THEORIE-AG 2024

@

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

Foreword

The „Theorietag Automaten und Formale Sprachen“, and Deduktionstreffen are the annual meetings of the special interest groups Automata and Formal Languages, Logic in the Informatics, and, respectively, Deduction-Systems of the Gesellschaft für Informatik e.V. (German Informatics society). These meetings usually include a series of contributed talks, presenting recent works of the members of the respective groups, as well as a set of invited talks. Moreover, these events include the annual meeting of the representatives of the aforementioned special interest groups.

In 2024, all the three events will be organised together, under the generic name Theorietag 2024, by the Theoretical Computer Science group of the Georg-August-Universität Göttingen (GAU). The Theorietag will take place on September 17-20 in the Institute for Informatics of the GAU. The invited talks are given by

- Lisa Beinborn (GAU)
- Joel D. Day (University of Loughborough)
- Stefan Göller (Universität Kassel)
- Dietrich Kuske (Technische Universität Ilmenau)
- Martin Lange (Universität Kassel)
- Carsten Lutz (Universität Leipzig)
- Andreas Maletti (Universität Leipzig)

Moreover, Martin Lange (Universität Kassel) will offer a one-hour tutorial. The rest of the program consists of 20 contributed talks. This proceedings booklet contains (extended) abstracts of all 27 talks.

We are grateful to the invited speakers and all other participants for contributing a talk.

Furthermore, we thank the Gesellschaft für Informatik and the GAU for generously supporting Theorietag. Moreover, we especially thank Patricia Nitzke, Paul Sarnighausen-Cahn, Tina Ringleb, Timo Specht, and Maximilian Winkler for helping with the organization!

We wish all participants an inspiring and pleasant stay in Göttingen.

Göttingen, September 10, 2024

Tore Koß
Florin Manea
Stefan Siemer

Inhaltsverzeichnis

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 5–5.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Language modeling developments in natural language processing: Do we still need formal reasoning techniques?

Lisa Beinborn

Professor for Human-Centered Data Science, University of Göttingen
lisa.beinborn@uni-goettingen.de

Abstract

Formal reasoning techniques used to be an essential component of any natural language processing course. Now, large language models (LLMs) have taken over the field because they excel in transferring information across tasks and domains. Their distributed representation of information is particularly suited for capturing the associative power of language but it makes the models prone to irrational and inconsistent output. Such hallucinations and errors are often overlooked when focusing on averaged evaluation metrics but they are critical in human-centered applications that require precision and reliability.

In this talk, I will explore how integrating the rigor of formal reasoning techniques can complement modern LLM approaches, focusing on three research areas: cross-lingual modeling, cognitively plausible modeling, and model interpretability.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 6–6.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Towards a Better Understanding of Word Equations With Constraints

Joel D. Day[(A)]

[(A)]Department of Computer Science,
Loughborough University,
Loughborough,
UK,
LE11 3TU
J.Day@lboro.ac.uk

Abstract

Let A be a finite alphabet and X be a set of variables. A word equation is a pair of words $U, V \in (X \cup A)^*$ written $U = V$. Its solutions are substitutions for the variables which identify the two sides U and V. Much of the work on word equations has concentrated on the complexity of deciding whether or not a solution exists. Since Makanins famous decidability result, successive algorithms and improved upper bounds have been developed, although resolving the existing gap to the NP-hard lower bound remains a long-standing open problem.

Much of the early motivation for studying word equations came from connections to arithmetic and number theory via connections to Diophantine equations and Hilberts 10th problem. Since Makanins result, there has also been a close connection and applications to group theory. More recently, there has been sustained interest in solving word equations originating from the formal methods community where automated reasoning tools called string-solvers, many of which include word equations as part of their input language, have had quite some success in security-oriented software analysis and verification. Further recent work suggests emerging applications in database theory and information extraction.

Often, in the applications mentioned above, simply solving word equations in isolation is not enough: additional constraints placed on the variables play an important role in their usage. Typical constraints include regular language membership and length comparisons, but a wide variety have been studied, particularly in the context of string-solvers. Understanding these additional constraints and the expressive power they bring is arguably just as important as developing algorithms. With this talk, I will try to provide an overview of some recent results and pertinent open problems on word equations extended with constraints with emphasis on their expressive power, as well as the context and connections to modern applications.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 7–7.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

The AC^0-Complexity Of Visibly Pushdown Languages

Stefan Göller[(A)] Nathan Grosshans[(B)]

[(A)] School of Electrical Engineering and Computer Science, Universität Kassel, Germany
stefan.goeller@uni-kassel.de

[(B)] Independent Scholar, Paris Region, France
nathan.grosshans@polytechnique.edu

Abstract

We study the question of which visibly pushdown languages (VPLs) are in the complexity class AC^0 and how to effectively decide this question. Our contribution is to introduce a particular subclass of one-turn VPLs, called *intermediate VPLs*, for which the raised question is entirely unclear: to the best of our knowledge our research community is unaware of containment or non-containment in AC^0 for *any* language in our newly introduced class. Our main result states that there is an algorithm that, given a visibly pushdown automaton, correctly outputs exactly one of the following: that its language L is in AC^0, some $m \geq 2$ such that MOD_m (the words over $\{0,1\}$ having a number of 1's divisible by m) is constant-depth reducible to L (implying that L is not in AC^0), or a finite disjoint union of intermediate VPLs that L is constant-depth equivalent to. In the latter of the three cases one can moreover effectively compute $k, l \in \mathbb{N}_{>0}$ with $k \neq l$ such that the concrete intermediate VPL $L(S \to \varepsilon \mid ac^{k-1}Sb_1 \mid ac^{l-1}Sb_2)$ is constant-depth reducible to the language L. Due to their particular nature we conjecture that either all intermediate VPLs are in AC^0 or all are not. As a corollary of our main result we obtain that in case the input language is a visibly counter language our algorithm can effectively determine if it is in AC^0 — hence our main result generalizes a result by Krebs et al. stating that it is decidable if a given visibly counter language is in AC^0 (when restricted to well-matched words).

For our proofs we revisit so-called Ext-algebras (introduced by Czarnetzki et al.), which are closely related to forest algebras (introduced by Bojańczyk and Walukiewicz), and use Green's relations.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 8–9.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Partially commuting pushdowns and rational trace relations

Dietrich Kuske[(A)] Chris Köcher[(B)]

[(A)]Ilmenau

[(B)]Kaiserslautern

It is well-known that the set of reachable configurations (i.e., state followed by pushdown content) of a pushdown system is regular. The symmetric property is less well-known: the set of configurations backwards reachable is regular as well. Caucal actually proved the stronger result that the reachability relation is prefix-recognizable and therefore rational. Now the above properties hold since rational relations preserve the regularity of languages when applied on the left or the right.

In this talk, we consider a variant of pushdown sytems where the pushdown content is partially commutative, i.e., a Mazurkiewicz trace. The aim is to investigate to what extent Caucal's ideas can be transfered into this more general setting.

This endeavour meets several difficulties since

☹$_1$ the notion of "regularity" has two natural generalisations in the context of Mazurkiewicz traces, namely "rationality" and "recognizability",

but

☹$_2$ rational trace relations do not preserve the rationality nor the recognizability of trace languages

and, even more fundamental,

☹$_3$ the reachability relation of a trace pushdown system is not prefix-recognizable.

To overcome these difficulties, we demonstrate the following.

☺$_3$ The reachability relation of a trace pushdown system is the union of concatenations of prefix-recognizable trace relations.

To make use of this finding, we introduce left-closed word relations that can be understood as generalizations of homomorphisms of the trace equivalence relation. Based on these word relations, we define lc-rational trace relations and show that all prefix-recognizable trace relations are lc-rational and that the class of lc-rational trace relations is closed under union and concatenation. Consequently, we obtain that the reachability relation of a pushdown system is lc-rational.

☺$_2$ The application of an lc-rational trace relation to a rational trace language preserves the rationality. And the application of the inverse of an lc-rational trace relation to a recognizable trace language yields a recognizable trace language.

From the above, we infer the following for sets of configurations that are forwards and backwards reachable, respectively.

☺$_1$ The set of configurations reachable from some rational set is rational. And the set of configurations backwards reachable from some recognizable set is recognizable.

Somewhat surprisingly, exchanging "rational" and "recognizable" in this statement makes it false. This highlights the difference between the generalisations "recognizable" and "rational" of the class of regular word languages; it also highlights the fact that the class of lc-rational trace relations is not closed under inversion and therefore lacks a fundamental property of rational relations.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 10–11.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Fixpoint Convergence and Higher-Order (Tutorial)

Martin Lange

Theoretische Informatik / Formale Methoden, Universität Kassel
martin.lange@uni-kassel.de

Fixpoint iteration according to Kleene is the main tool for evaluating recursively defined properties in temporal specification languages like the modal mu-calculus $\mathcal{L}_\mu$ [5], using model checking for instance [1, 4]. Finite convergence, i.e. the phenomenon of reaching a fixpoint after finitely many iterations only, is of particular interest for algorithmic properties of such logics.

The iteration of fixpoints of $\mathcal{L}_\mu$ formulas trivially converges after finitely many iterations over finite structures, but also clearly over (infinite) structures whose bisimulation quotient is finite, due to $\mathcal{L}_\mu$'s bisimulation-invariance property.

We start by reviewing the converse and present a (word) structure W on which all $\mathcal{L}_\mu$ fixpoints converge finitely, even though that structure does not have a finite bisimulation quotient [3]. We then introduce Higher-Order Fixpoint Logic HFL [6], an extension of $\mathcal{L}_\mu$ in which formulas denote (higher-order) functions on sets of states of a transition system, with a type order such that $\mathcal{L}_\mu$ is exactly HFL^0, the set of HFL formulas only using functions of order 0.

HFL is interesting in this context because it is very easy to construct an HFL^1 formula, i.e. using first-order functions, that does not converge finitely over W. Moreover, it introduces a hierarchy of classes of transition systems with finite convergence of (higher-order) fixpoints [2]. It is not unreasonable to suspect this hierarchy to collapse at a low type level, but this is currently an open question. We will present some ideas on how to tackle this.

This talk will mainly have a tutorial character, with some effort on familiarising the audience with the principles of lesser known logics like HFL. This talk has also been given at the Workshop on Modal Fixpoint Logics at the University of Amsterdam on 29/01/2024.

References

[1] C. Baier, J.-P. Katoen, *Principles of model checking*. MIT Press, 2008.

[2] F. Bruse, M. Lange, É. Lozes, Collapses of Fixpoint Alternation Hierarchies in Low Type-Levels of Higher-Order Fixpoint Logic. In: *Proc. FLoC Workshop on Programming and Reasoning on Infinite Structures, PARIS'14*. 2018.

[3] F. Bruse, M. Sälzer, M. Lange, Finite Convergence of μ-Calculus Fixpoints on Genuinely Infinite Structures. In: *Proc. 46th Int. Symp. on Mathematical Foundations of Computer Science, MFCS'21*. LIPIcs 202, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 24:1–24:19.

[4] E. M. Clarke, O. Grumberg, D. Kroening, D. A. Peled, H. Veith, *Model checking, 2nd Edition*. 2nd edition, MIT Press, 2018.

[5] D. KOZEN, Results on the Propositional μ-calculus. *TCS* **27** (1983), 333–354.

[6] M. VISWANATHAN, R. VISWANATHAN, A Higher Order Modal Fixed Point Logic. In: *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04*. LNCS 3170, Springer, 2004, 512–528.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 12–12.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Extremal Fitting Problems for Conjunctive Queries

Carsten Lutz[(A)]

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

Abstract

Conjunctive queries (CQs) represent a significant and extensively researched class of database queries. In this talk, I will explore fitting problems for CQs: given a set of positively and negatively labeled data examples, is there a CQ that fits all examples? Such fitting problems underpin the query by example paradigm and are closely related to questions in machine learning. My primary focus will be fitting CQs that are extremal in being most general or most specific among all fitting CQs. Key questions are then: do such extremal fitting CQs always exist? Are they unique? And how can they be characterized and computed? Addressing these questions involves the theory of homomorphisms, linking extremal fittings to homomorphism frontiers and homomorphism dualities. The presented work was carried out jointly with Balder ten Cate, Victor Dalmau and Maurice Funk.

THEORIE-AG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 13–13.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

The Weighted HOM-Problem

Andreas Maletti[(A)] Andreea-Teodora Nász[(A)] Erik Paul[(A)]

[(A)]Universität Leipzig
Department of Mathematics and Computer Science
{maletti,nasz,epaul}@informatik.uni-leipzig.de

Abstract

We introduce the weighted HOM-problem, its motivation, and its recent solution for the nonnegative integers [5]. The weighted HOM-problem is a generalization of the (unweighted) HOM-problem [3] and asks for a given recognizable weighted tree language L and a (suitable) tree homomorphism h whether the image $h(L)$ of L under h is again recognizable. This problem is relevant in all cases in which the output of some weighted tree transducer needs to be efficiently represented. To this end we introduce weighted tree automata with constraints based on their unweighted variants [1] and investigate their basic properties [4]. Besides it turns out that at least for the semiring of nonnegative integers the weighted HOM-problem is significantly simpler to decide than the unweighted version [2].

References

[1] B. BOGAERT, S. TISON, Equality and Disequality Constraints on Direct Subterms in Tree Automata. In: *Proc. 9th Ann. Symp. Theoretical Aspects of Computer Science*. LNCS 577, Springer, 1992, 161–171.

[2] C. CREUS, A. GASCÓN, G. GODOY, L. RAMOS, The HOM problem is EXPTIME-complete. *SIAM J. Comput.* **45** (2016) 4, 1230–1260.

[3] G. GODOY, O. GIMÉNEZ, The HOM problem is decidable. *J. ACM* **60** (2013) 4, 1–44.

[4] A. MALETTI, A. NÁSZ, Weighted Tree Automata with Constraints. *Theory Comput. Syst.* **68** (2024) 1, 1–28.

[5] A. MALETTI, A. NÁSZ, E. PAUL, Weighted HOM-Problem for Nonnegative Integers. In: *Proc. 41st Int. Symp. Theoretical Aspects of Computer Science*. LIPIcs 289, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 51:1–51:17.

THEORIE-AG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 14–18.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Rollercoaster with Plateaus

Duncan Adamson[(A)] Pamela Fleischmann[(B)] Annika Huch[(B)]

[(A)]University of St Andrews
duncan.adamson@st-andrews.ac.uk
[(B)]Kiel University
{fpa,ahu}@informatik.uni-kiel.de

Abstract

In this paper we investigate the problem of detecting all maximum length plateau-k-rollercoasters appearing as a subsequence of some given word, while allowing for plateaus. We define a plateau-k-rollercoaster as a word consisting of an alternating sequence of (weakly) increasing and decreasing *runs*, with each run containing at least k *distinct* elements. We determine the longest plateau-k-rollercoasters appearing as a subsequence in any given word w of length n over an alphabet of size σ in $O(n\sigma k)$ time. Further, we present an algorithm to determine the longest common plateau-k-rollercoaster within a set of words in $O(Nk\sigma)$ where N is the product of all word lengths within the set.

1. Introduction

Subsequences of words are heavily studied, especially with focus on longest increasing or decreasing (contiguous) subsequences [1, 11, 12], and longest common subsequences [10, 13, 9]. Those two problems can be combined into the notion of *rollercoasters* which are - roughly spoken - words such that increasing and decreasing contiguous subsequences alternate and one is interested in the longest rollercoaster as a (contiguous) subsequence of a given sequence (cf. - [2, 3]). Formally, a *run* is a maximal contiguous subsequence of a given word that is strictly increasing or strictly decreasing and a *k-rollercoaster* is defined as a word such that every run is of length at least $k \geq 3$ [4]. For instance, 45763278 is a 3-rollercoaster consisting of the the runs 457, 7632, and 278. Further, the longest rollercoaster of 53671641 is 567641. Biedl et al. [2, 3] introduced and solved the rollercoaster problem parametrised in the number ℓ of different letters in the given word by presenting an $O(\ell \log(\ell))$-time algorithm. Furthermore, they showed that a word of length n contains a rollercoaster of length at least $\lceil \frac{n}{2} \rceil$ for $n > 7$ and gave an algorithm that computes such a rollercoaster in $O(n \log(n))$. Gawrychowski et al. [7] later improved the searching for k-rollercoasters to time $O(\ell k^2)$ for sequences with ℓ distinct letters by relying on algorithms for computing longest increasing subsequences (e.g., [1, 5, 11, 12]). In [6], Fujita et al. solved the *longest common rollercoaster problem* that is to find the longest common rollercoaster contained as a subsequence in two given words by providing an algorithm that runs in $O(nmk)$ time and space, where n, m are respectively the lengths of the two words.

Our Contribution. Generalising these ideas, we focus our work on rollercoasters with *plateaus*. We refer to a plateau in a rollercoaster as a factor of the rollercoaster containing only a single type of letter. For example, the rollercoaster 11223 contains the plateaus 11 and 22, while still

being a 3-rollercoaster. We restrict our definition of k-rollercoaster to require that each maximal increasing and decreasing run contains k *unique* letters, rather than having length k. Thus, 11223 is a plateau-3-rollercoaster, however despite containing a weakly increasing subsequence of length 5, it is not a 5-rollercoaster. We give an algorithm that determines the length of the longest plateau-k-rollercoaster appearing as a subsequence of a word of length n in $O(n\sigma k)$ time. Futher, we compute the longest common plateau-k-rollercoaster within a set given words in $O(Nk\sigma)$ time where N is the product of all word lengths within the set.

2. Preliminaries

Let $\mathbb{N} = \{1,2,...\}$, $[n] = \{1,\ldots,n\}$, $[m,n] = \{m,m+1,\ldots,n\}$ for $m,n \in \mathbb{N}$ with $m \leq n$. An *alphabet* Σ is a non-empty finite set whose elements are called *letters*. We assume w.l.o.g. $\Sigma = \{1,2,\ldots,\sigma\}$ for some $\sigma \in \mathbb{N}$, with the usual order on $\mathbb{N}$ to use the computational model RAM with logarithmic word-size (see, e.g., [8]) for algorithmic results. Let Σ^* denote the set of all words over Σ and Σ^n the set of all words of length $n \in \mathbb{N}$. Given $w \in \Sigma^n$, and $i \in [n]$, $w[i]$ denotes the i^{th} symbol in w. Similarly, given $i,j \in [n]$ such that $i \leq j$, $w[i,j] = w[i]w[i+1]\cdots w[j]$ and $w[j,i] = \varepsilon$ for $j > i$. A word u is a *subsequence* of w if there exist indices $i_1, i_2, \ldots, i_{|u|} \in [n]$ such that $i_1 < i_2 < \cdots < i_{|u|}$ and $u = w[i_1]w[i_2]\ldots w[i_{|u|}]$. Informally, a plateau-run is a (weakly) increasing or decreasing word. For instance, $w = 123345556$ is a plateau-6-run since firstly the letters in w are increasing. The subsequence 123456 is the longest strictly increasing run in w.

Definition 2.1 *A factor $u = w[i,j]$ of $w \in \Sigma^n$ is a* plateau-run *if, for all $\ell \in [|u|-1]$, we either have $u[\ell] \leq u[\ell+1]$ or $u[\ell] \geq u[\ell+1]$. The* orientation *of a run is $\uparrow$ (*increasing*) if $u[\ell] \leq u[\ell+1]$, or $\downarrow$ (*decreasing*) if $u[\ell] \geq u[\ell+1]$ for every $\ell \in [|u|-1]$. Such a plateau-run is called* maximal *if $w[i-1] > u[1]$ and $u[|u|] > w[j+1]$ or $w[i-1] < u[1]$ and $u[|u|] < w[j+1]$ resp. (notice that if u is a prefix or suffix of w the related constraints on the maximality are omitted). A plateau-run u is a plateau-k-run for some $k \in \mathbb{N}$ if we have $|\operatorname{alph}(u)| \geq k$.*

Given a variable $\xi \in \{\uparrow,\downarrow\}$, we use $\overline{\xi}$ to denote the opposite orientation, i.e., $\overline{\uparrow} = \downarrow$ and $\overline{\downarrow} = \uparrow$. Notice that one obtains the classical *run* introduced in [2, 3] by changing $\leq$ and $\geq$ in Definition 2.1 into $<$ and $>$.We define rollercoasters as words containing alternating (weakly) increasing and decreasing factors that are not extendable and all contain at least 3 distinct letters.

Definition 2.2 *A word $w \in \Sigma^*$ is a* plateau-k-rollercoaster *for $k \in \mathbb{N}_{\geq 3}$ if every plateau-run r in w is a maximal plateau-k-run.*

The maximality of the runs within a plateau-rollercoaster implies that their orientation is alternating, i.e., if the i^{th} run of w is increasing, then w's $(i+1)^{\text{th}}$ run is decreasing and v.v. In plateau-k-rollercoasters the runs may overlap in more than one letter: the plateau-4-rollercoaster 12223444321112345 contains the plateau-runs 12223444, 44432111, and 1112345. For decomposing rollercoaster into their runs we define $x_1 \dot{-} x_2$ by $x_1 p^{-1} x_2$ where p is the longest prefix of x_2 which is also a suffix of x_1, for $x_1, x_2 \in \Sigma^*$, and $p^{-1}x_2 = y$ iff $x_2 = py$.

We extend the notion of $(k,h)_w$-rollercoasters ([2, 3]) by not counting the length of the last run but its number of distinct letters. This notion will be mainly used within the algorithmic constructions of rollercoasters: we follow the idea of forming a rollercoaster by appending letters and tracking whether we completed a plateau-k-run (which is done by the variable h).

