I F I G
Research
Report

Institut für Informatik

35. Theorietag „Automaten und Formale Sprachen“

Schotten, 30. September bis 2. Oktober 2025

Bianca Truthe (Hrsg.)

IFIG Research Report 2501
September 2025

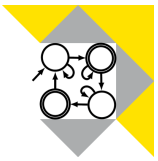
Institut für Informatik
JLU Gießen
Arndtstraße 2
35392 Gießen, Germany
Tel: +49-641-99-32141
Fax: +49-641-99-32149
mail@informatik.uni-giessen.de
www.informatik.uni-giessen.de

JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

IFIG Research Report

IFIG Research Report 2501, September 2025

35. Theorietag „Automaten und Formale Sprachen“



Schotten, 30. September bis 2. Oktober 2025

Tagungsband

Bianca Truthe (Hrsg.)

Institut für Informatik
Justus-Liebig-Universität Giessen
Arndtstr. 2
35392 Giessen

Inhaltsverzeichnis

Vorwort

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

Die bisherigen Theorietage wurden in Magdeburg (1991), in Kiel (1992), auf Schloß Dagstuhl (1993), in Herrsching bei München (1994 und 2003), auf Schloß Rauischholzhausen (1995), in Cunnersdorf in der Sächsischen Schweiz (1996), in Barnstorf bei Bremen (1997), in Riveris bei Trier (1998), in Schauenburg-Elmshagen bei Kassel (1999), in Wien (2000 und 2006 sowie online 2020), in Wendgräben bei Magdeburg (2001), in der Lutherstadt Wittenberg (2002, 2009 und 2018), in Caputh bei Potsdam (2004, 2014 und 2022), in Lauterbad bei Freudenstadt (2005), in Leipzig (2007 und online 2021), in Wettenberg-Launsbach bei Gießen (2008), in Baunatal bei Kassel (2010), in Allrode im Harz (2011), in Prag (2012), in Ilmenau (2013), in Speyer (2015), in Aukrug bei Kiel (2016), in Bonn (2017), in Bremen (2019), in Kaiserslautern (2023) und Göttingen (2024) ausgerichtet.

In diesem Jahr wird der Theorietag von der aktuellen Sprecherin der Fachgruppe mit Unterstützung durch weitere Angehörige der Justus-Liebig-Universität Gießen organisiert. Er findet mit dem vorgelagerten Workshop vom 30. September bis 2. Oktober 2025 im Haus „Sonnenberg“ in Schotten (Vogelsbergkreis) statt.

Zu dem Workshop haben die folgenden eingeladenen Sprecher einen Vortrag zugesagt:

- Uwe Meyer (Technische Hochschule Mittelhessen, Gießen),
- František Mráz (Karlsuniversität Prag),
- Luca Prigioniero (University of Loughborough),
- Inge Schwank (Universität zu Köln),
- Stefan Siemer (Georg-August-Universität Göttingen).

Weitere 15 Vorträge sind zum Theorietag angemeldet. Der vorliegende Tagungsband enthält Kurzfassungen aller 20 Beiträge.

Wir danken den eingeladenen Vortragenden für ihre Bereitschaft, den Workshop mit einem Beitrag zu bereichern sowie allen Teilnehmerinnen und Teilnehmern für ihre Teilnahme und insbesondere jenen, die ebenfalls einen Vortrag halten. Außerdem danken wir der Gesellschaft für Informatik und der Universität Gießen für die Unterstützung dieses Theorietags sowie dem Haus „Sonnenberg“ für die Ausgestaltung der Tagung und das leibliche Wohl.

Wir wünschen allen eine interessante und anregende Tagung sowie einen angenehmen Aufenthalt in Schotten.

Gießen, im September 2025 Bianca Truthe

THEORIE-AG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 1–4.

Reversible Computing

Uwe Meyer

Technische Hochschule Mittelhessen
uwe.meyer@mni.thm.de

Abstract

This is a summary of an invited talk given by the author at the "Theorietag 2025" in Schotten, organized by the Informatics Institute at Justus-Liebig-University GieÃ§en on behalf of the GI working group AFS.

