introduce a PTime-equivalent, yet simpler and more general, acceptance condition which we call extension acceptance.

In an extension condition the sets $\mathcal{F}_e$ and $\mathcal{G}_e$ are represented by two smaller sets $\mathcal{F}$ and $\mathcal{G}$. Based on $\mathcal{F}$ and $\mathcal{G}$ an extension relation is defined that relates each set of states to its minimal supersets in $\mathcal{F} \cup \mathcal{G}$. This relation can be computed efficiently and does not have to be part of the extension condition. An extension condition is then given by two sets $\mathcal{F}, \mathcal{G}$ such that each set of states has only extensions in $\mathcal{F}$ or only extension in $\mathcal{G}$ but at least one extension. For example, the Explicit-Muller condition $\mathcal{F}_e = \{\{2,3,4\},\{2,3\},\{2,4\},\{3,4\},\{2\},\{3\}\}$ is equivalent to the extension condition $(\mathcal{F},\mathcal{G})$ where $\mathcal{F} = \{\{2,3,4\}\}$ and $\mathcal{G} = \{\{1,2,3,4\},\{4\}\}$.

Even though extension automata can be much smaller than huge automata the conditions (1)-(3) still hold. In fact, for union and intersection standard product constructions suffice, yielding at most quadratic blowup. For complementation it suffices to swap $\mathcal{F}$ and $\mathcal{G}$, causing no blowup.

We argue that condition (4) holds as well. For this we define extension acceptance and compare its succinctness to the succinctness of other acceptance types. We conclude that deterministic extension automata and deterministic Zielonka DAG automata are the only known automaton types with the good properties (1)-(4).

Finally, we compare extension acceptance to other acceptance types for regular ω-languages, see for example table 1. We show that extension acceptance is an important and interesting addition to the existing acceptance types.

References

[1] Udi Boker. Why these automata types? In *LPAR*, volume 18, pages 143–163, 2018.

[2] Udi Boker. Inherent size blowup in ω-automata. In *International Conference on Developments in Language Theory*, pages 3–17. Springer, 2019.

[3] Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In *International Symposium on Mathematical Foundations of Computer Science*, pages 495–506. Springer, 2005.

[4] Orna Kupferman, Gila Morgenstern, and Aniello Murano. Typeness for ω-regular automata. In *International Symposium on Automated Technology for Verification and Analysis*, pages 324–338. Springer, 2004.

[5] C. Löding and H. Yue. Memory bounds for winning strategies in infinite games. 2008.

[6] C. Löding. Optimal bounds for transformations of omega-automata. In C. Pandu Rangan, Venkatesh Raman, and Ramaswamy Ramanujam, editors, *Foundations of Software*

Technology and Theoretical Computer Science, 19th Conference, Chennai, India, December 13-15, 1999, Proceedings, volume 1738 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 1999.

[7] Shmuel Safra. *Complexity of automata on infinite objects*. PhD thesis, 1989.

[8] Edmund M. Clarke, E. Allen Emerson, and A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[9] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and computation*, 115(1):1–37, 1994.

[10] E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Science of computer programming*, 8(3):275–306, 1987.

[11] Valerie King, Orna Kupferman, and Moshe Y. Vardi. On the complexity of parity word automata. In *International Conference on Foundations of Software Science and Computation Structures*, pages 276–286. Springer, 2001.

[12] Edmund M. Clarke, IA Browne, and Robert P. Kurshan. A unified approach for showing language containment and equivalence between various types of ω-automata. In *Colloquium on Trees in Algebra and Programming*, pages 103–116. Springer, 1990.

THEORIE-AG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 57–60.

Urgency Games

Eren Keskin Roland Meyer Sören van der Wall

TU Braunschweig
{e.keskin, roland.meyer, s.van-der-wall}@tu-bs.de

Abstract

We propose urgency games as a model for program synthesis tasks. The model is characterized by its simultaneous support of recursion and imperfect information. This is achieved by assigning urgencies to the choices made in the program. The choices are resolved in the order of urgency, as opposed to the order of sequential composition. Hence, the lower urgency choices remain hidden until all higher urgency choices are resolved. Our main result is a tight upper bound of $(2u-1)-\mathsf{EXPTIME}$ for the problem of solving urgency games, where u is the greatest urgency that appears in the game.