Definition 2.3 *For $\xi \in \{\uparrow, \downarrow\}$, $k \in \mathbb{N}$ and $h, \ell \in [k]$, w is a plateau-$(k,h)_\xi$-rollercoaster if the following properties hold for the decomposition $w = r_1 \bar{\cdot} \ldots \bar{\cdot} r_x$, $x \in \mathbb{N}$ of w into runs: First, the last plateau-run r_x has orientation ξ. Second, for all $i \in [x-1]$ the r_i is a plateau-k-run. The last plateau-run r_x is a plateau-h-run if $h \in [k-1]$ and a plateau-k-run if $h = k$.*

Note that plateau-$(k,k)_\xi$-rollercoasters for $\xi \in \{\uparrow, \downarrow\}$ is a classic plateau-k-rollercoasters. Further, a plateau-$(k,1)_\uparrow$-rollercoaster is either unary or also a plateau-$(k,k)_\downarrow$-rollercoaster since neighboured runs do overlap in in their resp. ends/beginnings within unary factors (analogously for $\downarrow$). The word $r = 12234322$ is not a plateau-4-rollercoaster since its last run does only contain 3 distinct letters but a plateau-$(4,3)_\downarrow$-rollercoaster. Further, consider $w = 43321$ and $k = 3$ which is a plateau-$(3,3)_\downarrow$- and a plateau-$(3,1)_\uparrow$-rollercoaster.

3. Longest (Common) Plateau-Rollercoaster

We begin with detecting the longest plateau-k-rollercoasters that appears as a subsequence within a given $w \in \Sigma^n$ for some $n \in \mathbb{N}$. Thus, we are extending the classical rollercoaster problem - given a word $w \in \Sigma^*$, determine the longest subsequence of w which is a rollercoaster - to the plateau-rollercoaster scenario. Thus, we introduce two sets of tables. First, $L_w^{k,h,\xi}$ (**L**ongest plateau-$(k,h)_\xi$-rollercoaster), with $L_w^{k,h,\xi}[i]$ denoting the length of the longest plateau-$(k,h)_\xi$-rollercoaster in $w[1,i]$ ending at position i, noting that this may be different from the length of the longest plateau-$(k,h)_\xi$-rollercoaster in $w[1,i]$. We abuse our notation by using $L_w^{k,k,\xi}[i]$ to denote the length of any plateau-k-rollercoaster ending with a ξ-run at $w[i]$. Secondly, we have the $n \times \sigma$ table P where $P_w[i,x]$ contains the index i' such that $i' \in [1,i]$ where $w[i'] = x$ and, $\forall j \in [i'+1, i]$, $w[j] \neq x$. Notice that the construction of P_w takes $O(n\sigma)$ time for $w \in \Sigma^n$.

For $w = 871264435161$, we have $L_w^{3,3,\uparrow}[9] = 7$ since the longest plateau-$(3,3)_\uparrow$-rollercoaster ending in $w[9]$ is 8712445. As a second example with an incomplete run in the end, we have $L_w^{3,2,\downarrow}(w)[8] = 7$ witnessed by the plateau-$(3,2)_\downarrow$-rollercoaster 8712443 ending in $w[8]$. Calculating $L_w^{k,k}$, we can see that the longest plateau-3-rollercoasters of w is of length 10 are given by 8712644311 and 8712644356. Using dynamic programming, we can compute the length of the longest common plateau-k-rollercoaster that ends in position i in a word w.

Lemma 3.1 *Given a word $w \in \Sigma^n$ and $i, k, h \in [n]$, then $L_w^{k,h,\xi}[i]$ can be determined in $O(\sigma)$ time from $L_w^{k,h',\xi'}[j]$, for all $\xi' \in \{\uparrow, \downarrow\}, h' \in [k], j \in [i-1]$.*

Theorem 3.2 *For $w \in \Sigma^n$, we can compute its longest plateau-k-rollercoaster in $O(n\sigma k)$ time.*

Now, we define for a word w the *k-rollercoaster table*, R_w for computing the longest common plateau-rollercoasters within a set of words. Informally, R_w is an extension of $L_w^{k,h,\xi}$, storing not only the longest plateau-$(k,h)_\xi$-rollercoasters ending at each position, but also the number of such rollercoasters. Using the table $L_w^{k,h,\xi}$ as a basis, we can compute the rollercoaster table R_w^k for a given input word $w \in \Sigma^n$ and $k \in \mathbb{N}$ in $O(n\sigma k)$ time.

Definition 3.3 *Given $w \in \Sigma^n$, $n, k \in \mathbb{N}$, the* rollercoaster table *of w and k is the table R_w^k of size $n \times 2 \times k$, indexed by the triples $i \in [1,n]$, $\xi \in \{\uparrow, \downarrow\}$, and $h \in [1,k]$, where $R_w^k[i, \xi, h]$ is the number of subsequences s which are plateau-$(k,h)_\xi$-rollercoasters ending at position i in w.*

We now proceed finding the longest common plateau-k-rollercoaster s in a set of words $\mathcal{W} = \{w_1, \ldots, w_n\}$, i.e., s is a plateau-k-rollercoaster in every w_i, $i \in [n]$. As with finding the longest single plateau-rollercoaster, there may be multiple solutions. We are using the same tools as before: the tables $L_{w_i}^{k,h,\xi}$, P_{w_i} and rollercoaster tables $R_{w_i}^{k}$, for every $w_i \in \mathcal{W}, \xi \in \{\uparrow, \downarrow\}$ and $h \in [k]$. From now on, let $\mathcal{W} = \{w_1, \ldots, w_n\} \subset \Sigma^*$ for $n \in \mathbb{N}$. We solve the longest common plateau-rollercoaster-problem with dynamic programming on the set of tables $\mathrm{LCR}_{\mathcal{W}}^{k,h,\xi}$.

Definition 3.4 *Let $h \in [k]$, and $\xi \in \{\uparrow, \downarrow\}$. Define the $|w_1| \times \cdots \times |w_n|$ matrix $\mathrm{LCR}_{\mathcal{W}}^{k,h,\xi}$ such that $\mathrm{LCR}_{\mathcal{W}}^{k,h,\xi}[i_1, \ldots, i_m]$ contains the length of the longest common plateau-$(k,h)_\xi$-rollercoaster of the words $w_1[1, i_1], w_2[1, i_2], \ldots, w_m[1, i_m]$ that includes $w_1[i_1], w_2[i_2], \ldots, w_m[i_m]$.*

The last condition of the definition already contains the conclusion that the longest common plateau-rollercoasters is empty if it ends in different letters. We compute the values of $LCR_{\mathcal{W}}^{k,h,\xi}$ similarly to the values of $L_w^{k,h,\xi}$. For notational brevity, we introduce the table $\mathcal{P}_{\mathcal{W}}$ analogously to P_w. Notice that $\mathcal{P}_{\mathcal{W}}$ may be computed in $O(N\sigma)$ time, where $N = \prod_{w \in \mathcal{W}} |w|$.

Definition 3.5 *The table $\mathcal{P}_{\mathcal{W}}$ of size $|w_1| \times |w_2| \times \cdots \times |w_m| \times \sigma$ is defined by $\mathcal{P}_{\mathcal{W}}[i_1, i_2, \ldots, i_m, x]$ $= (P_{w_1}[i_1, x],\ P_{w_2}[i_2, x],\ \ldots,\ P_{w_m}[i_m, x])$.*

As explained above, we only consider tuples $(i_1, \ldots, i_m)$ with $w_j[i_j] = w_{j'}[i_{j'}]$ for all $j, j' \in [m]$. The following lemma gives the computation for $\mathrm{LCR}_{\mathcal{W}}^{k,h,\xi}$ for all $h \in [k]$ and $\xi \in \{\uparrow, \downarrow\}$.

Lemma 3.6 *Given $h \in [k]$ and $(i_1, \ldots, i_m)$ appropriate, we have*

$$\mathrm{LCR}_{\mathcal{W}}^{k,k,\uparrow}[i_1, \ldots, i_m] = 1 + \max \Bigg\{ \max_{x \in [1, w_1[i_1]-1]} \{\mathrm{LCR}_{\mathcal{W}}^{k,k-1,\uparrow}[\mathcal{P}_{\mathcal{W}}[i_1 - 1, , \ldots, i_m - 1, x]\}, \max_{x \in [1, w_1[i_1]]} \{\mathrm{LCR}_{\mathcal{W}}^{k,k,\uparrow}[\mathcal{P}_{\mathcal{W}}[i_1 - 1, , \ldots, i_m - 1, x]\} \Bigg\},$$

if either $LCR_{\mathcal{W}}^{k,k-1,\uparrow}[\mathcal{P}_{\mathcal{W}}[i_1 - 1, \ldots, i_m - 1, x]) > 0$ for some $x \in [w_1[i_1] - 1]$ or
$\max_{x \in [1, w_1[i_1]]}(LCR_{\mathcal{W}}^{k,k,\uparrow}[\mathcal{P}_{\mathcal{W}}[i_1 - 1, \ldots, i_m - 1, x]) > 0$ *for some $x \in [w_1[i_1]]$,*

$$\mathrm{LCR}_{\mathcal{W}}^{k,h,\uparrow}[i_1, \ldots, i_m] = 1 + \max \Bigg\{ \max_{x \in [1, w_1[i_1]-1]} \{\mathrm{LCR}_{\mathcal{W}}^{k,h-1,\uparrow}[\mathcal{P}_{\mathcal{W}}[i_1 - 1, \ldots, i_m - 1, x]\}, \mathrm{LCR}_{\mathcal{W}}^{k,h,\uparrow}[\mathcal{P}_{\mathcal{W}}[i_1 - 1, \ldots, i_m - 1, w_1[i_1]] \Bigg\}$$

if either $LCR_{\mathcal{W}}^{k,h-1,\uparrow}[\mathcal{P}_{\mathcal{W}}[i_1 - 1, i_m - 1, x]] > 0$ for some $x \in [w_1[i_1] - 1]$, or $LCR_{\mathcal{W}}^{k,h,\uparrow}[\mathcal{P}_{\mathcal{W}}[i_1 - 1, dots, i_m - 1, w_1[i_1]] > 0$ for some $x \in [w_1[i_1]]$, and 0 otherwise, and finally $\mathrm{LCR}_{\mathcal{W}}^{k,1,\uparrow}[i_1, \ldots, i_m] = \mathrm{LCR}_{\mathcal{W}}^{k,k,\downarrow}[i_1, \ldots, i_m]$, if the respective plateau-$(k,1)_\xi$-rollercoasters are not unary. The values for $\xi = \downarrow$ can be computed analogously.

We conclude with computing the longest common plateau-k-rollercoaster of a finite set $\mathcal{W}$.

Theorem 3.7 *$\mathrm{LCR}_{\mathcal{W}}^{k,h,\xi}$, i.e., the length of the longest common plateau-k-rollercoaster, can be computed, for every $h \in [k], \xi \in \{\uparrow, \downarrow\}$, in $O(Nk\sigma)$ time, where $N = \prod_{w \in \mathcal{W}} |w|$.*

References

[1] D. ALDOUS, P. DIACONIS, Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem. *Bulletin of the American Mathematical Society* **36** (1999) 4, 413–432.

[2] T. BIEDL, A. BINIAZ, R. CUMMINGS, A. LUBIW, F. MANEA, D. NOWOTKA, J. O. SHALLIT, Rollercoasters and Caterpillars. In: *ICALP 2018*. LIPIcs 107, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 18:1–18:15.

[3] T. BIEDL, A. BINIAZ, R. CUMMINGS, A. LUBIW, F. MANEA, D. NOWOTKA, J. O. SHALLIT, Rollercoasters: Long Sequences without Short Runs. *SIAM J. Discret. Math.* **33** (2019) 2, 845–861.

[4] T. BIEDL, T. M. CHAN, M. DERKA, K. JAIN, A. LUBIW, Improved Bounds for Drawing Trees on Fixed Points with L-Shaped Edges. In: *GD 2017*. LNCS 10692, Springer, 2017, 305–317.

[5] M. L. FREDMAN, On computing the length of longest increasing subsequences. *Discret. Math.* **11** (1975) 1, 29–35.

[6] K. FUJITA, Y. NAKASHIMA, S. INENAGA, H. BANNAI, M. TAKEDA, Longest Common Rollercoasters. In: *SPIRE 2021*. LNCS 12944, Springer, 2021, 21–32.

[7] P. GAWRYCHOWSKI, F. MANEA, R. SERAFIN, Fast and Longest Rollercoasters. *Algorithmica* **84** (2022) 4, 1081–1106.

[8] J. KÄRKKÄINEN, P. SANDERS, S. BURKHARDT, Linear work suffix array construction. *J. ACM* **53** (2006) 6, 918–936.

[9] A. KOSOWSKI, An Efficient Algorithm for the Longest Tandem Scattered Subsequence Problem. In: *SPIRE 2004*. LNCS 3246, Springer, 2004, 93–100.

[10] M. KUTZ, G. S. BRODAL, K. KALIGOSI, I. KATRIEL, Faster algorithms for computing longest common increasing subsequences. *J. Discrete Algorithms* **9** (2011) 4, 314–325.

[11] D. ROMIK, *The surprising mathematics of longest increasing subsequences*. 4, Cambridge University Press, 2015.

[12] C. SCHENSTED, Longest increasing and decreasing subsequences. *Canadian Journal of mathematics* **13** (1961), 179–191.

[13] I. YANG, C. HUANG, K. CHAO, A fast algorithm for computing a longest common increasing subsequence. *Inf. Process. Lett.* **93** (2005) 5, 249–253.

THEORIE-AG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 19–25.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Tight Bounds for the Number of Absent Scattered Factors

Duncan Adamson[(A)] Pamela Fleischmann[(B)] Annika Huch[(B)]
Max Wiedenhöft[(B)]

[(A)]University of St Andrews, UK
duncan.adamson@st-andrews.ac.uk
[(B)]Kiel University, Germany
{fpa,ahu,maw}@informatik.uni-kiel.de

Abstract

A *scattered factor* of a word w is a word u obtained by deleting arbitary letters from w and keeping the order of the remaining. Barker et al. introduced the notion of k-*universality*: a word is k-universal, if it contains all possible words of length k over a given alphabet Σ as a scattered factor. Kosche et al. introduced the notion of *absent scattered factors (ASF)* to categorise the words not being scattered factors of a given word. In this paper, we investigate tight bounds on the possible number of ASFs of a given length k (also strictly longer than the shortest ASFs) among all words with the same universality extending the results of Kosche et al. Specifically, given a length k and universality index ι, we characterize ι-universal words with both the maximal and minimal number of ASFs of length k.

1. Introduction

Scattered factors serve as a complexity measure for a word's absent or existing information, with the relation between words and their scattered factors studied in combinatorics on words [28, 34], algorithms and stringology [4, 18, 3, 6, 32, 40], logics [20, 30, 29], as well as language and automata theory [1, 22, 23, 38, 39, 42] (see [31, Chapter 6, Subwords] for a profound overview on the topic). When examining discrete data, scattered factors are often used to model corrupted or lossy data [17, 10, 33] associated with the theoretical problem of reconstruction. These problems appear in a wide variety of fields, including bioinformatics in the context of protein sequences [9, 41] and formal software verification [42, 20].

In terms of theoretical work, the study of scattered factors is strongly related to the 1972 work of Simon [38] . There the famous congruence relation regarding piecewise-testable languages is introduced, nowadays known as *Simon's congruence* ([31, 36, 37, Chapter 6]). Despite heavy investigations, [39, 19, 12] and several attempts toward giving the number of congruence classes, i.e., the index of Simon's congruence, this problem remains open and continues to attract interest [27, 25, 26, 15, 16]. Current developments inspect both shortest ASFs of words [28] and relations to universality on the other hand [5, 8, 13]. This notion of universality serves as a measure of the containment of all scattered factors of up to a given length and originates from two sources: Karandikar et al. [22] introduced the notion of k-richness in the context of

piecewise testable languages [35, 24]. That coincides with the notion of k-universality. The language of k-universal words is extensively studied and characterised in [1, 5, 8, 13, 21]. One of the main insights on k-universal words is that a word is k-universal if and only if the arch factorisation of w (cf. [21]) has at least k arches.

Own Contribution. An open question that we address is to generalise the ideas of [28, 14] to ASFs that are longer than the shortest ones. A first generalisation is asking for the word containing exactly ι arches with the greatest (resp. fewest) number of absent factors of length k given ι and k with $\iota < k$. In this way, we build on the work of [1] in looking at the properties of k-universality within languages. While [1] considered the problem if a word is k-universal in a given language, we look for the word that is closest (resp. furthest) to k-universality within the restricted language of ι-universal words. We give a tight characterisation for both. Additionally, we characterise the words fulfilling the property and have the shortest possible length. After that, we show the structure of words $w_{min} \in \Sigma^*$ with a minimal number of existing scattered factors among all words with the same number of arches. Afterwards, we show that we can enumerate all scattered factors of any length $k \in [|w_{min}|]$ with constant delay $O(1)$.

Structure of the Work. We begin, in Section 2, with basic definitions used in this paper. After that, in Section 3, we show the results regarding words with a minimal and maximal number of ASFs. Afterwards, we briefly discuss their scattered factor counting and enumeration properties.

2. Prelims

Let $\mathbb{N}$ be the set of all natural numbers, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, $[n] = \{1, \dots, n\}$, $[i, j] = [j] \setminus [i-1]$, and $[n]_0 := [n] \cup \{0\}$. An *alphabet* Σ is a non empty, finite set whose elements are called *letters* or *symbols*. For a given $\sigma \in \mathbb{N}$, we fix $\Sigma = \{\mathtt{a}_1, \dots, \mathtt{a}_\sigma\}$. A *word* is a finite sequence of letters from Σ. Let Σ^* be the set of all finite words over Σ with concatenation and the empty word ε as neutral element. Set $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$. Let $w \in \Sigma^*$. For all $n \in \mathbb{N}_0$ define inductively, $w^0 = \varepsilon$ and $w^n = ww^{n-1}$. The *length* of w is the number of w's letters; thus $|\varepsilon| = 0$. For all $k \in \mathbb{N}_0$ set $\Sigma^k := \{w \in \Sigma^* \mid |w| = k\}$ (define $\Sigma^{\leq k}$, $\Sigma^{\geq k}$ analogously). We denote w's i^{th} letter by $w[i]$ and set $w[i..j] = w[i] \cdots w[j]$ if $i < j$, and ε if $i > j$ for all $i, j \in [|w|]$. Set $\text{alph}(w) = \{\mathtt{a} \in \Sigma \mid \exists i \in [|w|] : w[i] = \mathtt{a}\}$ as w's alphabet and for each $\mathtt{a} \in \Sigma$ set $|w|_\mathtt{a} = |\{i \in [|w|] \mid w[i] = \mathtt{a}\}|$. The word $u \in \Sigma^*$ is called a *factor* of w if there exist $x, y \in \Sigma^*$ such that $w = xuy$. In the case $x = \varepsilon$, we call u a *prefix* of w and *suffix* if $y = \varepsilon$. Define the *reverse* of w by $w^R = w[|w|] \cdots w[1]$. Let $<_\Sigma$ be a total order on Σ and denote the *lexicographical order* on Σ^* by $<$. Define w_Σ as the word in Σ^σ with $w_\Sigma[i] <_\Sigma w_\Sigma[i+1]$ and $\text{alph}(w) = \Sigma$. A word $u \in \Sigma^*$ is called a *scattered factor* of w - denoted as $u \in \text{ScatFact}(w)$ - if there exist $v_1, \dots, v_{|u|+1} \in \Sigma^*$ such that $w = v_1 u[1] v_2 u[2] \cdots v_{|}u|u[|u|] v_{|u|+1}$. Set $\text{ScatFact}_k(w) = \{u \in \text{ScatFact}(w) \mid |u| = k\}$ for some $k \in \mathbb{N}_0$. Two words $w, v \in \Sigma^*$ are *Simon congruent* w.r.t. $k \in \mathbb{N}_0$ ($w \sim_k v$) if $\text{ScatFact}_\ell(w) = \text{ScatFact}_\ell(v)$ for all $\ell \leq k$. The word $w \in \Sigma^*$ is called m-*nearly* k-*universal* if $|\text{ScatFact}_k(w)| = \sigma^k - m$. Let $\text{NUniv}_{\Sigma,m,k}$ be the set of all m-nearly k-universal words over Σ. If $m = 0$, the words are called k-universal, and if $m = 1$, they are called nearly k-universal. The respective sets are $\text{Univ}_{\Sigma,k}$ and $\text{NUniv}_{\Sigma,k}$. The largest $k \in \mathbb{N}_0$, such that a word is k-universal, is called the *universality index* $\iota(w)$. For $w \in \Sigma^*$ the *arch factorisation* by Hébrard [21] is given by $w = \text{ar}_1(w) \cdots \text{ar}_k(w)\text{r}(w)$ for $k \in \mathbb{N}_0$ with $\iota(\text{ar}_i(w)) = 1$ for all $i \in [k]$, $\text{ar}_i(w)[|\text{ar}_i(w)|] \notin \text{alph}(\text{ar}_i(w)[1..|\text{ar}_i(w)|-1])$ for all $i \in [k]$, and

$\text{alph}(\text{r}(w)) \subsetneq \Sigma$. The words $\text{ar}_i(w)$ are called *arches* of w and $\text{r}(w)$ is the *rest* of w. The *modus* $\text{m}(w)$ is given by $\text{ar}_1(w)[|\,\text{ar}_1(w)|]\cdots\text{ar}_k(w)[|\,\text{ar}_k(w)|]$, i.e., it is the word containing the unique last letters of each arch. The *inner of the i^{th} arch of w* is defined as the prefix of $\text{ar}_i(w)$ such that $\text{ar}_i(w) = \text{in}_i(w)\,\text{m}(w)[i]$ holds. We call $w \in \Sigma^*$ *perfect k-universal* if $\iota(w) = k$ and $\text{r}(w) = \varepsilon$. The set of all these words with $\text{alph}(w) = \Sigma$ is denoted by $\text{PUniv}_{\Sigma,k}$. Additionally, we call w *minimal perfect k-universal* if $w \in \text{PUniv}_{\Sigma,k}$ and w is of minimal length among all words in $\text{PUniv}_{\Sigma,k}$. The set of those words over Σ is denoted by $\text{MinPUniv}_{\Sigma,k}$. We only consider at least binary alphabets. Moreover, we assume $\Sigma = \text{alph}(w)$ for a given w, if not stated otherwise.