Reversible computation (also known as reverse computing) is a relatively specialized discipline that intersects with theoretical computer science, programming languages, hardware design, and quantum computing. The underlying motivation is rooted in two fundamental principles: (a) the Landauer principle, articulated by the German-American physicist Rolf Landauer, which asserts that the erasure of information inevitably incurs a thermodynamic cost in the form of energy dissipation; and (b) the concept that computations can be performed in a deterministic and reversible manner, thereby obviating the necessity of explicitly constructing inverse algorithms.

The talk explores both aspects. It begins by motivating the physical foundations of the Landauer principle and discussing its implications for hardware design. The first part of the talk focuses on theoretical models of computation âĂŤ such as finite automata, pushdown automata, and Turing machines. We conclude with a discussion of programming languages designed for reversible computation.

1. Foundation

In 1961, Rolf Landauer, a German-American physicist who then worked at IBM, published an article [9] in which he argues that logically irreversible operations will lead to heat dissipation in the order of $k \times ln\, 2 \times T$, where k is the Boltzmann constant and T the temperature measured in Kelvin. While this is a very small amount, it becomes significant as we miniaturize hardware and reach thermodynamic limits. Conventional logic gates such as AND, OR, and NAND are logically irreversible, as they discard information about their inputs. This irreversibility contributes to heat dissipation on modern chips. The so-called Dennard scaling, which held that as transistors shrink, power density remains constant, appears to no longer hold. As a result, clock speeds have stagnated at around 4-6 GHz due to thermal limitations.

In quantum computing, the laws of quantum physics require that quantum operations be represented by unitary matrices U, which satisfy $U^\dagger U = I$. As a consequence, every quantum gate has an inverse. Many quantum algorithms make use of Toffoli gates [12], which are universal for classical reversible computation. A Toffoli gate performs a controlled-controlled-NOT operation and acts on three bits or qubits (a, b, c) as follows: $\text{Toffoli}(a, b, c) = (a, b, c \oplus (a \cdot b))$. In particular, the logical AND operation can be computed reversibly by using the ancillary bit $c = 0$. The Toffoli gate leaves the control bits a and b unchanged and is therefore reversible.

In summary, there are strong conceptual parallels between reversible computation and quantum computing, both of which require logical operations to be invertible.

In theoretical computer science, reversibility has been studied for various devices:

1. Reversible finite automata have been studied in a number of publications. Angluin [1] defines a DFA A to be reversible if A and its inverse (swapping initial and final states, reversing all transitions) are deterministic. In this setting, there must only be one final state. Pin [11] shows that the family of languages accepted by reversible finite automata is a strict subset of REG (the class of regular languages), as for example the language a^*b^+ is not reversible

2. Going to the next level, Kutrib and Malcher studied reversible pushdown automata which (similar to the above definition) are deterministic in both directions [7]. Such automata recognize a family of languages strictly between REG and CFL, and it turns out that the problem of deciding whether a PDA is reversible is P-complete.

3. Similarly, Morita's definition of a reversible Turing machine [10] points in the same direction but necessarily imposes a more general form of backward determinism. Specifically, it requires that if two transitions lead to the same resulting state, then the head movements must be identical, and the symbols written on the tape must be different.

Clearly, not every function is injective, but can all functions be amended with information that makes them injective? And can all programs be made reversible?

Any function $f(x)$ can be turned into an injective function by using a helper function g such that $f^g(x) := (f(x), g(x))$. For example, if $f(x,y) = x+y$, then $f^g(x,y) = (x+y, x)$, as it is possible to determine y if x and $x+y$ are known.

First answers to the second question above were given by Rolf Landauer who suggested to "clutter up ... the machine with intermediate results" ([9]). The technique to record a trace of a computation on a second tape and replay it in reverse is now commonly known as "Landauer embedding" [2]. While being able to reverse a computation, it has got the drawback that the trace is left on one of the tapes as garbage data. In 1973, Charles Bennett [3] proposed a different idea whereby in addition, the results of the computation are copied to a third tape. By carefully replaying the trace, the second tape will eventually be cleansed.

2. Reversible Programming

Thus, all programs are reversible, but often only at the price of additional garbage data. In reversible programming, the aim is to avoid this garbage data by demanding that the initial program P and its inverse P^r are both deterministic, resulting in "clean reversibility" [6].