1. Introduction

We study reactive synthesis in a formulation inspired by syntax-guided synthesis. Given a program sketch, the task is to complete the program sketch to a full program that satisfies a given specification. The attribute reactive refers to the fact that the program is meant to interact with an environment. Completing a sketch means to augment it with a controller that selects among the implementation alternatives.

A key challenge in reactive synthesis is the aspect of imperfect information hidden behind the problem formulation. The program and the environment are different components that not only interact, but also perform local computations. As a result, they do not have full information about the local state of the other entity. The existing approaches to reactive synthesis do not handle imperfect information in a satisfactory way. The programming models are either undecidable or not expressive enough. Distributed synthesis [5] and pushdown games with imperfect information [4] are expressive models but not decidable [1, 2]. Bounded synthesis [3] and graph games with imperfect information [6] are decidable but finite state.

Our work aims to improve the situation in the setting of *recursive programs*. To this end, we propose urgency games that are played on the rewriting tree generated by an urgency grammar. An urgency grammar can be understood as a context-free grammar in which the rewriting options for a non-terminal are not represented by separate rules, but made explicit in the form of choice operators. Urgency grammars generalize the classical model in two directions. The first direction is alternation: there is not a single choice operator but an angelic choice represents implementation alternatives offered to the program template wheras a demonic choice represents the options of the environment. With the idea of games in mind, we say that angelic choices

are controlled by the existential player Eve and the demonic choices controlled by the universal player Adam. The second addition over context-free grammars are urgencies, natural numbers that are assigned to each choice. An angelic choice of urgency u will take the form $\vee_{\mathsf{u}}$, and a demonic choice with the same urgency is written as $\wedge_{\mathsf{u}}$.

An urgency grammar generates a derivation tree by resolving the choice operators, similar to how a context-free grammar would select derivation rules. The two additions, however, lead to two modifications of this process. The first is that the choice to be resolved next is determined by the (highest) urgency, unless there is a tie in which case the leftmost choice of this urgency is resolved first. The second modification will be explained in a moment.

Figure 1:: The semantics of two example terms (there are no non-terminals). The winning objective is given in gray.

To give an example, consider the terms given in Figure 1. We underline the choice that will get resolved next. In the first term, Adam makes the first choice because the urgency 2 of his choice is higher than the urgency 1 of Eve's choice. After each of his choices, Eve has an opportunity to respond. In the second term, the urgencies of the choices are flipped. This results in Eve making the first choice.

The second modification of the semantics is that urgency games do not generate plain trees but game arenas: terms in which an angelic (demonic) choice is to be resolved next are controlled by Eve (Adam). In the above example, we use circles to denote Eve's terms and boxes for Adam's terms. It remains to define a winning objective. Here we employ another mild generalization over context-free grammars. We replace the finite set of terminal symbols by a finite monoid, and implicitly assume that a concatenation $m_1.m_2$ is replaced by $m = m_1 \times m_2$. Eve plays a reachability game, her goal is to derive a monoid element from a given objective (subset of the monoid). Adam is supposed to prevent this. Monoids capture, in a uniform and simple way, both language-theoretic problems like inclusion in a regular language, and programming language situations in which variables are modified.

In the example, we consider the mod 3 addition group $M = \{0, 1, 2\}$ as our monoid and the set $O = \{0\}$ only containing the neutral element as our objective. In the first term, Eve has a strategy to reach the objective. In the second term, Adam has an option to guarantee that the result of the addition is not 0. This means that Adam has a strategy to avoid the objective. We say that a player wins from some term p, if the player has a strategy to win from p. So Eve wins the first term, while Adam wins the second term.

2. Solving Urgency Games

We solve urgency games using an effective denotational semantics approach. First, we construct a semantic domain. Then, we solve the problem at hand by finding the least solutions to a system

of equations in this domain. These equations correspond to the underlying urgency grammar. By a semantic domain, we mean a complete lattice endowed with the operators used in our model. To construct our semantic domain, we use a pre-order $\sqsubseteq$ between terms. It has the property that if $p \sqsubseteq q$, then replacing p with q in any term that Eve wins from, results in a term that she also wins from. If both directions of this relation hold, we write $p \equiv q$. The equivalence classes of $\equiv$ give us the semantic domain we seek.

To effectively operate in this domain, we define a set of normal form terms that has one representative term from each equivalence class. Normal form terms contain neither non-terminals, nor concatenation. They also have certain restrictions on how the choice operators they contain are nested. Since a normal form term is essentially a tree of choices, the winner of such a term can easily be evaluated. For this reason, finding solutions to the previously mentioned system of equations also amounts to solving the game.