3. Lower and Upper Bound for m

In this section, we consider the following problem: Given a specific universality $\iota \in \mathbb{N}$ and a number k, what is the minimum number of absent scattered factors of length k for any word w where $\iota(w) = \iota$? Further, we are interested in finding the shortest word w fulfilling this property. We start by determining which words are always absent from $\text{ScatFact}_k(w)$. We start with a proposition regarding the structure of the words yielding a lower bound.

Proposition 3.1 *Given $k \in \mathbb{N}$ and $w \in \Sigma^*$ with $k > \iota(w)$, the following equivalence holds*
1) for all $w' \in \Sigma^*$ with $\iota(w') = \iota(w)$ we have $|\,\text{ScatFact}_k(w')| \leq |\,\text{ScatFact}_k(w)|$,
2) there exists exactly one $\mathtt{a} \in \Sigma$ with $\text{ScatFact}_k(w) = \{v \in \Sigma^k |\ \text{m}(w)\mathtt{a} \notin \text{ScatFact}_{\iota(w)+1}(v)\}$.

Proposition 3.2 *Let $k \in \mathbb{N}_0$ and $w \in \Sigma^*$ with $k > \iota(w)$ and for all $w' \in \Sigma^*$ with $\iota(w) = \iota(w')$ we have $|\,\text{ScatFact}_k(w)| \geq |\,\text{ScatFact}_k(w')|$. Then, $|\,\text{ScatFact}_k(w)| = \sum_{j\in[0,\iota(w)]}\binom{k}{j}(\sigma-1)^{k-j}$.*

This gives us immediately $\sigma^k - \sum_{j\in[0,\iota']}\binom{k}{j}(\sigma-1)^{k-j}$ as the number of absent scattered factors which are absent in any case and thus a lower bound of m. We now move to the problem of determining the shortest word of a given universality ι having the minimum number of absent scattered factors of length k. With the help of some structural properties we obtain the following proposition and an implied corollary for $\iota, k \in \mathbb{N}$ with $k > \iota$, $w \in \Sigma^*$ with $\iota(w) = \iota$ such that for all $w' \in \Sigma^{\geq|w|}$ we have $|\,\text{ScatFact}_k(w)| \geq |\,\text{ScatFact}_k(w')|$ and for all $w' \in \Sigma^{<|w|}$ we have $|\,\text{ScatFact}_k(w)| > |\,\text{ScatFact}_k(w')|$, assuming $\iota(w) = \iota(w')$. This leads to the desired characterisation and also to the number of shortest words having a minimal number of absent scattered factors.

Proposition 3.3 *We have $|w| = (\iota+1)\cdot(\sigma-1)\cdot(k-\iota)+\iota$.*

Corollary 3.4 *We have $\text{in}_i(w) \in \text{MinPUniv}_{(\Sigma\setminus\{\text{m}(w)[i]\}),k-\iota}$ and $\text{r}(w) \in \text{MinPUniv}_{(\Sigma\setminus\{\mathtt{a}\}),k-\iota'}$ for all $i \in [\iota]$.*

Theorem 3.5 *Given $\iota, k \in \mathbb{N}$ and $w \in \Sigma^*$ with $k > \iota = \iota(w)$, the following equivalence holds.*
1) For all $w' \in \Sigma^{\geq|w|}$ we have $|\,\text{ScatFact}_k(w)| \geq |\,\text{ScatFact}_k(w')|$ and for all $w' \in \Sigma^{<|w|}$ we have $|\,\text{ScatFact}_k(w)| > |\,\text{ScatFact}_k(w')|$
2) there exists some $\mathtt{a} \in \Sigma$ such that $\text{m}(w) = \mathtt{a}^\iota$ and for all $i \in [\iota]$ we have $\text{in}_i(w), \text{r}(w) \in \text{MinPUniv}_{\Sigma',k-\iota}$ for $\Sigma' = \Sigma \setminus \{\mathtt{a}\}$.

Proposition 3.6 *There exist $\sigma((\sigma-1)!)^{(\iota+1)(k-\iota)}$ different shortest words w with $\iota(w) = \iota$.*

Now, we determine the maximum possible number of absent scattered factors of words with a given universality and a given scattered factor length. We do so by constructing a word with the minimum number of existing scattered factors amongst all words of the given universality. Hence, we need an approach which considers all scattered factors of a given $k \in [\iota(w)+1, |w|]$. First, we define a helper function $\text{next}_w(i, s)$ taking, as input, some alphabet size $s \in [\sigma]$ and index $i \in [|w|]$. The function returns the leftmost following position j after i in w such that $w[i+1..j]$ contains s distinct letters or -1 if no such position exists. In particular, we show that all words $w \in \Sigma^*$ of the form $\text{ar}_1(w) = \mathtt{a}_1 \cdots \mathtt{a}_\sigma$ and $\text{ar}_i(w) = \text{ar}_{i-1}(w)^R$ have the minimum number of scattered factors amongst all words with universality $\iota(w)$ over that alphabet $\Sigma = \{\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_\sigma\}$. We assume, for the remainder of this section, that w_{min} is a word of such structure, with $\text{ar}_1(w_{\text{min}}) = \mathtt{a}_1 \cdots \mathtt{a}_\sigma$, noting that the same construction can be applied for any permutation of Σ. Kosche et al. in [28] have shown that w_{min} has the fewest scattered factors of length $\iota(w_{\text{min}})+1$ amongst all words over $\text{alph}(w_{\text{min}})$ with universality $\iota(w_{\text{min}})$. We extend this to the general case, showing that w_{min} has the fewest scattered factors of length $k \in [|w|]$, amongst the set of words $\{v \in \Sigma^* | \iota(v) = \iota(w_{\text{min}})\}$, noting that for any $k \leq \iota(w_{\text{min}})$, all words in this set contain every word in Σ^k as a scattered factor, and thus this condition holds. This leads to the main theorem.

Theorem 3.7 *W.l.o.g. assume $\Sigma := \{\mathtt{a}_1, ..., \mathtt{a}_\sigma\}$ for some $\sigma \in \mathbb{N}$. Let $w \in \Sigma^*$ such that w.l.o.g. $\text{ar}_1(w) = \mathtt{a}_1 \cdots \mathtt{a}_\sigma$ and $\text{ar}_i(w) = \text{ar}_{i-1}(w)^R$ for all $i \in [\iota(w)] \setminus \{1\}$. Then for all $w' \in \Sigma^*$ with $\iota(w) = \iota(w')$ we have $|\text{ScatFact}_k(w)| \leq |\text{ScatFact}_k(w')|$.*

Finally, we briefly mention the problems of counting and enumerating the set of scattered factors within a given word. First, we can count the number of distinct scattered factors in a word w_{min} as structurally defined by Theorem 3.7. To obtain the specific number of scattered factors of w_{min} for all lengths $k < |w|$, one can use linear time algorithms[1], as shown by Elzinga et al. [11]. Whether a closed form for the specific case of w_{min} exists, remains an open question. By this, we can enumerate all scattered factors of w_{min} with constant delay. To do that, we first need to preprocess w_{min} into its scattered factors recognising automaton (see [7]). Then, we apply the algorithm given in [2] to obtain such an enumeration.

Proposition 3.8 *Let $w \in \Sigma^*$. For any $k \in [|w|]$, we can enumerate $\text{ScatFact}_k(w)$ with $O(1)$ delay and $O(\sigma \cdot |w|)$-time preprocessing.*

4. Conclusion

Within this work we investigated the minimal and maximal number of absent scattered factors for a given alphabet Σ, scattered factor length k and universality index ι. During the investigation we gave two tight bounds for m as well as constructions on ι-universal words over Σ that omit this number of absent scattered factors of length k.

[1] Consider `https://www.geeksforgeeks.org/count-distinct-subsequences/` for implementations (Accessed: 25. June 2024).

References

[1] D. ADAMSON, P. FLEISCHMANN, A. HUCH, T. KOSS, F. MANEA, D. NOWOTKA, k-Universality of Regular Languages. In: *ISAAC 2023*. LIPIcs 283, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 4:1–4:21.

[2] D. ADAMSON, P. GAWRYCHOWSKI, F. MANEA, Enumerating m-Length Walks inăDirected Graphs withăConstant Delay. In: J. A. SOTO, A. WIESE (eds.), *LATIN 2024: Theoretical Informatics*. Springer Nature Switzerland, Cham, 2024, 35–50.

[3] R. A. BAEZA-YATES, Searching Subsequences. *TCS* **78** (1991) 2, 363–376.

[4] H. BANNAI, T. I, T. KOCIUMAKA, D. KÖPPL, S. J. PUGLISI, Computing Longest Lyndon Subsequences and Longest Common Lyndon Subsequences. *Algorithmica* **86** (2024) 3, 735–756.

[5] L. BARKER, P. FLEISCHMANN, K. HARWARDT, F. MANEA, D. NOWOTKA, Scattered Factor-Universality of Words. In: *DLT 2020*. LNCS 12086, Springer, 2020, 14–28.

[6] K. BRINGMANN, M. KÜNNEMANN, Multivariate Fine-Grained Complexity of Longest Common Subsequence. In: *SODA 2018*. SIAM, 2018, 1216–1235.

[7] M. CROCHEMORE, B. MELICHAR, Z. TRONÍEK, Directed acyclic subsequence graphOverview. *Journal of Discrete Algorithms* **1** (2003) 3, 255–280.

[8] J. D. DAY, P. FLEISCHMANN, M. KOSCHE, T. KOSS, F. MANEA, S.SIEMER, The Edit Distance to k-Subsequence Universality. In: *STACS 2021*. LIPIcs 187, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 25:1–25:19.

[9] D. T. DO, T. T. Q. LE, N. LE, Using deep neural networks and biological subwords to detect protein S-sulfenylation sites. *Briefings Bioinform.* **22** (2021) 3.

[10] A. W. M. DRESS, P. L. ERDŐS, Reconstructing words from subwords in linear time. *Annals of Combinatorics* **8** (2005), 457–462.

[11] C. ELZINGA, S. RAHMANN, H. WANG, Algorithms for subsequence combinatorics. *Theoretical Computer Science* **409** (2008) 3, 394–404.

[12] L. FLEISCHER, M. KUFLEITNER, Testing Simon's congruence. In: *MFCS 2018*. LIPIcs 117, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 62:1–62:13.

[13] P. FLEISCHMANN, S. GERMANN, D. NOWOTKA, Scattered Factor Universality–The Power of the Remainder. *preprint arXiv:2104.09063 (published at RuFiDim)* (2021).

[14] P. FLEISCHMANN, L. HASCHKE, J. HÖFER, A. HUCH, A. MAYROCK, D. NOWOTKA, Nearly k-universal words - Investigating a part of Simon's congruence. *TCS* **974** (2023), 114113.

[15] P. FLEISCHMANN, J. HÖFER, A. HUCH, D. NOWOTKA, α-β-Factorization and the Binary Case of Simon's Congruence. In: *FCT 2023*. LNCS 14292, Springer, 2023, 190–204.

[16] P. FLEISCHMANN, S. KIM, T. KOSS, F. MANEA, D. NOWOTKA, S. SIEMER, M. WIEDENHÖFT, Matching Patterns with Variables Under Simon's Congruence. In: *RP 2023*. LNCS 14235, Springer, 2023, 155–170.

[17] P. FLEISCHMANN, M. LEJEUNE, F. MANEA, D. NOWOTKA, M. RIGO, Reconstructing Words from Right-Bounded-Block Words. *Int. J. Found. Comput. Sci.* **32** (2021) 6, 619–640.

[18] P. J. FORRESTER, A. MAYS, Finite size corrections relating to distributions of the length of longest increasing subsequences. *Adv. Appl. Math.* **145** (2023), 102482.

[19] P. GAWRYCHOWSKI, M. KOSCHE, T. KOSS, F. MANEA, S. SIEMER, Efficiently Testing Simon's Congruence. In: *STACS 2021*. LIPIcs 187, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 34:1–34:18.

[20] S. HALFON, P. SCHNOEBELEN, G. ZETZSCHE, Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In: *LICS 2017*. IEEE Computer Society, 2017, 1–12.

[21] J. HÉBRARD, An Algorithm for Distinguishing Efficiently Bit-Strings by their Subsequences. *TCS* **82** (1991) 1, 35–49.

[22] P. KARANDIKAR, M. KUFLEITNER, P. SCHNOEBELEN, On the index of Simon's congruence for piecewise testability. *Inf. Process. Lett.* **115** (2015) 4, 515–519.

[23] P. KARANDIKAR, P. SCHNOEBELEN, The height of piecewise-testable languages and the complexity of the logic of subwords. *Log. Methods Comput. Sci.* **15** (2019) 2.

[24] K. KÁTAI-URBÁN, P. P. PACH, G. PLUHÁR, A. PONGRÁCZ, C. SZABÓ, On the word problem for syntactic monoids of piecewise testable languages. In: *Semigroup Forum*. 84, Springer, 2012, 323–332.

[25] S. KIM, Y. HAN, S. KO, K. SALOMAA, On Simon's congruence closure of a string. *TCS* **972** (2023), 114078.

[26] S. KIM, Y. HAN, S. KO, K. SALOMAA, On the Simon's Congruence Neighborhood of Languages. In: *DLT*. LNCS 13911, Springer, 2023, 168–181.

[27] S. KIM, S. KO, Y. HAN, Simon's congruence pattern matching. *TCS* **994** (2024), 114478.

[28] M. KOSCHE, T. KOSS, F. MANEA, S. SIEMER, Absent Subsequences in Words. *Fundam. Informaticae* **189** (2022) 3-4, 199–240.

[29] D. KUSKE, The Subtrace Order and Counting First-Order Logic. In: *Computer Science – Theory and Applications*. Springer International Publishing, 2020, 289–302.

[30] D. KUSKE, G. ZETZSCHE, Languages Ordered by the Subword Order. In: *Foundations of Software Science and Computation Structures*. Springer International Publishing, 2019, 348–364.

[31] M. LOTHAIRE, *Combinatorics on Words*. Cambridge Mathematical Library, Cambridge University Press, 1997.

[32] D. MAIER, The Complexity of Some Problems on Subsequences and Supersequences. *J. ACM* **25** (1978) 2, 322–336.

[33] J. MANUCH, Characterization of a word by its subwords. In: *DLT 1999*. World Scientific, 1999, 210–219.

[34] A. MATEESCU, A. SALOMAA, S. YU, Subword histories and Parikh matrices. *J. Comput. Syst. Sci.* **68** (2004) 1, 1–21.

[35] P. P. PACH, Normal forms under Simon's congruence. In: *Semigroup Forum*. 97, Springer, 2018, 251–267.

[36] J.-E. PIN, Finite semigroups and recognizable languages: an introduction. *NATO Advanced Study Institute Semigroups, Formal Languages and Groups, J. Fountain (ed.), Kluwer academic publishers* (1995), 1–32.

[37] J.-E. PIN, The influence of Imre Simon's work in the theory of automata, languages and semigroups. In: *Semigroup Forum*. 98, Springer, 2019, 1–8.

[38] I. SIMON, *Hierarchies of events with dot-depth one*. Ph.D. thesis, University of Waterloo, 1972.

[39] I. SIMON, Piecewise testable events. In: *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975*. LNCS 33, Springer, 1975, 214–222.

[40] R. A. WAGNER, M. J. FISCHER, The String-to-String Correction Problem. *J. ACM* **21** (1974) 1, 168–173.

[41] C. WANG, K. CHO, J. GU, Neural Machine Translation with Byte-Level Subwords. In: *EAAI 2020*. AAAI Press, 2020, 9154–9160.

[42] G. ZETZSCHE, The Complexity of Downward Closure Comparisons. In: *ICALP 2016*. LIPIcs 55, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016, 123:1–123:14.

THEORIE-AG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 26–26.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Enumerating m-Length Walks in Directed Graphs with Constant Delay

Duncan Adamson[(A)] Florin Manea[(B)] Paweł Gawrychowski[(C)]

[(A)]St. Andrews University
Duncan.Adamson@st-andrews.ac.uk
[(B)]University of Göttingen
florin.manea@cs.uni-goettingen.de
[(C)]Wrocław University
gawry@cs.uni.wroc.pl

Abstract

In this work, we provide a novel enumeration algorithm for the set of all walks of a given length within a directed graph. Our algorithm has worst-case constant delay between outputting succinct representations of such walks, after a preprocessing step requiring linear time relative to the size of the graph. We apply these results to the problem of enumerating succinct representations of the strings of a given length from a prefix-closed regular language (languages accepted by a finite automaton which has final states only).

This presentation is based on the paper [1], presented at LATIN 2024.

References

[1] D. ADAMSON, P. GAWRYCHOWSKI, F. MANEA, Enumerating m-Length Walks in Directed Graphs with Constant Delay. In: J. A. SOTO, A. WIESE (eds.), *LATIN 2024: Theoretical Informatics - 16th Latin American Symposium, Puerto Varas, Chile, March 18-22, 2024, Proceedings, Part I*. Lecture Notes in Computer Science 14578, Springer, 2024, 35–50.
https://doi.org/10.1007/978-3-031-55598-5_3

THEORIE-AG 2024 T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024, Göttingen, 17. – 20.09.2024, Technical Report, pp. 27–30.

A Chomsky Grammar Simulator Written in the J Language

Hauke Rehr[(A)] Fleming Kretschmer[(B)]
Christian Höner zu Siederdissen[(C)] Clemens Grelck[(D)]
Emanuel Barth[(E)] Martin Mundhenk[(F)] Thomas Hinze[(G)]

[(A)] hauke.rehr@uni-jena.de
[(B)] fleming.kretschmer@uni-jena.de
[(C)] christian.hoener.zu.siederdissen@uni-jena.de
[(D)] clemens.grelck@uni-jena.de
[(E)] emanuel.barth@uni-jena.de
[(F)] martin.mundhenk@uni-jena.de
[(G)] thomas.hinze@uni-jena.de

Abstract

Emulation of the language generation process from a given Chomsky grammar turns out to be a time-consuming task when implemented imperatively using a Monte Carlo approach. Here a mainly functional approach written in *J* is introduced the recursion scheme of which has been optimized to operate more efficiently. Following a discussion of the entire source code, case studies addressing the Partition problem demonstrate its performance.

Notation

- For any $n \in \mathbb{N}_0$, let $I_n \coloneqq [1,n] \cap \mathbb{N}$.
- For any universe U and subset S drawn from U, let $\overline{S} \coloneqq U \setminus S$ denote the relative complement of S with respect to U.
- If a set is quantified over and no representative name is given, it will implicitly be the miniscule variant of the set name, e. g. $\underset{S}{\forall} s \in U$.
- If the set quantified over is some power of a set, positive natural indices are provided to constituent tuple elements in turn, e. g. $\underset{S^2}{\forall} s \in\!\!\neq\!\!\Rightarrow |\{s_1, s_2\}| = 2$.
- The above quantification rules apply to indices bound by index sets as well.

1. A Chomsky Grammar simulator

1.1. Mathematical Model

Given sets $\mathcal{T}$ and N with $\mathcal{T} \cap N = \emptyset$ and some $S \in N$, the shorthand $\mathcal{S} \coloneqq (\mathcal{T} \cup N)^*$, and any rule set $R \subset \mathcal{S}^{\mathcal{S} \setminus \mathcal{T}^*}$, $G = (N, \mathcal{T}, R, S)$ is called a grammar.
ε denotes the empty word. $\mathcal{A}^* \coloneqq \bigcup_{\mathbb{N}_0} \bigtimes_{I_n} \mathcal{A}$ is the Kleene hull of a language or alphabet $\mathcal{A}$.