The most prominent reversible language is Janus, originally created by Darby and Lutz at Caltech, and rediscovered in 2008 by Yokoyama, GlÃijck and Axelsen [13]. Janus is an imperative language that is based on three principles: (i) invertible updates only, (ii) forward and backward determinism, and (iii) inverse semantics. Principle (ii) — even though it is a direct analogue to the reversible machine models, requires mechanisms in control flow that are entirely absent in conventional computing: Since a conditional statement must also be executable in backward direction, its structure must be symmetric. Consequently, there is a second condition

```
                          n0 == 0 -> L1(n0)L2
                          L1(n1) <-
if (n=0)                  n2 := n1 + (1 ^ 0)
  then n+=1               -> L3(n2)
  else n-=1               L2(n3) <-
fi (n=1)                  n4 := n3 - (1 ^ 0)
                          -> L4(n4)
                          L3(n5)L4 <- n5 == 1
```

Figure 1: Janus and corresponding RSSA example

after the then- and the else-part. Similarly, loops require two conditions. Lastly, there must be a counterpart to the declaration (and initialisation) of a local variable, whereby the variable will be initialised when running the program backwards and destroyed when running forwards. But, in order to ensure reversibility, the end conditions of conditionals and loops, as well as the destruction of local variables must furthermore be done in such a way that the program is reversible. That is, during forward execution, the end condition must evaluate to true, when the then-branch was executed and false otherwise. Likewise, a local variable must be destroyed with its current value. Both constructs lead to the fact that the design of a reversible program is often perceived as unintuitive and difficult.

In 2021, a Janus compiler was built at THM [8] that amongst other target languages, compiles Janus to reversible SSA (RSSA) and is also, to the best of our knowledge, the first compiler to perform code optimizations such as common-subexpression-elimination, constant propagation for reversible programs [4].Figure 1 from [8] shows an example of an if-statement in Janus to the left, and its translation into RSSA to the right. Other functional, object-oriented, logic-based, or even concurrent programming languages exist in the academic world, but none of them is widely used. Clean reversibility is difficult to achieve as most functions are not injective. To address this limitation, Gail and Meyer proposed the hybrid Static Single Assignment (HSSA) form [5], which permits controlled irreversibility at explicitly designated points through two primitives: (a) `discard()`, which explicitly deletes non-reversible (garbage) data, and (b) its inverse, `oracle()`, which reintroduces data in a non-deterministic or externally supplied manner. The goal of the introduction of HSSA is to provide an intermediate language as a target language for compilers for "mostly" reversible high-level languages.

In summary, reverse computing offers a promising approach to addressing current and future challenges related to power consumption. However, commercially available hardware and mature programming tools to support its practical implementation are still lacking. Ongoing research represents an early stage in the development of both software and hardware for future computing paradigms.

References

[1] D. ANGLUIN, Inference of Reversible Languages. *J. ACM* **29** (1982) 3, 741–765. https://doi.org/10.1145/322326.322334

[2] H. B. AXELSEN, R. GLÜCK, What Do Reversible Programs Compute? In: M. HOFMANN (ed.), *Foundations of Software Science and Computational Structures*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, 42–56.

[3] C. H. BENNETT, Logical Reversibility of Computation. *IBM Journal of Research and Development* **17** (1973) 6, 525–532.

[4] N. DEWORETZKI, M. KUTRIB, U. MEYER, P.-D. RITZKE, Optimizing Reversible Programs. In: *Reversible Computation: 14th International Conference, RC 2022, Urbino, Italy, July 5–6, 2022, Proceedings*. Springer-Verlag, 2022, 224–238.
https://doi.org/10.1007/978-3-031-09005-9_16

[5] L. GAIL, U. MEYER, Connecting Reversible and Classical Computing Through Hybrid SSA. In: T. Æ. MOGENSEN, L. MIKULSKI (eds.), *Reversible Computation - 16th International Conference, RC 2024, Toruń, Poland, July 4-5, 2024, Proceedings*. Springer-Verlag, 2024, 161–178.
https://doi.org/10.1007/978-3-031-62076-8_11

[6] R. GLÃIJCK, T. YOKOYAMA, Reversible computing from a programming language perspective. *Theoretical Computer Science* **953** (2023), 113429.
https://www.sciencedirect.com/science/article/pii/S0304397522003619

[7] M. KUTRIB, A. MALCHER, Reversible pushdown automata. *Journal of Computer and System Sciences* **78** (2012) 6, 1814–1827. JCSS Multidisciplinary Emerging Networks and Systems (MENS).
https://www.sciencedirect.com/science/article/pii/S0022000011001498

[8] M. KUTRIB, U. MEYER, N. DEWORETZKI, M. SCHUSTER, Compiling Janus to RSSA. In: *Reversible Computation: 13th International Conference, RC 2021, Virtual Event, July 7–8, 2021, Proceedings*. Springer-Verlag, 2021, 64–78.
https://doi.org/10.1007/978-3-030-79837-6_4

[9] R. LANDAUER, Irreversibility and Heat Generation in the Computing Process. *IBM Journal of Research and Development* **5** (1961) 3, 183–191.