Calculating the pre-order $\sqsubseteq$ is crucial to our algorithm. In order to do this in a principled way, $\sqsubseteq$ is defined as an axiomatized relation. Using the axioms, we systematically transform arbitrary terms that do not contain non-terminals to $\equiv$-equivalent normal form terms. This allows us to resolve any operator while calculating the least solutions to the equation system at hand.

In the following, we demonstrate a single step of the normalization process, the elimination of concatenation, on our previous example $(0 \vee_1 1).(0 \wedge_2 2)$. The distributive behavior of concatenation wrt. urgency is the key insight we use in this step. The relevant axioms are given below. The symbols p, q, and r refer to arbitrary terms and u to an arbitrary urgency.

$$\text{(D1)}\frac{\mathsf{urg}(p) < \mathsf{u}}{\begin{array}{r} p.(q \vee_{\mathsf{u}} r) \equiv p.q \vee_{\mathsf{u}} p.r \\ p.(q \wedge_{\mathsf{u}} r) \equiv p.q \wedge_{\mathsf{u}} p.r \end{array}} \qquad \text{(D2)}\frac{\mathsf{urg}(p) \leq \mathsf{u}}{\begin{array}{r} (q \vee_{\mathsf{u}} r).p \equiv q.p \vee_{\mathsf{u}} r.p \\ (q \wedge_{\mathsf{u}} r).p \equiv q.p \wedge_{\mathsf{u}} r.p \end{array}}$$

The rules (D1) and (D2) tell us we can distribute the side with the lower urgency, into the side with the higher urgency. In the case of a tie, we can distribute the right side into the left side. We consider monoid elements to have 0 urgency. Remark that this corresponds to the rewriting order of these terms. After exhaustively applying this rule, the concatenation only appears at the monoid level. As previously discussed, such concatenations can be seen as monoid multiplications and resolved. We apply this process to our example:

$$\begin{aligned} (0 \vee_1 1).\underline{(0 \wedge_2 2)} &\equiv (0 \vee_1 1).0 \wedge_2 (0 \vee_1 1).2 \\ \underline{(0 \vee_1 1)}.0 &\equiv 0.0 \vee_1 1.0 \\ \underline{(0 \vee_1 1)}.2 &\equiv 0.2 \vee_1 1.2 \end{aligned}$$

Putting everything together:

$$\begin{aligned} (0 \vee_1 1).0 \wedge_2 (0 \vee_1 1).2 &\equiv (0.0 \vee_1 1.0) \wedge_2 (0.2 \vee_1 1.2) \\ &\equiv (0 \vee_1 1) \wedge_2 (2 \vee_1 0) \end{aligned}$$

Remark that the resulting tree of choices is identical to the one we see in Figure 1.

The maximum size of a normal form term is the deciding factor for time complexity. Both the amount of time required to resolve an operator, and the amount of iterations required to solve the system of equations are polynomial in this value. Given an urgency, the restrictions on the nesting structure of choice operators bound the maximum size of a normal form term. Namely, normal form terms with urgency 1 have a maximum size that is exponential in the size of the monoid, and each further increase in urgency causes a doubly exponential blow up. This establishes the upper bound part of the Theorem 2.1. The lower bound is established by a reduction from a model for synthesizing concurrent programs: context-bounded two-stack multi-pushdown games. We will not go into the details of this reduction here.

Theorem 2.1 *Solving an urgency game with maximal operator urgency* u *is* $(2\mathsf{u}-1)-$ *EXPTIME-complete.*

References

[1] B. AMINOF, A. LEGAY, A. MURANO, O. SERRE, M. Y. VARDI, Pushdown module checking with imperfect information. *IC* **223** (2013), 1–17.

[2] B. FINKBEINER, S. SCHEWE, Uniform Distributed Synthesis. In: *LICS*. IEEE, 2005, 321–330.

[3] B. FINKBEINER, S. SCHEWE, Bounded synthesis. *STTT* **15** (2013) 5-6, 519–539.

[4] O. KUPFERMAN, M. Y. VARDI, P. WOLPER, Module Checking. *IC* **164** (2001) 2, 322–344.

[5] A. PNUELI, R. ROSNER, Distributed Reactive Systems Are Hard to Synthesize. In: *FOCS*. IEEE, 1990, 746–757.

[6] J. H. REIF, The complexity of two-player games of incomplete information. *JCSS* **29** (1984) 2, 274–301.