Throughout this section, the substitution process unfolding the induced language $L(G)$ expressed by some grammar G will be shown by code samples written in the *J* language. For an introduction, see [3]. The grammar data itself will be provided to the program by means of *.json* files.

```
NB. === read the grammar file      ===
load'convert/json'
jsn =: dec_json freads grammar
'rules start_symbol term_symbols var_symbols' =: ({:/:{.) jsn
```

1.2. Iterative Substitution

The function R is actually meant to work on infix sets. Let Δ be its current domain. As a first step, an auxiliary definition $\underset{\mathcal{S}}{\forall}\rho(s) \coloneqq \bigcup_{\mathcal{S}^2}\bigcup_{\Delta}\{s_1 \circ R(\delta) \circ s_2 \mid s = s_1 \circ \delta \circ s_2\}$ where $\circ$ denotes concatenation treats infixes (any of s_1 and s_2 may be the empty word). As a second step, the domain of R gets broadened, including sets $W \in 2^{\mathcal{S}}$ of words: $R(W) \coloneqq \bigcup_{W}\rho(w)$.
The language generated by G can now be defined by $L(G) \coloneqq \bigcup_{\mathbb{N}_0} R^n(\{(S)\}) \cap \mathcal{T}^*$.

```
NB. === get all substitutions      ===
where =: >@] (I.@E.~ +"0 1 0:, #@]) 0 {:: [
presuf =: ((<@{.)`(<@}.))"0 _
sub =: 1 0 2 <@;@:{"1 {:@[ ,. where presuf"1 >@]
step =: (>rules) <@:sub"1 0/ ]
```

`step` receives a set of not yet expanded words W and applies the `sub` function to all elements of the product $R \times W$ where R is given as a list of pairs of words.
`sub` in turn looks for occurrences of the left hand side of any entry in R using `where`, splits the word accordingly, adds the right hand side of the current rule in front, rearranges these three parts using the permutation 1 0 2 and joins them into a single word.
`where` receives the same arguments, finds the indices of occurrences and adds both 0 and the infix length to them
`presuf` extracts a prefix and a suffix of the word accordingly.

So `sub` and `step` actually implement the functions ρ and R (on sets) given above. The definition of $L(G)$ above can be read with different precedence of union and intersection. The value is the same either way, but the implementation benefits from separating results (words $w \in \mathcal{T}^*$) outside of ρ's domain from words that may still get expanded.

`terminal` tells whether a word is a result (consists entirely of terminal symbols)
`filterminal` selects just these words
`union` implements set union by joining the set difference

```
NB. === advance to next iteration  ===
terminal =: (;term_symbols) */./@:e.~ >
filterminal =: #~ terminal"0
union =: ],-.
```

The function `explore` is written in an imperative style. It returns immediately in case the iteration count is exhausted `yet`. Otherwise a new list of results is obtained and joins the list of all results. Any not yet explored words neither seen already (`done`) nor determined to be results make for a new list of words to explore. This list gets added to `done` already. Should it be empty, the entire set $\bigcup_{\mathbb{N}_0} R^n(\{(S)\})$ has been explored, and the iteration converges.
A not exhausted list however gets processed by `step`, the result of which is made unique. So `explore` updates both `res` and `done` according to the words given, and returns the next generation of words, such that it lends itself to an iterative breadth first search.

```
explore =: 3 : 0
  if. iter = yet =: >: yet do. y return. end.
  res =: res union newres =. filterminal y
  done =: done, newdone =. y -. newres, done
  if. 0 = #newdone do.
    echo 'converged after ', ' steps',~ ": <: yet
    y return.
  end.
  ~. ; step newdone
)
```

Not only the input file name but also the number of rule application steps need to be provided on the command line, either a natural number or positive infinity for convergence:

```
NB. === show usage if args are bad ===
USAGE =: 0 : 0
script usage:
./grammar.ijs -js "grammar=.\'<filename>\' iter=.<number>"
)
hasdef =: 13 : '0 <: 4!:0 y'
0"_@".&>"(0) 3 }. ARGV
{{ if. -.*./hasdef'grammar';'iter' do. exit 1[echo USAGE end. }}0
```

Before the process gets started, the accumulation lists `res` and `done` need to be initialized as empty. Now the definition can be written just as above: apply the step `iter` many times to the list consisting of only the start symbol, filter for results one last time, and join this to all previous results.

```
NB. === run iteration, show result ===
res =: done =: a: $~ yet =: 0
result =: res union filterminal explore^:_ ,<start_symbol
exit 0 [ echo 'results', ": result
```

2. Case Studies

2.1. The Partition Problem: A Mathematical Model

Given $n \in \mathbb{N}$, some set V, an n-tuple $v = (v_i)_{I_n} \in V^n$, and $\pi : V \to \mathbb{R}^+$, let $p(v, N)$ iff $N \subset I_n, \sum_N \pi(v_n) = \sum_{\overline{N}} \pi(v_n)$.

Tell whether $\exists_{J \in 2^{I_n}} p(v, J)$ holds or not. If so, give an example $(J, \overline{J})$ satisfying $p(v, J)$.

In the discrete partition case studied in the following section, π is restricted to the codomain $\mathbb{N}$. (cf. [1])

Obviously, $p(v, J)$ holds true iff so does $p(v, \overline{J})$. Any elements of 2^{I_n} satisfying $p(v, \cdot)$ (or a lack thereof) provide a successful answer which may thus be given by n bits.

2.2. A Grammar for the Partition Problem

from → to		#	explanation
$S \to LM(A_i)R$		1	(A_i) are input symbols representing the values (v_i)
$M \to pMn$		1	M produces k many ps on its lhs and ns on its rhs
$M \to \varepsilon$		1	M disappears, k could be $\frac{1}{2}\sum v_i$ if natural
$pn \to np$		1	ns let ps go past
$n^{v_i}A_i \to \nu$		n	an input symbol A_i cancels as „negative“ v_i many ns
$p^{v_i}A_i \to \pi$		n	an input symbol A_i cancels as „positive“ v_i many ps
$p\nu \to \nu p$	$p\pi \to \pi p$	2	positive/negative cancellations go past ps
$n\nu \to \nu n$	$n\pi \to \pi n$	2	positive/negative cancellations go past ns
$L\nu \to \nu L$	$L\nu \to \nu L$	2	positive/negative cancellations extend the result on the left
$LR \to \varepsilon$		1	once no n, p, A_i, ν, π or M is left between L and R, a word of the language remains to their left

2.3. Impossible Partition; Single and Multiple Solutions

The Monte Carlo method (cf. [2]) suffers from never knowing *for certain* no solution exists. So here only J results are shown. The cases looked into are $(1,1,1,1,6)$ and $(1,1,5,5,6)$. Where there is a solution, both implementations can be assessed and compared.

problem	(1,1,3,4,5)		(2,2,3,4,5)		(1,1,1,1,2)		(1,2,2,3,3)	
	# steps	time	# steps	time	# steps	time	# steps	time
Monte Carlo	69	0.036s	76	2m51.941s	?	>6m0.000s	?	>6m0.000s
custom J	69	1.153s	76	3.752s	49	0.071s	112	2.629s

Obviously, the J implementation is more reliable. The Monte Carlo based program takes far too long when exposed to problems with multiple solutions, whereas the custom J implementation provided the correct set of answers quickly.

2.4. Conclusion

Using a complete terminating algorithm and choosing a tidy language where tacit and explicit code can be mixed both helps with the domain covered, running times (shown for small problem specimens only) and with code readability and thus maintenance.

References

[1] R. M. Karp, Reducibility Among Combinatorial Problems. In: R. E. Miller, J. W. Thatcher (eds.), *Complexity of Computer Computations*. The IBM Research Symposia Series, Plenum Press, New York, 1972, 85–103.

[2] D. E. Knuth, *The Art of Computer Programming. Volume II: Seminumerical Algorithms*. Third edition edition, Addison Wesley, Boston, MA, 1997.

[3] R. Stokes, *Learing* J*: An Introduction to the* J *Programming Language*. 2015. https://www.jsoftware.com/help/learning/contents.htm

THEORIE-AG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 31–35.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

The Relative Strength of #SAT Proof Systems

Olaf Beyersdorff[(A)] Johannes K. Fichte[(B)] Markus Hecher[(C)]
Tim Hoffmann[(A)] Kaspar Kasche[(A)]

[(A)]Friedrich Schiller University Jena, Germany
{olaf.beyersdorff,hoffmann.t,kaspar.kasche}@uni-jena.de

[(B)]Linköping University, Sweden
johannes.fichte@liu.se

[(C)]Massachusetts Institute of Technology, Cambridge MA, USA
hecher@mit.edu

Abstract

The propositional model counting problem #SAT asks to compute the number of satisfying assignments for a given propositional formula. Recently, three #SAT proof systems kcps (knowledge compilation proof system), MICE (model counting induction by claim extension), and CPOG (certified partitioned-operation graphs) have been introduced with the aim to model #SAT solving and enable proof logging for solvers.

Prior to this paper, the relations between these proof systems have been unclear and very few proof complexity results are known. We completely determine the simulation order of the three systems, establishing that CPOG simulates both MICE and kcps, while MICE and kcps are exponentially incomparable. This implies that CPOG is strictly stronger than the other two systems.

Introduction

The *propositional model counting problem* #SAT asks to compute the number of satisfying assignments for a given propositional formula [1]. The #SAT framework allows to efficiently encode and solve many real-world problems from areas such as probabilistic reasoning, risk analysis and explainable artificial intelligence. Interestingly, #SAT is among the hardest combinatorial problems and known to be #P-complete [11]. To put this in relation, by Toda's Theorem [10] any problem from the polynomial hierarchy (PH) can be solved in polynomial time by access to a #SAT oracle. In comparison, the SAT problem is on the first level of PH [6].

Over the last two decades, researchers and solver engineers improved effective *practical #SAT solving* with numerous available #SAT solvers using conceptually quite different approaches. An annual competition captures current trends of solvers and novel practical algorithms, but also reveals that correctness needs to be improved [8].

This paper was accepted at SAT 2024.

Parts of the work has been carried out while the first four authors were visiting the Simons Institute for Theory of Computation at UC Berkeley.

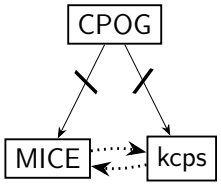


Figure 1: Simulation order of CPOG, MICE and kcps. A solid crossed edge from A to B indicates that A p-simulates B and A is exponentially stronger than B. Dotted lines indicate incomparability.

In contrast to these practical advances, little is known theoretically on the power and limitations of #SAT solving. In both SAT and quantified Boolean formulas (QBF), the main theoretical approach towards gauging the strength of SAT and QBF solvers is through *proof systems and proof complexity*. The relation between proofs and solving is important in at least two aspects.

Firstly, proof systems can *model aspects of solving*. A seminal result in this direction is that CDCL solvers – the predominant approach in SAT solving – tightly correspond to propositional resolution, in the sense that any (non-deterministic) CDCL run on an unsatisfiable formula can be efficiently translated into a resolution refutation of the formula and vice versa. Further results are known for practical CDCL and relations between QBF solving and related proof systems. This allows to apply the plethora of proof complexity results e.g. for propositional and QBF resolution to the analysis of solvers. For example any lower bound for proof size in propositional resolution directly translates into a lower bound for CDCL runtime.

Secondly, proof systems can be employed for *proof logging and certifying solver correctness* by designing certified tools. Therefore, formal proof systems are introduced where a practical proof can be efficiently verified by a relatively simple method and easily emitted during solving. Different proof systems and formats have been designed including RUP, RAT and DRAT for SAT and QRAT for QBF. These proof systems have been intensively studied and compared in terms of simulations. While the first modelling aspect needs weaker proof systems close to actual solving, the second proof-logging aspect favours very strong proof systems.

In comparison to the rich and intensely researched interplay between solving and proof complexity in SAT and QBF, significantly less is known in this regard for #SAT. In the past five years, *three different #SAT proof systems* have been introduced. These systems are kcps (2019) [5], MICE (2022) [2, 9], and CPOG (2023) [4]. These are the only #SAT proof systems so far.

The three proof systems are conceptually quite different: while kcps and CPOG are both static proof systems building on circuit classes used in knowledge compilation [7] on which model counting is efficient, MICE is a rule-based proof system using three simple rules to compute counts for successively more complex formulas. The historically first system kcps was inspired by #SAT solving using knowledge compilation techniques. Both subsequent systems MICE and CPOG were designed with a view towards certifying different #SAT solving approaches.

In contrast to the rich proof complexity results for SAT and QBF, almost nothing is known theoretically for the three #SAT proof systems. Only for MICE, an exponential proof size lower bound was shown last year [2], while the relations between the three systems in terms of simulations are open.

Our contributions

We perform a proof-complexity analysis of the three proof systems kcps, MICE and CPOG and completely determine their relative strength in terms of simulations and separations, leading to the picture in Figure 1. In more detail, our findings can be summarised as follows.

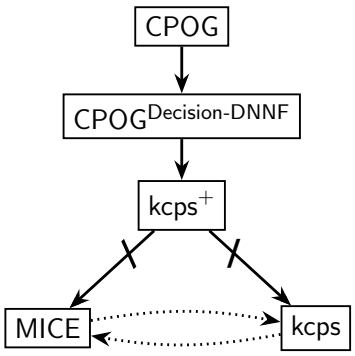


Figure 2: Detailed simulation order of #SAT proof systems. A solid edge from A to B indicates that A p-simulates B. If the edge is crossed, A is also exponentially separated from B. A dotted edge from A to B indicates that A is exponentially separated from B. All the simulations of this paper require only logarithmic space; those highlighted with "$*$" only need linear time.

A. Simulations between #SAT proof systems. We formally compare the three #SAT proof systems in terms of *simulations* and show that CPOG p-simulates kcps and MICE, i.e. both kcps and MICE proofs can be efficiently translated into CPOG proofs.

Rather than showing the two simulations directly, we consider two intermediate proof systems kcps$^+$ and CPOG$^{\text{Decision-DNNF}}$. The first of these was already suggested in [5] as a natural extension of kcps, while CPOG$^{\text{Decision-DNNF}}$ is newly introduced as a restriction of CPOG. The system CPOG uses POGs (partitioned-operation graphs) as the underlying circuit class, which in CPOG$^{\text{Decision-DNNF}}$ is restricted to Decision-DNNFs: the circuit class on which kcps and kcps$^+$ are based. Representing a CNF by any of these circuit models allows efficient counting. Yet, the proofs need to contain additional information as verifying the equivalence of a CNF to a circuit in these models is non-trivial. While the two additional systems simplify our analysis, we believe they are also natural and of independent interest for further research (cf. the discussion in the conclusion).

For these five proof systems, we determine the simulation order as depicted in Figure 2, refining Figure 1 and including pointers to the results. While the simulations of kcps by kcps$^+$ and of CPOG$^{\text{Decision-DNNF}}$ by CPOG follow almost by definition, the simulations of MICE by kcps$^+$ and kcps$^+$ by CPOG$^{\text{Decision-DNNF}}$ are more involved – in particular the first one – as they connect conceptually quite different proof formats. The proof systems kcps, kcps$^+$, CPOG$^{\text{Decision-DNNF}}$ and CPOG are all *static* as they are based on circuits equipped with additional information. In contrast, MICE is *rule-based* without any explicit connection to circuits.[1]

B. Exponential separations between #SAT proof systems. As our second main result we establish exponential separations between MICE and kcps in both directions. This entails exhibiting suitable CNF families that have short proofs in MICE, while requiring short kcps proofs, and vice versa. As a consequence, both systems are incomparable and at the same time exponentially weaker than kcps$^+$ and CPOG, thus resulting in the situation depicted in Figure 1.

Technically, we obtain one direction (kcps does not simulate MICE) by showing a tight characterisation of kcps proof size by regular resolution size on unsatisfiable formulas. A similar characterisation of MICE by full resolution was shown in [2]. As regular resolution is known to be exponentially weaker than resolution, the separation follows.

[1] However, it was noted already in [3] that from a MICE proof a Decision-DNNF for the CNF can be extracted. This does not, however, entail a simulation of MICE by kcps (which are based on Decision-DNNFs), and in fact such a simulation *fails* as implied by our separation results in B.

For the other direction (MICE does not simulate kcps) we use a variant of the pebbling formulas, prominent in propositional proof complexity. While the small Decision-DNNFs and short kcps proofs are relatively easy to obtain, the hardness argument for MICE is technically more involved.

The first separation positively answers an open question posed by Capelli [5] to find CNFs with polynomial-size Decision-DNNFs (these can always be extracted from short MICE proofs [3]), but no small kcps proofs. The second separation implies that we cannot efficiently transform Decision-DNNFs into MICE proofs.

References

[1] *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, 2021.

[2] Olaf Beyersdorff, Tim Hoffmann, and Luc Nicolas Spachmann. Proof complexity of propositional model counting. In *SAT'23*, volume 271, pages 2:1–2:18, 2023.

[3] Olaf Beyersdorff, Tim Hoffmann, and Luc Nicolas Spachmann. Proof complexity of propositional model counting. *ECCC*, pages TR24–030, 2024.

[4] Randal E. Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn J. H. Heule. Certified knowledge compilation with application to verified model counting. In *SAT'23*, volume 271, pages 6:1–6:20, 2023.

[5] Florent Capelli. Knowledge compilation languages as proof systems. In *SAT'19*, volume 11628, pages 90–99, 2019.

[6] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*, volume 43, pages 143–152. 2023.

[7] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *JAIR*, 17:229–264, 2002.

[8] Johannes K. Fichte, Markus Hecher, and Florim Hamiti. The model counting competition 2020. *JEA*, 26(1):1–26, 2021.

[9] Johannes Klaus Fichte, Markus Hecher, and Valentin Roland. Proofs for propositional model counting. In *SAT'22*, volume 236, pages 30:1–30:24, 2022.

[10] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

[11] Leslie G. Valiant. The complexity of computing the permanent. *TCS*, 8(2):189–201, 1979.

THEORIE-AG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 36–39.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Strukturelle und algorithmische Aspekte von Solitonautomaten

Henning Bordihn[(A)] Helena Schulz[(B)]

[(A)]Institut für Informatik und Computational Science
Universität Potsdam
henning@cs.uni-potsdam.de

[(B)]Fakultät für Elektrotechnik und Informatik,
Technische Universität Berlin
schulz-helena@gmx.de

Zusammenfassung

Es werden verschiedene Konzepte des Determinismus für die Varainte von Soliton-Automaten mit mehreren Solitonen definiert. Die Begriffe der Single- und Multi-Solitonautomaten werden weiter harmonisiert, so dass Modellkonsistenz auch in Bezug auf die Konzepte des Determinismus besser erreicht wird. Verschiedene Ergebnisse hinsichtlich der Determinismuskonzepte werden präsentiert. Ferner werden Korrektheit und Terminieren eines Algorithmus diskutiert, der alle Pfade berechnet, auf denen Solitone den Solitongraphen traversieren können, der dem Solitonautomaten zugrundeliegt.

1. Einleitung

Solitonautomaten bilden ein Modell, das auf Schaltverhalten auf molekularer Ebene basiert. Die Struktur bestimmter Moleküle wird duch die Traversion von Solitonen, gewissen stabilen Störungen, verändert. Betrachtet man die verschiedenen, auf diese Weise auseinander hervorgehenden Moleküle als Zustände, so erhält man ein System mit Automatenverhalten. Die Eingabezeichen für so einen Automaten sind Paare von Ein- und Austrittspunkten für Solitone im Molekül, somit Paare von externen Knoten im Solitongraphen, der das Molekül in abstrakter Form repräsentiert. Für einen kurzen Überblick über die Geschichte der Solitone verweisen wir auf [11] und [12, S. 18-19]. Eine umfangreiche Liste von Referenzen zu Solitonberechnungen und Solitonautomaten findet sich in [1].

Die ursprünglich betrachtete Variante der Solitonautomaten war darauf beschränkt, dass sich zu jedem Zeitpunkt höchstens ein Soliton in einem Molekül befindet [4, 6]. Für dieses Modell wurden verschiedene Determinsismuskonzepte eingeführt und Charakterisierungen gegeben [5, 6, 7, 8]. Ferner wurden Solitonautomaten nach ihren Transitionshalbgruppen klassifiziert [6]. In [9] wurde gezeigt, wie sich alle endlichen Automaten als Produkte von Solitonautomaten darstellen lassen.

Später wurden Solitonautomaten betrachtet, in denen sich mehrere Solitone gleichzeitig durch das zugrundeliegende Molekül bewegen können [1]. Die Eingabezeichen sind hier *Bursts* von Solitonen. Ein Burst legt fest, mit welchen diskreten zeitlichen Abständen sich Solitone

von welchem externen Eingangsknoten zu welchem externen Ausgangsknoten bewegen. Der Single-Solitonautomat ist ein Spezialfall des Multi-Solitonautomaten. In [10] wurde gezeigt, dass im Gegensatz zu Single-Solitonautomaten Bursts nicht umkehrbar sind, aber zu jedem Burst ein Burst konstruiert werden kann, der die Solitoneffekte rückgängig macht. Ferner wurde die Kombinierbarkeit von mehreren Bursts zu einem Burst mit gleichem Solitoneffekt konstruktiv gezeigt.

In [3, 14] wurden die für Single-Solitonautomaten bekannten Definitionen von Determinismus angepasst und um weitere Konzepte ergänzt. Einige Ergebnisse zu den Determinismusbegriffen in Multi-Solitonautomaten wurden in [2] gegeben. Außerdem findet sich in [14] die Beschreibung einer Software zur Simulation von Single- und Multi-Solitonautomaten [13]. Entscheidend hierfür ist die Berechnung aller Pfade in gegebenen Solitongraphen, die Solitone beim Traversieren nehmen können. Eine Diskussion über die Korrektheit und das Terminieren des entsprechenden Algorithmus findet sich in [3].