[10] K. MORITA, *Reversible Turing Machines*. Springer Japan, Tokyo, 2017.
https://doi.org/10.1007/978-4-431-56606-9_5

[11] J.-E. PIN, On reversible automata. In: I. SIMON (ed.), *LATIN '92*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992, 401–416.

[12] T. TOFFOLI, Reversible Computing. In: *Proceedings of the 7th Colloquium on Automata, Languages and Programming*. Springer-Verlag, Berlin, Heidelberg, 1980, 632–644.

[13] T. YOKOYAMA, H. B. AXELSEN, R. GLÜCK, Principles of a reversible programming language. In: *Proceedings of the 5th Conference on Computing Frontiers*. CF '08, Association for Computing Machinery, New York, NY, USA, 2008, 43–54.
https://doi.org/10.1145/1366230.1366239

THEORIE-TAG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 5–8.

Finite Automata with Translucent Words

František Mráz

Department of software and teacher training, Faculty of Mathematics and Physics
Malostraské nám. 25, 118 00 Prague 1, Czech Republic
frantisek.mraz@mff.cuni.cz

Abstract

Among the various extensions of finite automata that allow for non-sequential input processing, finite automata with translucent letters stand out due to their simplicity. In these automata, each state has a set of letters that are invisible to the automaton while it is in that state. This feature allows them to recognize a semi-linear superset of regular languages, which is not comparable to context-free languages. A more recent extension, known as finite automata with translucent words, associates sets of invisible words with states, making the longest prefix formed from these words invisible as well. This overview summarizes known results on finite automata with translucent words, compares them to finite automata with translucent letters, and highlights open problems related to these types of automata.

1. Introduction

Nondeterministic finite automata with translucent letters (NFAwtl), including their deterministic variant (DFAwtl), were introduced over a decade ago by Benedek Nagy and Friedrich Otto [6]. The model emerged from a reinterpretation of *cooperating distributed systems of stateless deterministic R-automata with a window size of one* [5]. The original motivation for the development of restarting automata was to model linguistic analysis by reduction, particularly for natural languages with free word-order.

In an NFAwtl, each state q has a set of "translucent" letters $\tau(q)$, a subset of its input alphabet. When the automaton is in state q, it effectively becomes blind to the letters in $\tau(q)$. During computation, instead of strictly reading from left to right, the automaton identifies and processes (reads and deletes) the *first non-translucent letter* from the left in the remaining input. If only translucent letters remain or the input is empty, the automaton encounters an end-of-tape marker $\lhd$ and halts, accepting if the current state is final.

NFAwtls are significantly more expressive than standard finite automata. They accept a superset of regular languages, including many context-free and some non-context-free languages. The class $\mathcal{L}(\mathsf{NFAwtl})$ of languages accepted by NFAwtls contains only semi-linear languages and properly includes all rational trace languages [7]. In contrast, the class of languages accepted by DFAwtls, $\mathcal{L}(\mathsf{DFAwtl})$, is incomparable with that of rational trace languages. $\mathcal{L}(\mathsf{NFAwtl})$ is closed under union, concatenation, and Kleene star but not under complementation, intersection with regular sets, or non-erasing morphism. $\mathcal{L}(\mathsf{DFAwtl})$ is not closed under union, product, or Kleene star, but it is closed under complement.

The fixed membership problem is the problem of deciding whether an input word is accepted by a fixed automaton. The fixed membership problem for DFAwtls can be solved in $O(n \log n)$ time, where n is the input length. Emptiness and finiteness are decidable for NFAwtl, but inclusion and equivalence are undecidable even for DFAwtl.