THEORIE-TAG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 61–61.

Subsequences in Bounded Ranges: Matching and Analysis Problems

Maria Kosche[(A)] Tore Koß[(A)] Florin Manea[(A)] Viktoriya Pak[(A)]

[(A)]Göttingen University, Germany
{maria.kosche,tore.koss,florin.manea,viktoriya.pak}@cs.uni-goettingen.de

Abstract

In this talk, we present a variant of the classical algorithmic problem of checking whether a given word v is a subsequence of another word w: we consider the problem of deciding, given a number p (defining a range-bound) and two words v and w, whether there exists a factor $w[i : i+p-1]$ (also called a range of length p of w) having v as subsequence (i. e., v occurs as a subsequence in the bounded range $w[i : i+p-1]$). We give matching upper and lower quadratic bounds for the time complexity of this problem. Further, we consider a series of algorithmic problems in this setting, in which, for given integers k, p and a word w, we analyse the set p-Subseq$_k(w)$ of all words of length k which occur as subsequence of some factor of length p of w. Among these, we consider the k-universality problem, the k-equivalence problem, as well as problems related to absent subsequences. Surprisingly, unlike the case of the classical model of subsequences in words where such problems have efficient solutions in general, we show that most of these problems become intractable in the new setting when subsequences in bounded ranges are considered. Finally, we provide an example of how some of our results can be applied to subsequence matching problems for circular words.

The paper this talk is based on is available on arXiv: https://arxiv.org/abs/2207.09201.

THEORIE-TAG 2022

H. Bordihn (Herausgeber): Theorietag 2022, Caputh, 19. – 21.9.2022
Universität Potsdam, Technical Report, pp. 63–65.

On the Accepting State Complexity of Operations on Permutation Automata

Christian Rauch Markus Holzer

Institut für Informatik, Universität Giessen, Arndstr. 2, 35392 Giessen, Germany
{christian.rauch,holzer}@informatik.uni-giessen.de

Recently, the accepting state complexity of a regular language was introduced in [2]. It is defined to be the minimal number of accepting states needed for a finite state device, either deterministic or nondeterministic, that accepts it. More formally, the *accepting state complexity* of a language L accepted by a finite automaton (FA) is defined as

$$\mathtt{asc}_X(L) = \min\{\, \mathtt{asc}_X(A) \mid A \text{ is a FA of type } X \text{ with } L = L(A) \,\},$$

where $\mathtt{asc}_X(A)$ refers to the number of final states of the automaton A of type $X \in \{\mathrm{D}, \mathrm{N}\}$. If there is no danger of confusion we abbreviate $\mathtt{asc}_\mathrm{D}$ by simply writing $\mathtt{asc}$ instead.

While the accepting state complexity forms a strict hierarchy of language classes for deterministic finite automata, it collapses for nondeterministic state devices, since every regular language L not containing the empty word is accepted by a nondeterministic finite automaton with a single final state, i.e., $\mathtt{asc}_\mathrm{N}(L) = 1$. If the empty word belongs to the language, the nondeterministic accepting state complexity $\mathtt{asc}_\mathrm{N}$ is at most two. Thus, the conversion from nondeterministic to equivalent deterministic finite automata can produce unbounded deterministic accepting state complexity for a regular language. Moreover, also the operational accepting state complexity $\mathtt{asc}$ was studied in the literature, which is defined as follows:

> Given are three non-negative integers m, n, and α and a regularity preserving language operation $\circ$, are there minimal DFAs A and B with accepting state complexity m and n, respectively, such that the language $L(A) \circ L(B)$ is accepted by a minimal deterministic finite automaton with α accepting states?

Following the terminology of [5] we call values α "magic" if there are no such automata A and B. The following results were shown in [2] and [3] for the operational accepting state complexity on languages accepted by DFAs—for accepting state complexities m and n one can obtain all values from the given number set for α:

- Complement: $\mathbb{N} \cup \{\, 0 \mid m = 1 \,\}$.
- Kleene star and Kleene plus: $\mathbb{N}$.
- Union: $\mathbb{N}$.
- Set difference: $\mathbb{N}$.
- Intersection: $[0, mn]$ (if the input alphabet is at least binary).
- Reversal: $\mathbb{N}$.
- Quotient: $\mathbb{N} \cup \{0\}$.