2. Determinismuskonzepte

Für den Single-Solitonautomaten werden Determinismus und starker Determinismus unterschieden [6]. Ein Solitonautomat heißt deterministisch, wenn für jeden seiner Zustände (jedes durch Solitontraversionen erreichbare Molekül) und jedes Eingabezeichen (jedes Paar externer Knoten der zugehörigen Solitongraphen) höchstens ein Nachfolgezustand existiert. Bei starkem Determinismus gibt es zudem für jeden Zustand und jedes Eingabezeichen nur einen Pfad, den das Soliton nehmen kann. Diese Konzepte lassen sich auf Multi-Solitonautomaten übertragen. Für Multi-Solitonautomaten sind eine Reihe weiterer Determinsimuskonzepte eingeführt worden [14, 2]. Insbesondere wurde der Grad des Nichtdeterminismus betrachtet, der angibt, wie viele Nachfolgezustände je Zustand und Eingabezeichen auftreten können. In [2] ist gezeigt, dass der Grad des Nichtdeterminismus für Solitonautomaten ein verbundenes Maß ist, das heißt für jede ganze Zahl $g \geq 1$ existiert ein Solitonautomat, dessen Grad des Nichtdeterminismus genau g ist. Ferner wurde gezeigt, dass Multi-Solitonautomaten genau dann stark deterministisch sind, wenn der zugrundeliegende Graph ein Baum ist. Damit unterscheidet sich die Charakterisierung des starken Determinsimus vom Single-Solitonfall, für den neben Bäumen auch sogenannte Kastanien stark deterministisch sind.

3. Simulation von Solitonautomaten

Die im Zusammenhang mit der Bachelorarbeit [14] entwickelte Software zur Simulation von Solitonautomaten ist zum Experimentieren und Demonstrieren des Konzepts der Solitonautomaten geeignet. Die verschiedenen Determinismuseigenschaften werden für eingegebene Solitonautomaten entschieden, ihr Grad des Nichtdeterminismus wird berechnet [13].

Diese Software basiert auf einem Algorithmus, der alle Solitonpfade für einen Solitongraphen ermittelt. Da die sich durch die Passage von Solitonen verändernden chemischen Bindungen, dargestellt als Kantengewichte, berücksichtigt werden müssen, wurde ein Ansatz nach dem Prinzip der Tiefensuche gewählt. Um zu erkennen, unter welchen Umständen ein Backtracking zu erfolgen hat, mussten Bedingungen definiert werden, die garantieren, dass ein weiterer Abstieg in die Tiefe keine neuen Pfade mehr erschließen wird. Zu diesem Zweck wurde

das Konzept der Nachfolge-Äquivalenz von Konfigurationen des Solitonautomaten definiert und ausgenutzt, so dass ein Terminieren des Tiefensuche-Algorithmus garantiert werden kann.

4. Ausblick

Abschließend werden einige offene Probleme für zukünftige Forschungen aufgelistet.

1. Für Single-Solitonautomaten ist bekannt, dass bestimmte Pfade in Solitongraphen für jegliche Solitone unpassierbar sind. In den bekannten Beispielen werden diese Pfade durchlässig, wenn Bursts von Solitonen zugelassen werden. Eine interessante Forschungsfrage ist, ob undurchlässige Pfade in Solitongraphen im Fall von Multi-Solitonautomaten überhaupt auftreten können. Ein Soliton, das einen Knoten eines für dieses Soliton undurchlässigen Pfades passiert, öffnet den Weg für ein zweites Soliton, das folgt. Die Frage ist, ob dieses Prinzip so verallgemeinert werden kann, dass eine unbegrenzte Anzahl von undurchlässigen Pfaden geöffnet werden kann, die hintereinander „versteckt“ auftreten können, ohne dass es zu Kollisionen kommt, so dass jedes Soliton den Graphen wieder verlassen kann und es zu einem vollständigen Zustandsübergang kommt.

2. Zusätzlich zur Charakterisierung von stark deterministischen Solitonengraphen könnte man auch versuchen deterministische Solitongraphen entsprechend weiterer der eingeführten Determinismuskonzepte zu charakterisieren.

3. Welche Determinismuseigenschaften bleiben unter der Anwedung von Operationen auf Solitonautomaten (wie z.B. Komposition) erhalten?

4. Ein weiteres Feld für zukünftige Forschung ist die Untersuchung der Transitionshalbgruppen von Multi-Solitonautomaten.

Literatur

[1] H. BORDIHN, H. JÜRGENSEN, Multi-wave soliton automata. *J. Autom. Lang. Comb.* **27** (2022) 1–3, 91–130.

[2] H. BORDIHN, H. SCHULZ, Determinism in Multi-Soliton Automata. In: F. MANEA, G. PIGHIZZINI (eds.), *Proceedings 14th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2024)*. Electronic Proceedings of Theoretical Computer Science (EPTCS) 407, 2024, 47–59. To appear.

[3] H. BORDIHN, H. SCHULZ, Determinism and Simulation of Soliton Automata. *Journal of Automata, Languages and Combinatorics* (2025). Accepted at JALC.

[4] J. DASSOW, H. JÜRGENSEN, Soliton Automata. In: L. BUDACH, R. G. BAKHARAJEV, O. B. LIPANOV (eds.), *Fundamentals of Computation Theory, International Conference FCT'87, Kazan, USSR, June 22-26, 1987, Proceedings*. Lecture Notes in Computer Science 278, Springer-Verlag, Berlin, 1987, 95–102.

[5] J. DASSOW, H. JÜRGENSEN, Deterministic soliton automata. In: *Proceedings of Toyohashi Symposium on Theoretical Computer Science, August 30 – September 1, 1990, Toyohashi, Japan*. 1990, p. 33. The full paper was delayed in the mail. Therefore, the proceedings only contain an abstract. The full paper was distributed together with the proceedings (5 pp.).

[6] J. DASSOW, H. JÜRGENSEN, Soliton Automata. *J. Comput. Syst. Sci.* **40** (1990) 2, 154–181.

[7] J. DASSOW, H. JÜRGENSEN, Deterministic Soliton Automata with a Single Exterior Node. *Theor. Comput. Sci.* **84** (1991) 2, 281–292.

[8] J. DASSOW, H. JÜRGENSEN, Deterministic Soliton Automata with at Most One Cycle. *J. Comput. Syst. Sci.* **46** (1993) 2, 155–197.

[9] F. GÉCSEG, H. JÜRGENSEN, Automata Represented by Products of Soliton Automata. *Theor. Comput. Sci.* **74** (1990) 2, 163–181.

[10] T. KOSS, Reverting and combining soliton bursts. *J. Autom. Lang. Comb.* **27** (2022) 1–3, 179–186.

[11] P. S. LOMDAHL, What is a soliton? *Los Alamos Science* **10** (1984), 27–31.

[12] Y. LU, *Solitons & Polarons in Conducting Polymers*. World Scientific, Singapore, 1988.

[13] H. SCHULZ. `https://github.com/schulz-helena/soliton-implementation`. GitHub Repository 2022.

[14] H. SCHULZ, *Untersuchungen zum Determinismuskonzept bei Mehrwellen-Soliton-Automaten anhand einer zu implementierenden Simulation*. Bsc thesis, Universität Potsdam, 2023.

THEORIE-AG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 40–43.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Deploying a New Proof in Formal Language Theory with an Eye to Formalisation

Marco B. Caminati[A]

[A] School of Computing and Communications
Lancaster University in Leipzig
Nikolaistrasse 10
04109 Leipzig
Germany
m.caminati@lancaster.ac.uk

Abstract

In a recent paper, I introduced a new proof of a recent result linking formal language theory and computational group theory, one goal being to make its formalisation more feasible. I will discuss the state of this.

1. Introduction

In the traditional rendition of formal grammar, we found rules expressed as a finite concatenation of terminals and non-terminals on the left, then an arrow pointing to the right, and then an arbitrary concatenation of terminals and non-terminals on the right of the arrow. Context-freeness imposes that the concatenation on the left hand side of the arrow consist of exactly one non-terminal. This allows us to give an implicational interpretation of each rule by reversing the arrow, by looking at each concatenation operation on the right-hand side of the arrow as a logical conjunction, and by considering each non-terminal as a unary logical predicate, taking a string of terminals and returning true or false based on whether its argument can or cannot be generated by the grammar.

To make a concrete example, consider a fragment of a toy English grammar with S as the starting non-terminal and a standard rule $S \to NP\,VP$ to allow us generating simple sentences having a subject and a verb. In the implicational view, S takes one terminal string as an argument, and returns true exactly when that argument can be obtained by concatenating arguments of non-terminals on the right-hand side of the rule when the latter is a valid logical inference. Therefore,

$$S(\text{James shouts}) \leftarrow NP(\text{James}) \wedge VP(\text{shouts})$$

being a valid logical implication (since both $NP(\text{James})$ and $VP(\text{shouts})$ return true), tells us that $S(\text{James shouts})$ also returns true and therefore its argument is a string of the grammar. On

[A] The author is grateful to the Hausdorff Research Institute for Mathematics in Bonn, Germany, for hosting him as part of the trimester "Prospects of formal mathematics," funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2047/1 – 390685813. This work was initiated and partially developed during the author's stay at the trimester.

the contrary, "shouts James" is not, since we cannot find an analogous instantiation of the given rule.

Now, what happens if we allow each non-terminal to take not only one argument, but a finite number, called its *fanout*? Obviously, we obtain a generalisation of context-free grammars, and it is natural to ask how much more powerful these are, compared to context-free grammars. Formally, we are now looking at rules of the form

$$A\left(y_0,\ldots,y_{f(A)}\right) \leftarrow B_0(x_{0,0},\ldots,x_{0,f(B_0)}),\ldots,B_m\left(x_{m,0},\ldots,x_{m,f(B_m)}\right),$$

where f is a map returning the fanout of each non-terminal, the x's are variables (placeholders), and the y's are concatenated expressions specifying how the $x_{i,j}$ and the terminals are *composed* together to form the arguments of A.

One first step in our investigation is to further categorise the family of grammars thus obtained, by imposing additional constraints to generate sub-families. We will be looking at *multiple context-free grammars* (MCFGs); that is, grammars with rules of the form above where, additionally, we have:

- linearity in the $x_{i,j}$'s;
- pairwise distinctness of the $x_{i,j}$'s.

2. O_2 as a testbed for MCFGs

Consider the alphabet of pairwise conjugated letters $\Sigma_2 := \left\{a,\overline{a},b,\overline{b}\right\}$, where the overbar denotes conjugacy. O_2 is the language of strings in that alphabet having as many a's as $\overline{a}$'s and as many b's as $\overline{b}$'s. It is one member of the family of *balanced* languages:

$$O_n := \left\{w \in \{a_0,\overline{a_0},\ldots,a_{n-1},\overline{a_{n-1}}\}^* :\ |w|_{a_j} = |w|_{\overline{a_j}}\right\}.$$

Another way to characterise a balanced language is via the introduction of the ancillary concept of *balance*: given a string w of Σ_n, its balance $\mu(w)$ is the n-tuple of integers $\left(|w|_{a_1} - |w|_{\overline{a_1}},\ldots,|w|_{a_n} - |w|_{\overline{a_n}}\right)$, making a string balanced if and only if all the entries of its balance are 0.

Note that, for typographical convenience, we used a and b in lieu of a_0, a_1 in the case $n = 2$. The alphabet of O_n will be here denoted as Σ_n.

Now, as a simple non-context free language, O_2 is a reasonable testbed to assess the power of MCFGs, also because the question of whether O_n are multiple context-free languages (MCFLs) relates to others in computational linguistics and computational group theory [5]. The question was answered in the positive as follows:

1. Salvati answered for O_2 in his seminal paper [5] using a *geometrical* proof,
2. then revised/simplified by Nederhof in [4];
3. Then, Ho answered in the positive the question for O_n, but with bigger fanout than needed [3].
4. Finally, Gebhardt, Meunier, and Salvati showed that O_n is a MCFL of fanout n for any n [2].

3. Bump-based proof

The MCFG, G_2, we will use is as follows:

$$r_0 : I(\varepsilon,\varepsilon) \leftarrow \qquad r_l : I(vxw,y) \leftarrow I(v,w), I(x,y)$$
$$r_a : I(a,\overline{a}) \leftarrow \qquad r_r : I(v,xwy) \leftarrow I(v,w), I(x,y)$$
$$r_{\overline{a}} : I(\overline{a},a) \leftarrow \qquad r_n : I(vx,yw) \leftarrow I(v,w), I(x,y)$$
$$r_b : I\left(b,\overline{b}\right) \leftarrow \qquad r_s : I(vx,wy) \leftarrow I(v,w), I(x,y)$$
$$r_{\overline{b}} : I\left(\overline{b},b\right) \leftarrow \qquad r_z : S(vw) \leftarrow I(v,w)$$

Obviously, $L(G_2) \subseteq O_2$, so the question boils down to showing that any balanced string of O_2 can be generated by G_2. Additionally, since there is only one meaningful non-terminal, I, we can formally drop it, and we are left to show that it is possible to split *any* pair of factors of *any* balanced string into two pairs of factors of smaller balanced strings. This is called a *balanced decomposition* of the original pair. So now we are given a pair (p,q) such that $pq \in O_2$, and need to obtain smaller strings in O_2 with at most two cuts. In the original geometrical view, p is a walk of unit steps along the axes in the plane from the origin O to some point P, and q is a walk back to O. The balancedness corresponds to the closedness of the walk (as a curve).

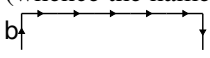
We will start an elementary, non-geometrical proof starting from the non-geometrical (although it does have an obvious geometrical interpretation) of *shortness*: a (possibly non-balanced) string of O_n is short if no string with the same balance is shorter. A tuple of short strings will also be termed short. Now, one first interesting fact is that there is an elementary proof of the following Lemma for the short case (where $\underline{z}$ is the reversal of the string z):

Lemma 3.1 *Let (x,y) be a proper, short factorisation of a string of O_2 of length > 2. Then, either $\overline{x}(0) = \underline{y}(0)$, or $\underline{x}(0) = \overline{y}(0)$, or there are p and q proper left factors of x and y, respectively, such that $pq \in O_2$.*

Corollary 3.1 *Any proper, short, binary factorisation of a string of O_2 of length bigger than 2 admits a balanced decomposition.*

Proof. Lemma 3.1 asserts that it is possible to find such a decomposition of a very special form (equivalent to using either r_s or a restricted form of r_n). Hence, the corollary follows immediately. □

The short case just established acts as the base case for an inductive proof as follows:

- We introduce the notion of bump: a *minimal* non-short string.
- For $n = 2$, it must have the form (whence the name) $\alpha\beta^m\overline{\alpha}$, with $\beta \notin \{\alpha,\overline{\alpha}\}$, where α is called the *direction* of the bump: 

- For any string p, $B(p)$ contains the start and end of all the factors of p which are bumps.
- $B^X(p)$ is similar, but only refers to bumps having direction comprised in a set of allowed directions X.

- Obviously, p is short if and only if $B(p) = \emptyset$.
- We reduce to the short case by removing bumps one at a time, and always a bump of *minimal length*.
- Once it is removed, by induction, we obtain a balanced pair which is decomposable.

The fact that a minimal bump can safely be removed is sanctioned by the following Lemma (where $u \preceq v$ means that u is a left factor of v):

Lemma 3.2 *Let s_0, s_1 be non-empty strings of Σ_2^*. Assume that*

1. $\mathcal{B} := [i, 1+i+k] \in B^{\alpha}(s_0) \cap \text{argmin}_{\text{length}}\left(B^{\{\alpha,\overline{\alpha}\}}(s_0) \cup B^{\{\alpha,\overline{\alpha}\}}(s_1)\right)$,
2. (s_0, s_1) *is balanced but not bal-decomposable,*
3. *for any p_0 and p_1 being substrings of s_0 and of s_1, respectively, such that $p_0p_1 \in O_2$ and $|p_0p_1| < |s_0s_1|$, it holds that (p_0, p_1) is balanced-decomposable, and*
4. $k < |s_0| - 1$.

Then $i = 0$, $k \geq 2$, and $\overline{s_0(i+1)}^j \alpha \preceq x$ for some $x \in \{\underline{s}_0, s_1, \underline{s}_1\}$ and $1 \leq j < k$.

The inductive bump removal enabled by the previous lemma allows us to always reduce to the base short case, thereby obtaining the final theorem [1]:

Theorem 3.2 *Any proper, binary factorisation of a string of O_2 of length bigger than 2 admits a balanced decomposition. In other words, the set $\{(z_0, z_1) \,.\, z_0z_1 \in O_2,\ |z_0z_1| > 2,\ \min\{|z_0|, |z_1|\} > 0, (z_0, z_1)$ admits no balanced decomposition$\}$ is empty.*

References

[1] MARCO B CAMINATI, O_2 is a Multiple Context-Free Grammar: An Implementation-, Formalisation-Friendly Proof. In: *International Conference on Developments in Language Theory*. Springer, 2024, 82–97.

[2] KILIAN GEBHARDT, FRÉDÉRIC MEUNIER, SYLVAIN SALVATI, O_n is an n-MCFL. *Journal of Computer and System Sciences* **127** (2022), 41–52.

[3] MENG-CHE HO, The word problem of $\mathbb{Z}_n$ is a multiple context-free language. *Groups Complexity Cryptology* **10** (2018) 1, 9–15.

[4] MARK-JAN NEDERHOF, A short proof that O_2 is an MCFL. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2016, 1117–1126.

[5] SYLVAIN SALVATI, MIX is a 2-MCFL and the word problem in $\mathbb{Z}_2$ is captured by the IO and the OI hierarchies. *Journal of Computer and System Sciences* **81** (2015) 7, 1252–1277.

THEORIE-AG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 44–44.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Word Equations for Information Extraction on Text

Joel D. Day[(A)]

[(A)]Department of Computer Science,
Loughborough University,
Loughborough,
UK,
LE11 3TU
J.Day@lboro.ac.uk

Abstract

Extracting information and deciding properties of text is a canonical problem in computer science, with a wide variety of individual problems and corresponding solutions. These range from simple "static pattern" matching algorithms to more complex string logics and query languages. As is so often the case, there are compromises to be made: (canonical) regular expressions benefit from the strong computational and mathematical properties of regular languages but in many applications are too limited in what they can express, resulting in commonly used extensions via backreferences or variables. These extensions facilitate the identification and extraction of possibly repeated elements which are not known a priori, but come at the cost of greater computational complexity. Other models, such as document spanners provide relational operators mirroring those canonically occurring in the context of relational databases.

The idea of querying textual data in the same way one might query a database makes a lot of sense: there are many settings in which large amounts of data are stored in text. Naturally therefore, several novel formalisations have been introduced with exactly this in mind. One consequence of a relational perspective on querying or processing strings is inclusion of equality comparisons, which together with the use of variables results in the presence of word equations. Indeed, word equations arguably offer a natural alternative for expressing various properties of strings, and formalisms such as the logic FC and its recursive sibling FC-Datalog embrace the use of word equations as part of the syntax. However, dealing with word equations remains challenging both from a theoretical and practical perspective. While this can lead in some cases to undecidability, FC and FC-Datalog overcome this by restricting word equations to a finite universe consisting of all the factors of a given input string. Nevertheless, word equations in their traditional, infinite-universe setting still play a significant role in understanding these models providing, new problems and further motivation for some existing ones.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 45–45.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Subsequence Matching and Analysis Problems for Formal Languages

Szilárd Zsolt Fazekas[(A)] Tore Koß[(B)] Florin Manea[(B)]
Robert Mercaş[(C)] Timo Specht[(B)]

[(A)]Akita University
szilard.fazekas@ie.akita-u.ac.jp

[(B)]University of Göttingen
{tore.koss@,florin.manea@cs,timo.specht@stud}.uni-goettingen.de

[(C)]Loughborough University
R.G.Mercas@lboro.ac.uk

Abstract

In this work, we study a series of algorithmic problems related to the subsequences occurring in the strings of a given language, under the assumption that this language is succinctly represented by a grammar generating it, or an automaton accepting it. In particular, we focus on the following problems: Given a string w and a language L, does there exist a word of L which has w as subsequence? Do all words of L have w as a subsequence? Given an integer k alongside L, does there exist a word of L which has all strings of length k, over the alphabet of L, as subsequences? Do all words of L have all strings of length k as subsequences? For the last two problems, efficient algorithms were already presented in [1] for the case when L is a regular language, and efficient solutions can be easily obtained for the first two problems. We extend that work as follows: we give sufficient conditions on the class of input-languages, under which these problems are decidable; we provide efficient algorithms for all these problems in the case when the input language is context-free; we show that all problems are undecidable for context-sensitive languages. Finally, we provide a series of initial results related to a class of languages that strictly includes the regular languages and is strictly included in the class of context-sensitive languages, but is incomparable to the of class context-free languages; these results deviate significantly from those reported for language-classes from the Chomsky hierarchy.

This presentation is based on the paper [2], to appear at ISAAC 2024.

References

[1] D. ADAMSON, P. FLEISCHMANN, A. HUCH, T. KOSS, F. MANEA, D. NOWOTKA, k-Universality of Regular Languages. In: S. IWATA, N. KAKIMURA (eds.), *34th International Symposium on Algorithms and Computation (ISAAC 2023)*. LIPIcs 283, 2023, 4:1–4:21. Full version: https://arxiv.org/abs/2311.10658.

[2] S. Z. FAZEKAS, T. KOSS, F. MANEA, R. MERCAŞ, T. SPECHT, Subsequence Matching and Analysis Problems for Formal Languages. In: *35th International Symposium on Algorithms and Computation, ISAAC 2024*. LIPIcs to appear, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

THEORIE-TAG 2024

T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 46–46.

GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

On the Expressibility of Formal Languages via Formulas over Strings

Justus Greve-Kramer[(A)]

[(A)]University of Göttingen
justus.grevekramer@stud.uni-goettingen.de

Abstract

A formula over strings is a logical statement which contains string variables. In this work, we take a closer look at the expressive power of formulas containing language membership constraints and linear letter-count constraints. More precisely, given a formula of strings together with a variable x_0 therein, we study the language of words which can be substituted for x_0 without making the formula unsatisfiable.