One may have noticed that for none of the above mentioned operations magic numbers exist.[1] The obtained results on the accepting state complexity prove that this measure is significantly different to the original state complexity. What is missing for the accepting state complexity is a study for certain sub-families of the regular languages in order to better understand the intrinsic behaviour of this measure.

We close this gap by studying the operational accepting state complexity for the class of permutation automata (PFAs) which accept the so called p-regular languages, also named pure-group languages. This language family is of particular interest from an algebraic point of view since their syntactic monoid induces a group. Additionally permutation automata together with permutation-reset automata play a key role in the decomposition of deterministic finite automata (DFAs), see, e.g., [7]. It is also worth to mention that the class of p-regular languages was one of the first subclasses of the regular languages for which the star height problem was shown to be decidable, see, e.g., [1]. Recently, the family of p-regular languages, and thus PFAs, gained renewed interest. For instance, in [6] the decomposing of PFAs into the intersection of smaller automata of the same kind was investigated. Moreover the operational state complexity on PFAs was studied in [4]. Up to our knowledge the operational accepting state complexity of p-regular languages was not investigated so far.

For the accepting state complexity of p-regular languages we obtain the following results:

- For the operations complement, union, set difference, Kleene star, and Kleene plus we extend the results for DFAs to the family of languages accepted by PFAs. This means even though PFAs are restricted in their expressive power compared to arbitrary DFAs, there are no magic numbers for the accepting state complexity of PFAs w.r.t. the above mentioned operations.

- When considering the reversal operation a significant difference appears. While for DFAs the reversal operation induces the whole set $\mathbb{N}$ as accepting state complexities as mentioned above, this is not the case for PFAs, where we can prove that the number $\alpha = 1$ is magic for every $m \geq 2$. In fact, we prove that for $m = 2$ no other magic number as $\alpha = 1$ exists. Whether this is also true for larger m is left open.

- Yet another difference in the accepting state complexity comes from the quotient operation. Here it turns out that for unary languages accepted by PFAs only the range $[1, mn]$ is obtainable for the accepting state complexity. This is entirely different compared to the general case.

- Finally, the unary case for the accepting state complexity of the intersection operation for DFAs in general was left open in [3]. We close this gap by considering this problem in detail. In this way, we identify a whole range of magic numbers for the intersection of unary languages accepted by PFAs and extend this result to the case of DFAs, solving the left open problem mentioned above.

The obtained results show that the accepting state complexity is an interesting measure that is worth to be studied also for other sub-regular language families that appear in the literature.

[1] The intersection of two languages L_1 and L_2 of accepting state complexity m and n is accepted by the cross product of the minimal DFAs accepting these languages. Thus, the accepting state complexity is directly bounded by mn. Therefore, the numbers greater mn are not of interest.

References

[1] J. BRZOZOWSKI, Open Problems About Regular Languages. In: R. V. BOOK (ed.), *Formal Language Theory*. Academic Press, 1980, 23–47.

[2] J. DASSOW, On the Number of Accepting States of Finite Automata. *J. Autom., Lang. Comb.* **21** (2016) 1–2, 55–67.

[3] M. HOSPODÁR, M. HOLZER, The Ranges of Accepting State Complexities of Languages Resulting From Some Operations. In: C. CAMPEANU (ed.), *Proceedings of the* 23*th Conference on Implemenation and Application of Automata*. Number 10977 in LNCS, Springer, Charlottetown, Prince Edward Island, Canada, 2018, 198–210.

[4] M. HOSPODÁR, P. MLYNÁRČIK, Operations on Permutation Automata. In: N. JONOSKA, D. SAVCHUK (eds.), *Proceedings of the* 24*th International Conference on Developments in Language Theory*. Number 12086 in LNCS, Springer, Tampa, Florida, USA, 2020, 122–136.

[5] K. IWAMA, Y. KAMBAYASHI, K. TAKAKI, Tight bounds on the number of states of DFAs that are equivalent to n-state NFAs. *Theoret. Comput. Sci.* **237** (2000) 1–2, 485–494.

[6] I. JECKER, N. MAZZOCCHI, P. WOLF, Decomposing Permutation Automata. In: S. HADDAD, D. VARACCA (eds.), *Proceedings of the* 32*nd International Conference on Concurrency Theory*. LIPIcs 203, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, Virtual Conference, 2021, 18:1–18:19.

[7] H. P. ZEIGER, Yet Another Proof of the Cascade Decomposition Theorem for Finite Automata. *Math. Syst. Theory* **1** (1967) 3, 225–228.