We compare the class of expressible languages with well-known classes of formal languages, like the context-free and context-sensitive languages. It turns out that the languages expressible in VPL + LETCOUNT are all context-sensitive, and those expressible in REG + LETCOUNT are even context-free when the underlying alphabet is binary. When the underlying alphabet contains three or more letters, the languages expressible in REG + LETCOUNT are not necessarily context-free. We show that none of the reverse inclusions hold, by examining a language of palindromes.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 47–50.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Deterministic Parikh Automata on Infinite Words (Extended Abstract)

Mario Grobler[(A)] Sebastian Siebertz[(A)]

[(A)]University of Bremen {grobler,siebertz}@uni-bremen.de

Abstract

Various variants of Parikh automata on infinite words have recently been introduced in the literature. However, with some exceptions only their non-deterministic versions have been considered. In this paper we study the deterministic versions of all variants of Parikh automata on infinite words that have not yet been studied. We compare the expressiveness of the deterministic models and investigate their closure properties and decision problems with applications to model checking. The model of deterministic limit Parikh automata turns out to be most interesting, as it is the only deterministic Parikh model generalizing the ω-regular languages, the only deterministic Parikh model closed under the Boolean operations and the only deterministic Parikh model for which all common decision problems are decidable. On our way, we establish tight complexity bounds for the universality problem for deterministic finite word Parikh automata.

1. Introduction

Finite automata operating on infinite words find many applications in the world of formal languages, logic, formal verification, games, and many more. In many scenarios, non-determinism adds expressiveness or succinctness to their deterministic counterparts. However, this often comes at the price of important decision problems becoming hard to solve, or even undecidable [1, 2]. This is not only true for the most common model of finite automata operating on infinite words, namely Büchi automata (which recognize the ω-regular languages), but also for various extensions leaving the ω-regular realm. These include Parikh automata (PA) on infinite words, which gained a lot of attention just recently [3, 5, 6]. Such automata are equipped with a constant number of counters (positive integers), and the counter values can be checked to satisfy some linear constraints. Several acceptance conditions for Parikh automata on infinite words have been considered in the literature, including deterministic and non-deterministic reachability PA, safety PA, Büchi PA and co-Büchi PA [5], as well as non-deterministic limit PA, reachability-regular PA, and strong and weak reset PA [3]. For all of these models, except safety PA and co-Büchi PA, testing emptiness is coNP-complete and universality is undecidable [3, 5]. In contrast, for safety PA and co-Büchi PA, both problems are undecidable; however, requiring determinism yields a coNP-complete universality problem [5].

This motivates the study of the deterministic variants of the remaining models, namely deterministic limit PA, reachability-regular PA, strong reset PA, and weak reset PA. In this paper we investigate their expressiveness, closure properties, and common decision problems.

[(A)]The full version of this paper can be found on arXiv [4]

2. Results

Grobler et al. [3] have shown that almost all non-deterministic variants of the aforementioned models (the exception being safety PA and co-Büchi PA again) form a hierarchy:

$$\begin{aligned}\text{reachability PA} \subseteq \text{reachability-regular PA} &= \text{limit PA}\\ \subseteq \text{Büchi PA} \subseteq \text{weak reset PA} &= \text{strong reset PA}.\end{aligned}$$

First, we show that this is not the case for their deterministic variants. While

$$\text{deterministic strong reset PA} \subseteq \text{deterministic weak reset PA}$$

and

$$\begin{aligned}\text{deterministic reachability PA} \subseteq \text{deterministic reachability-regular PA}\\ \subseteq \text{deterministic weak reset PA}\end{aligned}$$

still holds, all other models become pairwise incomparable. Furthermore, we show that among all studied deterministic models only deterministic limit PA generalize Büchi automata in the sense that they recognize all ω-regular languages.

Deterministic limit PA also shine in the light of closure properties: as we show, among all studied PA operating on infinite words (the deterministic variants as well as the non-deterministic ones) they are the only ones closed under union, intersection and complement.

	$\cup$	$\cap$	$\bar{}$
deterministic limit PA	✓	✓	✓
deterministic reachability-regular PA	✗	✗	✗
deterministic weak reset PA	✗	✗	✗
deterministic strong reset PA	✗	✗	✗

Table 1: Closure properties.

This benefit also yields decidable decision problems. In contrast to the other models that were studied in [3, 5], emptiness and universality are decidable for deterministic limit PA. We show that also strong reset PA benefit from determinism: although having bad closure properties their universality problems becomes decidable. However, as we show, for deterministic reachability-regular PA and deterministic weak reset PA the universality problem remains undecidable.

Finally, we study their intersection-emptiness and inclusion problems, as these are important problems in order to study model checking problems, the core problems in the field of formal verification. In the model checking problems we are given a system $\mathcal{K}$ as a Kripke structure (a safety automaton) or a PA and a specification $\mathcal{A}$ as a PA. The question whether at least one computation of a Kripke structure satisfies the specification (which we call existential safety model checking and boils down to solving intersection-emptiness) has been studied in [3] and is motivated by the detection of faulty computations. The authors of [3] show that this problem is coNP-complete for (non-deterministic) reset PA and for all models that are weaker. Similarly,

	$L = \varnothing$?	$uv^\omega \in L$?	$L = \Sigma^\omega$?
det. limit PA	coNP-complete	NP-complete	decidable, Π_2^P-hard
det. reachability-regular PA	coNP-complete	NP-complete	undecidable
det. weak reset PA	coNP-complete	NP-complete	undecidable
det. strong reset PA	coNP-complete	NP-complete	Π_2^P-complete

Table 2: Decision problems.

the question whether all computations of a PA satisfy the specification (which we call universal PA model checking and boils down to solving inclusion) has been studied in [5]. The authors of [5] show that this problem is coNP-complete for deterministic safety and co-Büchi PA, and undecidable for their non-deterministic counterparts as well as for deterministic reachability and Büchi PA. We study this problem as well as the universal safety model checking problem and the existential PA model checking problem for the remaining deterministic models. Again, deterministic limit PA shine as all these problems are decidable for them. For the other models, some problems are decidable and some are not.

In the following table we list our results. Some of the questions have already been studied in the literature or follow immediately from the results of [3, 5]. We fill the missing gaps and remark that these results are the technically most demanding results of this paper.

	Model Checking			
	Kripke		PA	
	$\exists$	$\forall$	$\exists$	$\forall$
det. limit PA	coNP-c. [3]	dec., Π_2^P-hard	coNP-c.	dec., Π_2^P-hard
det. reachability-regular PA	coNP-c. [3]	undec.	coNP-c.	undec.
det. weak reset PA	coNP-c. [3]	undec.	undec.	undec.
det. strong reset PA	coNP-c. [3]	Π_2^P-c.	undec.	Π_2^P-c.
det. reachability PA	coNP-c. [3]	undec. [5]	coNP-c. [5]	undec. [5]
det. Büchi PA	coNP-c. [3]	undec. [5]	coNP-c.	undec. [5]
det. safety PA	undec. [5]	coNP-c. [5]	undec. [5]	coNP-c. [5]
det. co-Büchi PA	undec. [5]	coNP-c. [5]	undec. [5]	coNP-c. [5]

Table 3: (Un)decidability results of model checking problems.

We remark that the Π_2^P-hardness results we obtain for deterministic limit PA are not tight, that is, it remains open whether these problems can be solved in Π_2^P. However, we are able to show that if we have the guarantee that the semi-linear sets of the PA can be complemented in polynomial time, then the Π_2^P-hard problems for det. limit PA and det. strong reset PA become coNP-complete. We also remark that the coNP-completeness result for deterministic Büchi PA does not follow from [5].

References

[1] E. M. CLARKE, T. A. HENZINGER, H. VEITH, R. BLOEM, *Handbook of Model Checking*. 1st edition, Springer Publishing Company, Incorporated, 2018.

[2] E. GRÄDEL, W. THOMAS, T. WILKE (eds.), *Automata logics, and infinite games: a guide to current research*. Springer-Verlag, Berlin, Heidelberg, 2002.

[3] M. GROBLER, L. SABELLEK, S. SIEBERTZ, Remarks on Parikh-Recognizable Omega-languages. In: A. MURANO, A. SILVA (eds.), *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*. Leibniz International Proceedings in Informatics (LIPIcs) 288, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2024, 31:1–31:21.

[4] M. GROBLER, S. SIEBERTZ, Deterministic Parikh automata on infinite words. 2024. `https://arxiv.org/abs/2401.14737`

[5] S. GUHA, I. JECKER, K. LEHTINEN, M. ZIMMERMANN, Parikh Automata over Infinite Words. In: *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022)*. Leibniz International Proceedings in Informatics (LIPIcs) 250, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 40:1–40:20.

[6] L. HERRMANN, V. PETH, S. RUDOLPH, Decidable (Ac)counting with Parikh and Muller: Adding Presburger Arithmetic to Monadic Second-Order Logic over Tree-Interpretable Structures. In: *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*. Leibniz International Proceedings in Informatics (LIPIcs) 288, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2024, 33:1–33:19.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 51–54.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

The Pumping Lemma for Context-Free Languages is Undecidable

Hermann Gruber[(A)] Markus Holzer[(B)] Christian Rauch[(B)]

[(A)]Planerio GmbH, Theresienhöhe 11A, 80538 München
h.gruber@planerio.de

[(B)]Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
{holzer,christian.rauch}@informatik.uni-giessen.de

Since the beginning of automata and formal language theory, researchers have studied pumping and iteration properties of formal languages to gain better insights into the computational complexity and expressive power of various types of language accepting or generating mechanisms. It is well known that not all formal language families obey pumping properties as, e.g., context-sensitive or Type-0 languages. Hence, satisfying a particular pumping property gives certain information about the structure of the language family, and is very often used to show that a particular language does not belong to the language family in question. For instance, the famous Bar-Hillel pumping lemma [3, page 154, Theorem 4.1] reads as follows—observe that the original Bar-Hillel pumping lemma uses two pumping constants, one for the length of the word and the other for the sub-word that can be pumped.

Lemma 1 *Let L be a context-free language over Σ. Then, there is a constant p (depending on L) such that the following holds: If $z \in L$ and $|z| \geq p$, then there are words $u, v, w, x, y \in \Sigma^*$ such that $z = uvwxy$, $|vx| \geq 1$, $|vwx| \leq p$, and $uv^twx^ty \in L$ for $t \geq 0$—it is then said that v and x can be (simultaneously)* pumped *in z.*

Bar-Hillel's lemma applied to the language $L = \{\, a^n b^n c^n \mid n \geq 0 \,\}$ shows that this language is *not* context free. In fact, the literature on pumping properties is far-reaching, with very different applications, see, e.g., [6], where language families are defined *via* pumping properties, [11] with a focus on pumping with the additional requirement that repeating a sub-word is allowed only if it is done a minimal number of times, or [8], which investigates regular pumping of Turing machine languages and learnability, just to mention a few.

We continue our research on the PUMPING-PROBLEM initiated in [4]. This is the problem of deciding for an automaton A (or a grammar G, respectively) and a value p, whether the language $L(A)$ (the set $L(G)$, respectively) satisfies a previously fixed pumping lemma w.r.t. the value p. For finite automata and Kozen's [9] and Jaffe's [7] pumping lemmata for regular languages, it has been shown that this problem is already intractable for DFAs, namely coNP-complete—this is the case for both pumping lemmata. This is quite remarkable, as it is a rare example of a finite automaton problem where the studied property becomes intractable for a single deterministic device. Jaffe's pumping property turns out to be more complex for NFAs, namely

This is a summary of a paper presented at the 28th International Conference on Developments in Language Theory (DLT) held in Göttingen, Germany, August 12–16, 2024.

PSPACE-complete, while for Kozen's lemma it is shown to be coNP-hard and contained in Π_2^{P} for nondeterministic finite state devices. Furthermore, analysis of these problems has led to the conclusion that they are inapproximable unless the Exponential Time Hypothesis (ETH) fails. Since regular languages also satisfy context-free pumping lemmata, e.g., Bar-Hillel's pumping lemma, the natural question is how complicated it is to decide whether a regular or a context-free language satisfies a given context-free pumping lemma w.r.t. the pumping parameter involved? This is the starting point of the current paper.

Here we study the complexity of the PUMPING-PROBLEM for regular, linear context-free, and context-free languages w.r.t. Bar-Hillel's pumping lemma [3] and its variants for regular [10] and linear context-free language [5]. Before investigating these problems in detail, we present some basic properties of the minimal pumping constants w.r.t. these pumping lemmata. For instance, we show that for a context-free language L generated by a context-free grammar G with n nonterminals, and where m being the maximum length of the right-hand-side of the productions in G, with $m \geq 2$, there is a constant p fulfilling Lemma 1 w.r.t. L such that $p \leq m^{2n+3}$. It is shown that in almost all cases the PUMPING-PROBLEM is undecidable; for context-free and linear context-free grammars and pumping lemma the statement reads as follows:

Theorem 1 *Given a context-free grammar G and a natural number p, it is undecidable whether for the language $L(G)$ the statement of Lemma 1 for the value p holds. The statement remains valid if a linear context-free grammar and the pumping lemma for linear-context-free languages is considered instead.*

Except when we consider regular languages represented by finite automata or right-linear grammars, the problem becomes decidable.

Theorem 2 *Given a finite automaton A and a natural number p, it is decidable whether for the language $L(A)$ the statement of the pumping lemma for regular languages, linear context-free languages, and context-free languages holds for the value p.*

A more detailed analysis shows that the PUMPING-PROBLEM for context-free languages w.r.t. Bar Hillel's pumping lemma is complete for the Π_1^0-level of the arithmetical hierarchy. Observe that every language in Π_1^0 is the complement of a recursively enumerable language. A summary of the results obtained is given in Table 1.

Language family	Pumping w.r.t. value p		
	regular	linear context-free	context-free
REG	decidable		
LIN	undecidable (Π_1^0)		
CFL			

Table 1: Decidability status of the PUMPING-PROBLEM for different language families and pumping lemmata.

It is worth mentioning that the problem remains undecidable even if regular pumping is considered. On the other hand, context-free pumping becomes decidable if the underlying language family is regular. This result relies heavily on the congruence representation of regular languages. Whether this result can be extended to other language families that allow for some congruence representation, such as the family of visibly pushdown languages [1, 2], is subject of further research. Furthermore, for the decidable variants of the problem, it remains to consider their computational complexity, which would nicely complement our previous investigations mentioned above.

Literatur

[1] Alur, R., Kumar, V., Madhusudan, P., Viswanathan, M.: Congruences for visibly pushdown languages. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) Proceedings of the 32nd International Colloquim Automata, Languages and Programming. pp. 1102–1114. No. 3580 in LNCS, Springer, Lisbon, Portugal (2005)

[2] Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM **56**(3), Art. 16 (2009)

[3] Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung **14**, 143–177 (1961)

[4] Gruber, H., Holzer, M., Rauch, C.: The pumping lemma for regular languages is hard. In: Nagy, B. (ed.) Proceedings of the 27th International Conference on Implementation and Application of Automata. pp. 128–140. No. 14151 in LNCS, Springer, Famagusta, Cyprus (2023).

[5] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)

[6] Horváth, S.: The family of languages satisfying Bar-Hillel's lemma. RAIRO–Informatique théorique et Applications / Theoretical Informatics and Applications **12**(3), 193–199 (1978)

[7] Jaffe, J.: A necessary and sufficient pumping lemma for regular languages. SIGACT News **10**(2), 48–49 (Sommer 1978).

[8] Kalociński, D.: On computability and learnability of the pumping lemma function. In: Dediu, A.H., Martín-Vide, C., Sierra-Rodríguez, J.L., Truthe, B. (eds.) Proceedings of the 8th International Conference Language and Automata Theory and Applications. pp. 433–440. No. 8370 in LNCS, Springer, Madrid, Spain (2014)

[9] Kozen, D.C.: Automata and Computability. Undergraduate Texts in Computer Science, Springer (1997).

[10] Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. **3**, 114–125 (1959).

[11] Sommerhalder, R.: Classes of languages proof against regular pumping. RAIRO–Informatique théorique et Applications / Theoretical Informatics and Applications **14**(2), 169–18 (1980)

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 55–58.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Verschiedene Arten von kometartigen Sprachen und ihre Verwendung bei kontextualen Grammatiken

Marvin Ködding Bianca Truthe

Institut für Informatik, Universität Gießen
Arndtstr. 2, 35392 Gießen, Germany
{marvin.koedding,bianca.truthe}@informatik.uni-giessen.de

Zusammenfassung

In diesem Beitrag stellen wir Forschungsergebnisse zur Erzeugungskraft von kontextualen Grammatiken mit Auswahl aus subregulären Sprachfamilien vor. Wir untersuchen verschiedene Klassen kometartiger Sprachen und vergleichen sie mit den bisher untersuchten Familien, um die akutelle Hierarchie zu erweitern. Darauf aufbauend erweitern wir die Hierarchie von Sprachen, die durch kontextuale Grammatiken erzeugt werden.

1. Einleitung

Kontextuale Grammatiken wurden von SOLOMON MARCUS in der Veröffentlichung [3] eingeführt, wobei die externen kontextualen Grammatiken die einfachste Form von kontextualen Grammatiken bilden. Hierbei wird ein Wort x mit einem Kontext (u, v) versehen und damit zum Wort uxv abgeleitet. Um den Ableitungsprozess zu steuern, wurden kontextuale Grammatiken mit einem Auswahlmechanismus ausgestattet, wobei ein Kontext (u, v) nur noch dann um ein Wort x gesetzt werden darf, wenn x zu einer assoziierten Sprache (genannt Auswahlsprache) gehört.

Es wurde untersucht, inwiefern der Ableitungsprozess von kontextualen Grammatiken durch Sprachen gesteuert werden kann, die zu speziellen Sprachfamilien gehören. Sogenannte subreguläre Sprachfamilien wurden dafür zuerst von JÜRGEN DASSOW in [1] betrachtet.

In diesem Beitrag erweitern wir die Hierarchie von subregulären Sprachfamilien um einige Familien kometartiger Sprachen. Außerdem untersuchen wir die Erzeugungskraft von externen kontextualen Grammatiken mit Auswahl in solchen subregulären Sprachfamilien.

2. Definitionen

Eine Menge an Wörtern über einem Alphabet V nennen wir *Sprache* über dem Alphabet V. Es sei V ein Alphabet, dann bezeichnet V^* die Menge aller Wörter über dem Alphabet V.

Wir definieren die folgenden Einschränkungen für reguläre Sprachen. In Klammern geben wir an, wie wir die jeweilige Familie aller dieser Sprachen bezeichnen.

Dies ist eine Kurzform unseres Beitrags auf der Konferenz NCMA 2024 ([2]).

Es sei L eine Sprache über einem Alphabet V. Bezüglich des Alphabets V nennen wir die Sprache L

- **kombinational** (*COMB*), wenn sie in der Form $L = V^*A$ mit $A \subseteq V$ dargestellbar ist,
- **definit** (*DEF*), wenn sie in der Form $L = A \cup V^*B$ mit endlichen Teilmengen A und B von V^* dargestellbar ist,
- **symmetrisch definit** (*SYDEF*), wenn $L = EV^*H$ für zwei reguläre Sprachen E und H über V gilt,
- **nilpotent** (*NIL*), wenn die Sprache L oder ihr Komplement $V^* \setminus L$ endlich ist,
- **kommutativ** (*COMM*), wenn zu jedem Wort auch alle seine Permutationen in der Sprache liegen,
- **zirkulär** (*CIRC*), wenn zu jedem Wort auch alle seine zirkulären Verschiebungen zur Sprache gehöhren,
- **abgeschlossen unter Suffixbildung** (*SUF*), wenn für je zwei Wörter x und y über V gilt, dass $xy \in L$ impliziert $y \in L$ (zu jedem Wort gehören auch alle seine Suffixe zur Sprache L),
- **nicht-zählend** (*NC*), wenn es eine Zahl $k \in \mathbb{N}$ so gibt, dass für je drei Wörter x, y und z über V gilt, dass das Wort xy^kz genau dann zur Sprache L gehört, wenn das Wort $xy^{k+1}z$ dazu gehört,
- kleenesternfrei (*SF*), wenn die Sprache L durch einen regulären Ausdruck beschrieben wird, der als Operationen nur Konkatenation, Vereinigung und Komplementbildung enthält,
- **vereinigungsfrei** (*UF*), wenn die Sprache L durch einen regulären Ausdruck ohne Vereinigungsoperator beschrieben wird,
- **endlich** (*FIN*), wenn die Mächtigkeit der Sprache L eine natürliche Zahl ist,
- **monoidal** (*MON*), falls $L = V^*$ gilt,
- **potenzseparierend** (*PS*), falls es eine natürliche Zahl $m \in \mathbb{N}$ so gibt, dass für jedes Wort $x \in V^*$ entweder $J_x^m \cap L = \emptyset$ oder $J_x^m \subseteq L$ gilt, wobei $J_x^m = \{x^n \mid n \geq m\}$ gilt,
- **geordnet** (*ORD*), falls es einen deterministischen endlichen Automaten gibt, der die Sprache L akzeptiert und zu dem eine totale Ordnung $(Z, \prec)$ auf der Zustandmenge Z existiert, die unter der Überführungsfunktion erhalten bleibt (zu jedem Buchstaben $a \in V$ und je zwei Zuständen $z, z' \in Z$ impliziert $z \preceq z'$ die Beziehung $\delta(z,a) \preceq \delta(z',a)$),
- **Stern** (*STAR*), falls $L = H^*$ für eine reguläre Sprache $H \subseteq V^*$ gilt,
- **linksseitiger Komet, rechtsseitiger Komet** (*LCOM*, *RCOM*), falls $L = HG^*$ (linksseitig) oder $L = G^*H$ (rechtsseitig) gilt, wobei $G^* \in$ *STAR*, aber $G \notin \{\emptyset, \{\lambda\}\}$ gilt und $H \subseteq V^*$ eine reguläre Sprache ist,
- **zweiseitiger Komet** (*2COM*), falls $L = EG^*H$ gilt, wobei $G^* \in$ *STAR*, aber $G \notin \{\emptyset, \{\lambda\}\}$ gilt sowie E und H reguläre Sprachen über V sind.

In manchen dieser Sprachfamilien liegen auch nicht-reguläre Sprachen. Wir betrachten hier aber nur solche, die regulär sind.

Es sei $\mathcal{F}$ eine Familie von Sprachen. Eine kontextuale Grammatik mit einer Auswahl aus $\mathcal{F}$ ist ein Tripel $G = (V, \mathcal{S}, A)$, wobei

- V ein Alphabet ist,
- $\mathcal{S}$ eine endliche Menge von Paaren (S, C) ist, wobei S eine Auswahlsprache über einem Teilalphabet $U \subseteq V$ ist, die zur Familie $\mathcal{F}$ gehört, und $C \subset V^* \times V^*$ eine endliche Menge

von Kontexten ist, in der bei jedem Paar $(u, v) \in C$ mindestens eine Komponente nicht leer ist: $uv \neq \lambda$,

– A eine endliche Teilmenge von V^* ist (deren Elemente Axiome heißen).

Ein direkter externer Ableitungsschritt mittels einer kontextualen Grammatik G wird wie folgt definiert: Ein Wort x wird zu einem Wort y abgeleitet (geschrieben $x \Longrightarrow y$), falls es ein Paar $(S, C) \in \mathcal{S}$ so gibt, dass $x \in S$ und $y = uxv$ mit einem Kontext $(u, v) \in C$ gilt.

Wir bezeichnen mit $\Longrightarrow^*$ den reflexiven und transitiven Abschluss der Relation $\Longrightarrow$. Die Sprache, die von G erzeugt wird, ist $L = \{\, z \mid x \Longrightarrow^* z \text{ für } x \in A \,\}$.

Zu einer Sprachfamilie $\mathcal{F}$ bezeichnen wir mit $\mathcal{EC}(\mathcal{F})$ die Familie aller Sprachen, die extern durch kontextuale Grammatiken mit Auswahl aus $\mathcal{F}$ erzeugt werden.

3. Ergebnisse

In der Literatur wird oft definiert, dass zwei Sprachen äquivalent sind, wenn sie sich nur im Leerwort unterscheiden. Analog werden zwei Sprachfamilien als äquivalent definiert, wenn sie sich nur in den Sprachen $\emptyset$ oder $\{\lambda\}$ unterscheiden. Nach dieser Definition gilt, dass *STAR* eine echte Teilmenge der Sprachfamilien *LCOM*, *RCOM* und *2COM* ist.

Wir betrachten aber die Mengen *STAR* und *STAR* $\setminus \{\{\lambda\}\}$ als unterschiedlich. Daraus folgt, dass die Sprachfamilie *STAR* unvergleichbar zu den Familien der kometartigen Sprachen ist.

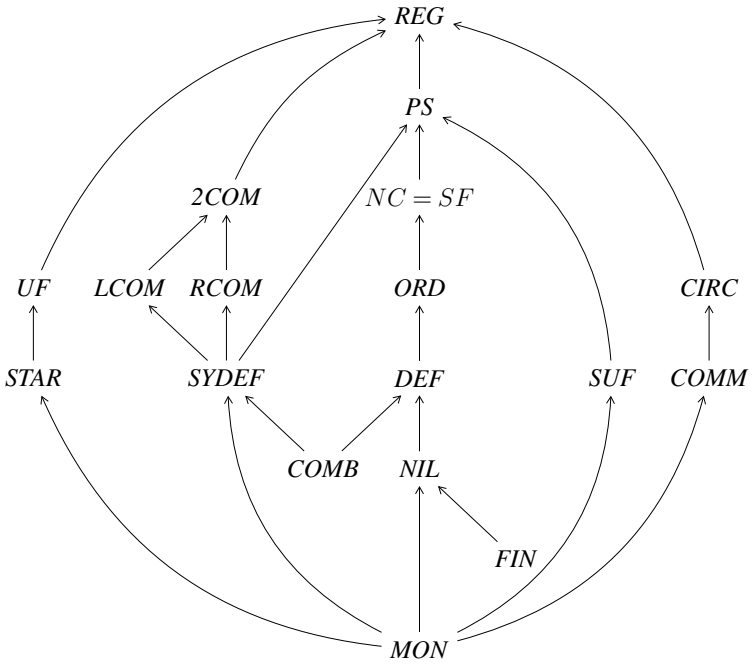


Abbildung 1: Hierarchie von subregulären Sprachfamilien

Satz 3.1 *Es gelten die Teilmengenbeziehungen aus Abbildung 1. Ein Pfeil von einem Knoten X zu einem Knoten Y steht für eine echte Teilmengenbeziehung $X \subset Y$. Falls zwei Familien nicht durch einen gerichteten Pfad verbunden sind, so sind sie unvergleichbar.*

Hinsichtlich der Familien von Sprachen, die von kontextualen Grammatiken erzeugt werden, ergibt sich die folgende Hierarchie.

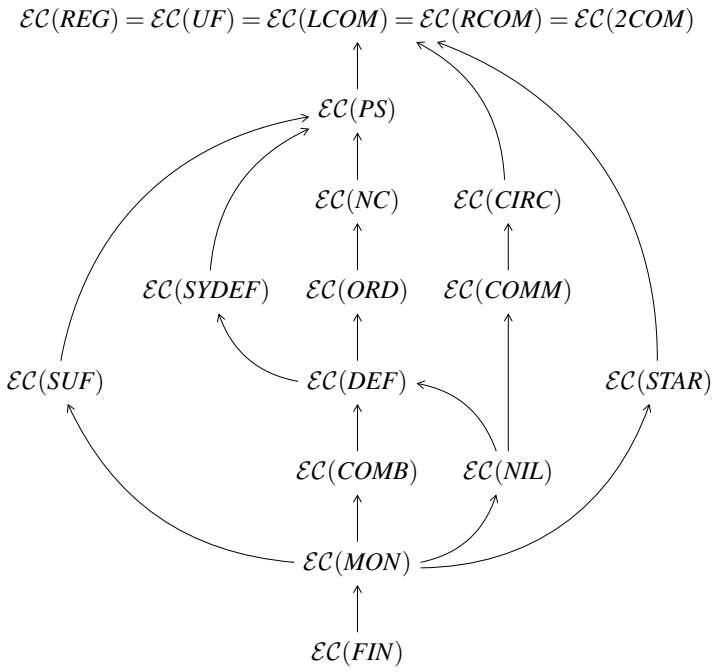


Abbildung 2: Hierarchie von Sprachfamilien als Auswahl für externe kontextuale Grammatiken

Satz 3.2 *Es gelten die Teilmengenbeziehungen aus Abbildung 2. Ein Pfeil von einem Knoten X zu einem Knoten Y steht für eine echte Teilmengenbeziehung $X \subset Y$. Falls zwei Familien nicht verbunden sind, so sind diese unvergleichbar.*

Literatur

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

[2] M. KÖDDING, B. TRUTHE, Various Types of Comet Languages and their Application in External Contextual Grammars. In: G. PIGHIZZINI, F. MANEA (eds.), *Proceedings of the 14th International Workshop on Non-Classical Models of Automata and Applications, NCMA 2024, Göttingen, Germany, 12th–13th August, 2024*. EPTCS 407, 2024, 120–137.

[3] S. MARCUS, Contextual grammars. *Revue Roumaine de Mathématique Pures et Appliquées* **14** (1969), 1525–1534.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 59–59.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

An Application of Matrix Semigroup Problems to Word Equations With Length Constraints

Matthew Konefal[(X)]

[(X)]Loughborough University
m.konefal2@lboro.ac.uk

Abstract

The solubility of word equations is decidable by a classical result of Makanin. There is motivation - both intrinsic and extrinsic - to investigate more general theories. One of the most studied is $WE + LEN$, which allows in its formulas both word equations and linear inequalities over the lengths of the equations' string variables. These 'word equations with length constraints' have a solubility problem with open decidability status. We show that the solubility of a single quadratic word equation in ≤ 2 variables and with arbtirary length constraints is decidable. The nontrivial case is that in which the equation is 'skeleton-equivalent' to $E := XY \doteq YX$ (i.e., it differs from E only in its terminal symbols). In this case, the decidability follows from results relating to reachability in the matrix semigroup $\mathrm{GL}(2, \mathbb{Z})$. The key insight is to treat the graph generated by the 'Neilsen transformations' algorithm as a finite automaton whose edges are labelled with particular matrices.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 60–61.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

KoAT: An Automatic Complexity and Termination Analysis Tool for Integer Programs

Nils Lommen Éléanore Meyer Jürgen Giesl

RWTH Aachen University
{lommen,eleanore.meyer,giesl}@cs.rwth-aachen.de

Abstract

KoAT is a tool to automatically infer complexity bounds for (probabilistic) integer programs, based on an alternating modular inference of upper runtime and size bounds for program parts [1]. By inferring runtime bounds for individual subprograms repeatedly, in the end we obtain a bound on the runtime complexity of the whole program. Furthermore, KoAT can be used to automatically prove termination of programs.

To infer runtime bounds for subprograms, we use multiphase linear ranking functions (MΦRFs) [2]. In contrast to classical ranking functions, MΦRFs can also represent bounds on multiple "phases" of program executions. Moreover, we embedded a complete technique for inferring runtime bounds of so-called triangular weakly non-linear loops (twn-loops) in our complexity analysis tool [3]. The update of a twn-loop is triangular, i.e., we can order the program variables such that the update of any x_i does not depend on the variables $x_1, \ldots, x_{i-1}$ with smaller indices. So the restriction to triangular updates prohibits "cyclic dependencies" of variables. Due to this, one can compute closed forms for the repeated updates of twn-loops, which makes them especially suitable for automatic program analysis. In particular, this approach also allows us to analyze the complexity of programs with non-linear arithmetic. In addition, we developed a novel procedure to infer size bounds via closed forms as well, which is also suitable for programs with non-linear arithmetic [4].

We adapted KoAT's framework for automated complexity analysis to *probabilistic* integer programs in [6]. To improve the power of automatic complexity analysis further, we integrated control-flow refinement via partial evaluation into KoAT's approach [2] and recently adapted this refinement for probabilistic programs as well [5].

KoAT's source code, a web interface, a binary, and a Docker image are available at

`https://koat.verify.rwth-aachen.de`.

References

[1] M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, J. Giesl, Analyzing Runtime and Size Complexity of Integer Programs. *ACM Trans. Program. Lang. Syst.* **38** (2016), 13:1–13:50.

[2] J. Giesl, N. Lommen, M. Hark, F. Meyer, Improving Automatic Complexity Analysis of Integer Programs. In: *The Logic of Software. A Tasting Menu of Formal Methods*. LNCS 13360, 2022, 193–228.

[3] N. LOMMEN, F. MEYER, J. GIESL, Automatic Complexity Analysis of Integer Programs via Triangular Weakly Non-Linear Loops. In: *Proc. IJCAR '22*. LNCS 13385, 2022, 734–754.

[4] N. LOMMEN, J. GIESL, Targeting Completeness: Using Closed Forms for Size Bounds of Integer Programs. In: *Proc. FroCoS '23*. LNCS 14279, 2023, 3–22.

[5] N. LOMMEN, E. MEYER, J. GIESL, Control-Flow Refinement for Complexity Analysis of Probabilistic Programs in KoAT (Short Paper). In: *Proc. IJCAR '24*. LNCS 14739, 2024, 233–243.

[6] F. MEYER, M. HARK, J. GIESL, Inferring Expected Runtimes of Probabilistic Integer Programs Using Expected Sizes. In: *Proc. TACAS '21*. LNCS 12651, 2021, 250–269.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 62–62.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

The Weighted HOM-Problem over Nonnegative Integers

Andreas Maletti[(A)] Andreea-Teodora Nász[(A)] Erik Paul[(A)]

[(A)]Universität Leipzig
Department of Mathematics and Computer Science
{maletti,nasz,epaul}@informatik.uni-leipzig.de

Abstract

The (weighted) HOM problem asks, given a regular (weighted) tree language and a tree homomorphism as input, whether the image is again regular. In 2010, the boolean variant of this problem was shown to be decidable [1]. Research on the weighted versions of this question, however, has been scarce and facing limitations. Recently, the group of automata theorists of the Leipzig University has proved that the N-weighted HOM problem is decidable in nondeterministic logarithmic space [2]. In this proof, the autors used a novel argument based on linear algebra. The presentation gives an intuitive introduction to the formal decision problem in question, and illustrates the idea of the proof on a user-friendly example.

References

[1] G. GODOY, O. GIMÉNEZ, L. RAMOS, C. ÀLVAREZ, The HOM problem is decidable. In: *Proc. 42nd ACM symp. Theory of Computing*. ACM, 2010, 485–494.

[2] A. MALETTI, A.-T. NÁSZ, E. PAUL, Weighted HOM-Problem for Nonnegative Integers. In: *41st International Symposium on Theoretical Aspects of Computer Science*. 2024.

THEORIE-AG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 63–63.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Algorithms for the Enumeration of Minimal and Shortest Absent Subsequences of a String

Florin Manea[(A)] Tina Ringleb[(A)] Stefan Siemer[(A)]
Maximilian Winkler[(A)]

[(A)]University of Göttingen
{florin.manea,stefan.siemer}@cs.uni-goettingen.de
{tina.ringleb,maximilian.winkler01}@stud.uni-goettingen.de

Abstract

In this work, we present a series of algorithmic results regarding absent subsequences in strings. Given a string w, another string v is an *absent subsequence* of w if v does not occur as a subsequence of w. Following [3, 4, 1], for a string w, we are interested in its *shortest absent subsequences*, i.e., the shortest strings which are absent subsequences of w, and its *minimal absent subsequences*, i.e., those strings which are absent subsequences of w but whose every subsequence occurs in w. We provide an efficient algorithm for identifying the longest minimal absent subsequence of a string. Further, we present optimal algorithms (with linear time preprocessing and output-linear delay) for the enumeration of the shortest and, respectively, minimal absent subsequences. We also discuss how we can incrementally enumerate these strings with constant delay, in the setting defined by [2].

References

[1] D. ADAMSON, P. FLEISCHMANN, A. HUCH, M. WIEDENHÖFT, Tight Bounds for the Number of Absent Scattered Factors. *CoRR* **abs/2407.18599** (2024).
https://doi.org/10.48550/arXiv.2407.18599

[2] D. ADAMSON, P. GAWRYCHOWSKI, F. MANEA, Enumerating m-Length Walks in Directed Graphs with Constant Delay. In: J. A. SOTO, A. WIESE (eds.), *LATIN 2024: Theoretical Informatics - 16th Latin American Symposium, Puerto Varas, Chile, March 18-22, 2024, Proceedings, Part I*. Lecture Notes in Computer Science 14578, Springer, 2024, 35–50.
https://doi.org/10.1007/978-3-031-55598-5_3

[3] M. KOSCHE, T. KOSS, F. MANEA, S. SIEMER, Absent Subsequences in Words. In: P. C. BELL, P. TOTZKE, I. POTAPOV (eds.), *Reachability Problems - 15th International Conference, RP 2021, Liverpool, UK, October 25-27, 2021, Proceedings*. LNCS 13035, 2021, 115–131.

[4] M. KOSCHE, T. KOSS, F. MANEA, S. SIEMER, Absent Subsequences in Words. *Fundam. Informaticae* **189** (2022) 3-4, 199–240.
https://doi.org/10.3233/FI-222159

THEORIE-AG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 64–64.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Ternary is Still Good for Parikh Matrices

Robert Mercaş[(A)] Wen Chean Teh[(B)]

[(A)]Department of Computer Science, Loughborough University, UK,
R.G.Mercas@lboro.ac.uk

[(B)]School of Mathematical Sciences, Universiti Sains Malaysia, Malaysia,
dasmenteh@usm.my

Abstract

The focus of this presentation are Parikh matrices with emphasis on two concrete problems. In the first part of our presentation we show that a conjecture by Dick at al. in 2021 only stands in the case of ternary alphabets, while providing counterexamples for larger alphabets. In particular, we show that the only type of distinguishability in the case of 3-letter alphabets is the trivial one. The second part of the paper builds on the notion of Parikh matrices for projections of words, discussed initially in this work, and answers, once more in the case of a ternary alphabet, a question posed by Atanasiu et al. in 2022 with regards to the minimal Hamming distance in between words sharing a congruency class.

Main results

In this work we give an in-depth analysis of the structure that the projections' words of amiable sequences exhibit. To this end, we show that for ternary alphabets amiable words are distinguishable via downwards projections if and only if such projections are trivially deducible, i.e., the considered words contain length two factors that consist of letters not consecutive in the ordered alphabet. Buiding on the notion of distinguishability as an outcome of projections from ternary alphabets, we are also able to show that, within every M-equivalence class of 3-letter words at least two words have their Hamming distance as one of the values $\{2,4,7,8\}$.

This presentation is based on unpublished results from [1].

References

[1] R. MERCAŞ, W. C. TEH, Ternary is Still Good for Parikh Matrices. *Theoretical Computer Science* (2024). Submitted.

[(A)]This work has been done under a Return Fellowship from the Alexander von Humboldt Foundation.

THEORIE-TAG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 65–70.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

The Equivalence Problem of E-Pattern Languages with Regular Constraints is Undecidable

Dirk Nowotka[(A)] Max Wiedenhöft[(A)]

[(A)]Kiel University, Department of Computer Science
{dn,maw}@informatik.uni-kiel.de

Abstract

Patterns are words with terminals and variables. The language of a pattern is the set of words obtained by uniformly substituting all variables with words that contain only terminals. Regular constraints restrict valid substitutions of variables by associating with each variable a regular language representable by, e.g., finite automata. Pattern languages with regular constraints contain only words in which each variable is substituted according to a set of regular constraints. We consider the membership, inclusion, and equivalence problems for erasing and non-erasing pattern languages with regular constraints. Our main result shows that the erasing equivalence problem—one of the most prominent open problems in the realm of patterns—becomes undecidable if regular constraints are allowed in addition to variable equality.

1. Introduction

A *pattern* is a finite word consisting of symbols from a finite set of letters $\Sigma = \{a_1, ..., a_\sigma\}$, also called terminals, and from an infinite set of variables $X = \{x_1, x_2, ...\}$ with $\Sigma \cap X = \emptyset$. It is a natural and compact device to define formal languages. Words consisting of only terminal symbols are obtained from patterns by a *substitution* h, a terminal preserving morphism which maps all variables from a pattern to words over the terminal alphabet. The *language* of a pattern consists of all words obtainable from that pattern by substitutions. We differentiate between two kinds of substitutions. Originally, pattern languages introduced by Angluin [1] only consisted of words obtained by *non-erasing substitutions* that required all variables to be mapped to non-empty words. Thus, those languages are also called *NE-pattern languages*. Later, so called *erasing-/extended-* or just *E-pattern languages* have been introduced by Shinohara [25]. In these, substitutions are also allowed to map variables to the empty word. Consider, for example, the pattern $\alpha := x_1\mathtt{ab}x_2x_2$. Then, by mapping x_1 to $\mathtt{aaa}$ and x_2 to $\mathtt{ba}$ with a substitution h, we obtain the word $h(\alpha) = \mathtt{aaaabbaba}$. If we consider the E-pattern language of α, we could also map x_2 to the empty word ε with a substitution h' which also maps x_1 to $\mathtt{aaa}$ and obtain $h'(\alpha) = \mathtt{aaaab}$.

Due to its practical and simple definition, patterns and their corresponding languages occur in numerous areas regarding computer science and discrete mathematics, including unavoidable patterns [14, 17], algorithmic learning theory [1, 5, 26], word equations [17], theory of extended regular expressions with back references [9], and database theory [7, 24].

[(A)]This work was supported by the DFG project number 437493335.

The main problems regarding patterns and pattern languages are the *membership problem* (and its variations [6, 10, 11]), the *inclusion problem*, and the *equivalence problem* in both the erasing (E) and non-erasing (NE) cases. The membership problem determines if a word belongs to a pattern's language. This problem is NP-complete for both E- and NE-pattern languages [1, 14]. The inclusion problem asks if one pattern's language is included in another's. Jiang et al. [15] showed that it is generally undecidable for E- and NE-pattern languages. Freydenberger and Reidenbach [8], and Bremer and Freydenberger [2] proved its undecidability for all alphabets with size ≥ 2 in both E- and NE-pattern languages. The equivalence problem tests if two patterns generate the same language. It is trivially decidable for NE-pattern languages [1]. Whether its decidable for E-pattern languages is one of the major open problems in the field [15, 20, 21, 22, 23]. However, for terminal-free patterns, the inclusion and equivalence problems in E-pattern languages have been shown to be NP-complete [4, 15]. The decidability of the inclusion problem for terminal-free NE-pattern languages remains unresolved, though.

Various extensions to patterns and pattern languages have been introduced over time. Some examples are the bounded scope coincidence degree, patterns with bounded treewidth, k-local patterns, and strongly-nested patterns (see [3] and references therein). Koshiba [16] introduced so called *typed patterns* to enhance the expressiveness of pattern languages by restricting substitutions of variables to arbitrary recursive languages. This has been extended by Geilke and Zilles [12] who introduced the notion of *relational patterns* and *relational pattern languages*.

We consider a specific class of typed- or relational patterns called *patterns with regular constraints*. Let $\mathcal{L}_{Reg}$ be the set of all regular languages. Then, we say that a mapping $r : X \to \mathcal{L}_{Reg}$ is a *regular constraint* that implicitly defines *languages on variables* $x \in X$ by $L_r(x) = r(x)$. Let $\mathcal{C}_{Reg}$ be the *set of all regular constraints*. A *patterns with regular constraints* $(\alpha, r_\alpha) \in (\Sigma \cup X)^* \times \mathcal{C}_{Reg}$ is a pattern which is associated with a regular constraint. A substitution h is r_α-*valid* if all variables are substituted according to r_α. The (non-)erasing language $L_{(N)E}(\alpha, r_\alpha)$ of a pattern with regular constraints is defined analogously to (non-)erasing pattern languages $L_{(N)E}(\alpha)$ with the additional requirement that all substitutions must be r_α-valid.

The paper on which this talk is based examines the erasing (E) and non-erasing (NE) pattern languages with regular constraints. The membership problem for both is NP-complete, while the inclusion problem is undecidable for the general and terminal-free versions. This immediately follows from known results. The main result shows that the equivalence problem for erasing pattern languages with regular constraints is indeed undecidable.

2. Results on Pattern Languages with Regular Constraints

The membership problems of both, the erasing and non-erasing pattern languages, have been shown to be NP-complete [1, 14]. Hence, we observe the following for patterns with regular constraints.

Corollary 2.1 *Let (α, r_α) be a pattern with regular constraints and $w \in \Sigma^*$. The decision problem of whether $w \in L_X(\alpha, r_\alpha)$ for $X \in \{E, NE\}$ is NP-complete.*

Indeed, we immediately obtain NP-hardness in both cases by the previous results shown in [1, 14] for patterns. NP-containment follows by knowing that a valid certificate results in a substitution of α which has at most length $|w|$.

One other notable problem regarding patterns is the inclusion problem. The undecidabilities of the inclusion problems for patterns in the erasing and non-erasing cases have been initially shown by Jiang et al. [15] for unbounded alphabets and have been refined and extended to finite alphabets of sizes greater or equal to 2 in [2, 8]. Hence we have the following.

Theorem 2.2 *[2, 8, 15] Let α and β be two patterns. In general, for all alphabets Σ with $|\Sigma| \geq 2$, it is undecidable to answer whether*

1. $L_E(\alpha) \subseteq L_E(\beta)$, *or*
2. $L_{NE}(\alpha) \subseteq L_{NE}(\beta)$.

From that, we immediately obtain the following for patterns with regular constraints.

Corollary 2.3 *Let (α, r_α) and (β, r_β) be two patterns with regular constraints. In both, the terminal-free and the non terminal-free cases for α and β we have in general, for all alphabets Σ with $|\Sigma| \geq 2$, that it is undecidable to answer whether*

1. $L_E(\alpha, r_\alpha) \subseteq L_E(\beta, r_\beta)$, *or*
2. $L_{NE}(\alpha, r_\alpha) \subseteq L_{NE}(\beta, r_\beta)$.

Indeed, the general results follow immediately from Theorem 2.2. Additionally, in the terminal-free cases, we can reduce the general versions to the terminal free versions by substituting each terminal letter $\mathtt{a} \in \Sigma$ which occurs in a pattern α by a new variable $x_\mathtt{a}$ and setting $L(x_\mathtt{a}) = \{\mathtt{a}\}$. This results in effectively the same problem instances without using terminals in the pattern words.

The main result of the paper considers the equivalence problem for erasing pattern languages with regular constraints. In particular, it is shown that this problem is undecidable. This is done by a reduction of the problem whether a non-deterministic 2-counter automaton without input (see, e.g., [13, 18]) has some accepting computation to the problem of whether the erasing pattern languages of two patterns with regular constraints are equal. We obtain the following.

Theorem 2.4 *Let (α, r_α) and (β, r_β) be two patterns with regular constraints. In general, it is undecidable to decide whether $L_E(\alpha, r_\alpha) = L_E(\beta, r_\beta)$ for all alphabets Σ with $\Sigma \geq 2$.*

3. Further Discussion

The constructed patterns in the reduction for the result of Theorem 2.4 are both terminal-free. Hence, as the terminal-free patterns with regular constraints are a subset of all patterns with regular constraints, this result is directly shown for the general and terminal-free variant together. We mention the following fact.

Corollary 3.1 *Let (α, r_α) and (β, r_β) be two patterns with regular constraints such that $\alpha, \beta \in X^*$, i.e. α and β are terminal-free patterns. In general, it is undecidable to decide whether $L_E(\alpha, r_\alpha) = L_E(\beta, r_\beta)$ for all alphabets Σ with $\Sigma \geq 2$.*

With that, we obtain undecidability for nearly all problems regarding pattern languages with regular constraints. The only open case is the equivalence problem of non-erasing pattern languages with regular constraints. Using regular constraints, the problem becomes at least as hard as deciding the equivalence of two given regular languages witnessed by the following example.

Example 3.2 *Let (α, r_α) and (β, r_β) be two patterns with regular constraints such that $\alpha = x$ and $\beta = y$ for some $x, y \in X$. Then $L_{NE}(\alpha, r_\alpha) = L_{NE}(\beta, r_\beta)$ if and only if $L(x) \setminus \{\varepsilon\} = L(y) \setminus \{\varepsilon\}$.*

Despite the most prominent open problem for patterns being undecidable in the case of pattern languages with regular constraints, we see that even this problem, which is trivially decidable for patterns without regular constraints, becomes much harder in this setting. We propose the following open question to which we have no definite conjecture so far. An overview of the current state of patterns with regular constraints can be found in Table 1.

Question 4.3 *Given two patterns with regular constraints (α, r_α) and (β, r_β), is it generally decidable to answer whether $L_{NE}(\alpha, r_\alpha) = L_{NE}(\beta, r_\beta)$?*

Problem	General	Terminal-Free
E-Membership	NP-complete	NP-complete
E-Inclusion	Undecidable	Undecidable
E-Equivalence	Undecidable	Undecidable
NE-Membership	NP-complete	NP-complete
NE-Inclusion	Undecidable	Undecidable
NE-Equivalence	Open	Open

Table 1: Current state regarding pattern languages with regular constraints

The paper on which this talk is based is accepted at CIAA 2024 [19].

References

[1] D. ANGLUIN, Finding Patterns Common to a Set of Strings. *J. Comput. Syst. Sci.* **21** (1980) 1, 46–62.

[2] J. BREMER, D. D. FREYDENBERGER, Inclusion problems for patterns with a bounded number of variables. *Information and Computation* **220-221** (2012), 15–43.

[3] J. D. DAY, P. FLEISCHMANN, F. MANEA, D. NOWOTKA, Local Patterns. In: S. LOKAM, R. RAMANUJAM (eds.), *FSTTCS 2017*. LIPIcs 93, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2018, 24:1–24:14.

[4] A. EHRENFEUCHT, G. ROZENBERG, Finding a Homomorphism Between Two Words is NP-Complete. *Inf. Process. Lett.* **9** (1979) 2, 86–88.

[5] H. FERNAU, F. MANEA, R. MERCAŞ, M. L. SCHMID, Revisiting Shinohara's algorithm for computing descriptive patterns. *TCS* **733** (2018), 44–54. Special Issue on Learning Theory and Complexity.

[6] P. FLEISCHMANN, S. KIM, T. KOSS, F. MANEA, D. NOWOTKA, S. SIEMER, M. WIEDENHÖFT, Matching Patterns with Variables Under Simon's Congruence. In: O. BOURNEZ, E. FORMENTI, I. POTAPOV (eds.), *Reachability Problems*. Springer Nature Switzerland, Cham, 2023, 155–170.

[7] D. D. FREYDENBERGER, L. PETERFREUND, The Theory of Concatenation over Finite Models. In: N. BANSAL, E. MERELLI, J. WORRELL (eds.), *ICALP 2021, Proceedings*. LIPIcs 198, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 130:1–130:17.

[8] D. D. FREYDENBERGER, D. REIDENBACH, Bad news on decision problems for patterns. *Information and Computation* **208** (2010) 1, 83–96.

[9] D. D. FREYDENBERGER, M. L. SCHMID, Deterministic regular expressions with back-references. *Journal of Computer and System Sciences* **105** (2019), 1–39.

[10] P. GAWRYCHOWSKI, F. MANEA, S. SIEMER, Matching Patterns with Variables Under Hamming Distance. In: F. BONCHI, S. J. PUGLISI (eds.), *MFCS 2021*. Leibniz International Proceedings in Informatics (LIPIcs) 202, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021, 48:1–48:24.

[11] P. GAWRYCHOWSKI, F. MANEA, S. SIEMER, Matching Patterns with Variables Under Edit Distance. In: D. ARROYUELO, B. POBLETE (eds.), *String Processing and Information Retrieval*. Springer International Publishing, Cham, 2022, 275–289.

[12] M. GEILKE, S. ZILLES, Learning Relational Patterns. In: J. KIVINEN, C. SZEPESVÁRI, E. UKKONEN, T. ZEUGMANN (eds.), *Algorithmic Learning Theory*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, 84–98.

[13] O. H. IBARRA, Reversal-Bounded Multicounter Machines and Their Decision Problems. *J. ACM* **25** (1978) 1, 116–133.

[14] T. JIANG, E. KINBER, A. SALOMAA, K. SALOMAA, S. YU, Pattern languages with and without erasing. *International Journal of Computer Mathematics* **50** (1994) 3-4, 147–163.

[15] T. JIANG, A. SALOMAA, K. SALOMAA, S. YU, Decision Problems for Patterns. *Journal of Computer and System Sciences* **50** (1995) 1, 53–63.

[16] T. KOSHIBA, Typed pattern languages and their learnability. In: P. VITÁNYI (ed.), *Computational Learning Theory*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, 367–379.

[17] M. LOTHAIRE, *Combinatorics on Words*. 2 edition, Cambridge Mathematical Library, Cambridge University Press, 1997.

[18] M. L. MINSKY, Recursive Unsolvability of Post's Problem of "Tag" and other Topics in Theory of Turing Machines. *Annals of Mathematics* **74** (1961) 3, 437–455.

[19] D. NOWOTKA, M. WIEDENHÖFT, The Equivalence Problem of E-Pattern Languages with Regular Constraints is Undecidable. In: *CIAA 2024 (28th International Conference on Implementation and Application of Automata)*. LNCS, to appear, 2024.

[20] E. OHLEBUSCH, E. UKKONEN, On the equivalence problem for E-pattern languages. In: *MFCS 1996*. Springer Berlin Heidelberg, 1996, 457–468.

[21] D. REIDENBACH, On the Equivalence Problem for E-Pattern Languages Over Small Alphabets. In: *DLT*. Springer Berlin Heidelberg, 2004, 368–380.

[22] D. REIDENBACH, On the Learnability of E-pattern Languages over Small Alphabets. In: *Learning Theory*. Springer Berlin Heidelberg, 2004, 140–154.

[23] D. REIDENBACH, An examination of Ohlebusch and Ukkonen's conjecture on the equivalence problem for E-pattern languages. *J. Autom. Lang. Comb.* **12** (2007) 3, 407–426.

[24] 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*. ACM, 2022, 139–150.

[25] T. SHINOHARA, *Polynomial time inference of extended regular pattern languages*. Springer Berlin Heidelberg, 1983, 115–127.

[26] T. SHINOHARA, S. ARIKAWA, *Pattern inference*. Springer Berlin Heidelberg, 1995, 259–291.

THEORIE-AG 2024
T. Koß, F. Manea, S. Siemer (Eds.): Theorietag 2024,
Göttingen, 17. – 20.09.2024, Technical Report, pp. 71–74.
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Recent Logical Characterizations of GNNs and Their Implications for Formal Reasoning

Marco Sälzer[(A)]

[(A)]Theoretical Computer Science / Formal Methods, University of Kassel
marco.saelzer@uni-kassel.de

1. Introduction

Graph Neural Networks (GNNs) have emerged as a powerful tool in machine learning, particularly for analyzing and processing structured data in the form of graphs. Their effectiveness in various applications, such as social network analysis [10], molecular modeling [12, 9], and knowledge graph reasoning [13], has been widely recognized. However, as neural network-based models generally are increasingly applied in safety-critical domains, including autonomous driving [4] and healthcare [7], the need for robust verification methods to ensure the reliability and safety of GNN becomes also more important.

Previous research has identified significant challenges in the formal verification of GNNs, particularly by establishing undecidability results regarding the verification of essential safety properties [11]. Despite a growing body of work concerned with non-formal verification of GNN [6], there has been a notable lack of foundational results aside of [11] to guide the development of verification algorithms. A specific line of research has recently focused on logical characterizations of GNNs [1, 5, 8, 2]. In particular, the recent [8] shows that GNNs can be effectively translated into logical formulas within the $K^{\#}$ framework, which essentially is an extension of graded modal logic (GML) [3] by linear arithmetic. This development extends the foundational work by Barceló et al. [1], who first explored the logical expressiveness of certain GNNs in comparison to GML.

These advancements have significant implications for the verification of GNNs. By transforming GNNs into logical formulas, it becomes possible to leverage established logical reasoning methods to verify their behavior, providing a more rigorous foundation for ensuring the safety and reliability of GNN-based systems. In this talk, we discuss these implications.

2. A Recent Logical Characterization of GNN

We summarize the results of [8] in the following. In summary, the authors show that $K^{\#}$ exactly captures a subclass of *aggregation-combine graph neural networks* (GNNs). Thereby, it is assumed that a GNN $\mathcal{A}$ computes a node query, meaning $\mathcal{A} : \mathcal{G}_\bullet \to \{0, 1\}$ where $\mathcal{G}_\bullet$ denotes the class of all pairs of undirected, node-labelled graphs G and nodes v of G.

This talk mainly summarizes the results of [8] by P. Nunn, M. Sälzer, F. Schwarzentruber, N. Troquard, which was recently presented at IJCAI 24.

The first major result demonstrates the expressive equivalence between $K^\#$ and GNNs by showing that any formula in $K^\#$ can be effectively translated into a corresponding GNN. This translation is efficient, operating in polynomial time relative to the size of the formula.

Theorem 2.1 *[8] For every $K^\#$-formula φ, we can compute in polynomial time wrt. $|\varphi|$ a GNN $\mathcal{A}_\varphi$ such that $[[\varphi]] = [[\mathcal{A}_\varphi]]$.*

Conversely, the authors also establish that any GNN can be translated into a $K^\#$ formula. This result is significant because it shows that the behavior of any GNN can be captured and analyzed within the logical framework of $K^\#$. This translation is also computationally efficient, as it can be performed in polynomial time relative to the size of the GNN.

Theorem 2.2 *[8] For every GNN $\mathcal{A}$, we can compute in polynomial time wrt. $|\mathcal{A}|$ a $K^\#$-formula $\varphi_\mathcal{A}$, represented as a DAG, such that $[[\mathcal{A}]] = [[\varphi_\mathcal{A}]]$.*

In addition to this expressive equivalence, the authors further explore the complexity of reasoning within $K^\#$. Specifically, they analyze the complexity of the satisfiability problem for $K^\#$, determining its place within the complexity class **PSPACE**. This result is important for understanding the computational limits of reasoning about GNNs using $K^\#$.

Theorem 2.3 *[8] The satisfiability problem of the logic $K^\#$ is **PSPACE**-complete.*

It is worth noting that there is concurrent work [2] that independently establishes similar results regarding the relationship between GNNs and logic. The emergence of these results from multiple sources underscores the importance of logical characterizations in the analysis and verification of GNNs.

3. Implications for Formal Reasoning

Formal reasoning about Graph Neural Networks (GNNs) is a critical area of research, particularly in the context of verifying the behavior of these models in various applications. A central problem in this domain is determining whether a given GNN $\mathcal{A}$ can produce a specific output on any node of a graph from a class of interest. Specifically, the problem is to decide whether there exists a pointed graph $(G, v) \in \mathcal{G}_\bullet$ such that $\mathcal{A}(G, v) = 1$, where $\mathcal{G}_\bullet$ denotes the class of all undirected, node-labeled graphs and nodes. This problem is fundamental to understanding the expressive power and limitations of GNNs, as well as their potential failure in real-world applications.

A key result in this area, established in [11], provides a significant insight into the decidability of this problem under certain conditions:

Theorem 3.1 ([11]) *The problem, given GNN $\mathcal{A}$, of deciding whether there is a pointed graph $(G, v) \in \mathcal{G}_\bullet$ such that $\mathcal{A}(G, v) = 1$ is decidable if there is a uniform bound on the degree of graphs in $\mathcal{G}_\bullet$.*

The logical characterization provided by $K^\#$ (or the one of [2]) offers a powerful tool for extending the decidability result mentioned above. Specifically, the logical framework allows us to remove the restriction on the degree of the graphs in $\mathcal{G}_\bullet$ while ensuring that the problem

remains decidable, as long as we restrict to graphs with propositional labels. Moreover, the problem can be placed in the complexity class **PSPACE**.

Theorem 3.2 *Given a GNN $\mathcal{A}$, the problem of deciding whether there exists a pointed graph $(G, v) \in \mathcal{G}_\bullet$ such that $\mathcal{A}(G, v) = 1$ where all graphs of $\mathcal{G}_\bullet$ have purely propositional labels is decidable. Furthermore, the problem is **PSPACE**-complete.*

These findings extend our understanding of the formal reasoning capabilities available for GNNs. In the corresponding talk, we will delve deeper into these results, exploring their implications in greater detail and discussing possible further advancements in the logical characterization and verification of GNNs.

Literatur

[1] P. BARCELÓ, E. V. KOSTYLEV, M. MONET, J. PÉREZ, J. REUTTER, J.-P. SILVA, The Logical Expressiveness of Graph Neural Networks. In: *8th International Conference on Learning Representations (ICLR 2020)*. Virtual conference, Ethiopia, 2020.
https://hal.archives-ouvertes.fr/hal-03356968

[2] M. BENEDIKT, C. LU, B. MOTIK, T. TAN, Decidability of Graph Neural Networks via Logical Characterizations. In: K. BRINGMANN, M. GROHE, G. PUPPIS, O. SVENSSON (eds.), *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*. LIPIcs 297, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 127:1–127:20.
https://doi.org/10.4230/LIPIcs.ICALP.2024.127

[3] M. DE RIJKE, A Note on Graded Modal Logic. *Studia Logica: An International Journal for Symbolic Logic* **64** (2000) 2, 271–283.
http://www.jstor.org/stable/20016145

[4] S. M. GRIGORESCU, B. TRASNEA, T. T. COCIAS, G. MACESANU, A survey of deep learning techniques for autonomous driving. *J. Field Robotics* (2020).

[5] M. GROHE, The Logic of Graph Neural Networks. In: *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 2021, 1–17.
https://doi.org/10.1109/LICS52264.2021.9470677

[6] S. GÜNNEMANN, Graph Neural Networks: Adversarial Robustness. In: L. WU, P. CUI, J. PEI, L. ZHAO (eds.), *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Singapore, Singapore, 2022, 149–176.

[7] G. LITJENS, T. KOOI, B. E. BEJNORDI, A. A. A. SETIO, F. CIOMPI, M. GHAFOORIAN, J. A. W. M. VAN DER LAAK, B. VAN GINNEKEN, C. I. SÁNCHEZ, A survey on deep learning in medical image analysis. *Medical Image Anal.* (2017).

[8] P. NUNN, M. SÄLZER, F. SCHWARZENTRUBER, N. TROQUARD, A Logic for Reasoning About Aggregate-Combine Graph Neural Networks. *CoRR* **abs/2405.00205** (2024).
https://doi.org/10.48550/arXiv.2405.00205

[9] P. REISER, M. NEUBERT, A. EBERHARD, L. TORRESI, C. ZHOU, C. SHAO, H. METNI, C. VAN HOESEL, H. SCHOPMANS, T. SOMMER, P. REISER22GGNMATERIALCHEMISTRY FRIEDERICH, Graph neural networks for materials science and chemistry. *Communications Materials* **3** (2022) 93.

[10] A. SALAMAT, X. LUO, A. JAFARI, HeteroGraphRec: A heterogeneous graph-based neural networks for social recommendations. *Knowl. Based Syst.* **217** (2021), 106817. https://doi.org/10.1016/j.knosys.2021.106817

[11] M. SÄLZER, M. LANGE, Fundamental Limits in Formal Verification of Message-Passing Neural Networks. In: *The Eleventh International Conference on Learning Representations*. 2023. https://openreview.net/forum?id=WlbG820mRH-

[12] J. XIONG, Z. XIONG, K. CHEN, H. JIANG, M. ZHENG, Graph neural networks for automated de novo drug design. *Drug Discovery Today* **26** (2021) 6, 1382–1393. https://www.sciencedirect.com/science/article/pii/S1359644621000787

[13] Z. YE, Y. J. KUMAR, G. O. SING, F. SONG, J. WANG, A Comprehensive Survey of Graph Neural Networks for Knowledge Graphs. *IEEE Access* **10** (2022), 75729–75741.