The initial model has inspired several significant extensions, in particular the following [1]:

- In a *non-returning NFAwtl*, the reading head does not return to the start of the remaining input after reading a symbol. Instead, it continues from the position of the deleted letter.
- A *repetitive NFAwtl* can change state and re-read the remaining tape from the beginning upon reaching the end-of-tape marker.

2. Finite Automata with Translucent Words

Benedek Nagy and Friedrich Otto [8] generalized the translucency from sets of letters to finite sets of words. A finite automaton with translucent words reads (and deletes) the first letter from the left that is only preceded by a prefix formed by a product of words that are translucent for the current state. To ensure that the computation relation can be efficiently determined, specific technical restrictions are placed on the translucency mapping $\tau(q)$ (see the definition below).

Definition 2.1 ([2]) *A* finite automaton with translucent words*, or an* NFAwtw*, is defined by a 6-tuple $A = (Q, \Sigma, \lhd, \tau, I, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet, $\lhd \notin \Sigma$ is a sentinel that marks the end of tape, $I \subseteq Q$ is a set of initial states, $\tau : Q \to \mathcal{P}_{\text{fin}}(\Sigma^*)$ is a* translucency mapping*, and*

$$\delta : Q \times (\Sigma \cup \{\lhd\}) \to \mathcal{P}(Q) \cup \{\mathsf{Accept}, \mathsf{Reject}\}$$

is a transition function. We require that, for each state $q \in Q$ and each letter $a \in \Sigma$, $\delta(q, a) \subseteq Q$ and $\delta(q, \lhd) \in \{\mathsf{Accept}, \mathsf{Reject}\}$. The latter means that when the automaton sees the sentinel $\lhd$, it halts and immediately accepts or rejects its input.

For each state $q \in Q$, let $\Sigma_q^{(A)} = \{\, a \in \Sigma \mid \delta(q, a) \neq \emptyset \,\}$ denote the set of letters that A can read in state q. It is required that the set of translucent words $\tau(q)$ satisfies the following two restrictions:

- *For nonempty $\tau(q)$, the set $\tau(q)$ is a finite prefix code.*
- *No word in $\tau(q)$ begins with a letter from the set $\Sigma_q^{(A)}$.*

Actually, this means that the set $\tau(q) \cup \Sigma_q^{(A)}$ is also a finite prefix code.

Automaton A induces on its set of configurations $Q \cdot \Sigma^* \cdot \lhd \cup \{\mathsf{Accept}, \mathsf{Reject}\}$ the following single-step computation relation $\vdash_A$, where $q \in Q$ and $w \in \Sigma^*$:

$$qw \cdot \lhd \vdash_A \begin{cases} q'uv \cdot \lhd, & \text{if } w = uav,\ u \in (\tau(q))^*,\ a \in \Sigma_q^{(A)},\ v \in \Sigma^*, \text{ and } q' \in \delta(q, a), \\ \mathsf{Reject}, & \text{if } w = uav,\ u \in (\tau(q))^*,\ a \in \Sigma \setminus \Sigma_q^{(A)},\ v \in \Sigma^*, \text{ and } av \notin \tau(q) \cdot \Sigma^*, \\ \mathsf{Accept}, & \text{if } w \in (\tau(q))^* \text{ and } \delta(q, \lhd) = \mathsf{Accept}, \\ \mathsf{Reject}, & \text{if } w \in (\tau(q))^* \text{ and } \delta(q, \lhd) = \mathsf{Reject}. \end{cases}$$

A word $w \in \Sigma^*$ *is* accepted by A *if there exists an initial state* $q_0 \in I$ *and a computation* $q_0 w \cdot \lhd \vdash_A^* \mathsf{Accept}$*, where* $\vdash_A^*$ *is the reflexive and transitive closure of* $\vdash_A$*. Then,* $L(A)$ *denotes the* language accepted by A*, and* $\mathcal{L}(\mathsf{NFAwtw})$ *denotes the class of all languages accepted by NFAwtws.*

An NFAwtw $A = (Q, \Sigma, \lhd, \tau, I, \delta)$ *is* deterministic *(or a* DFAwtw*) if* $|I| = 1$ *and* $|\delta(q, a)| \leq 1$ *for each* $q \in Q$ *and* $a \in \Sigma$*. The class of all languages that are accepted by DFAwtws is denoted as* $\mathcal{L}(\mathsf{DFAwtw})$.

3. Expressive Power and Complexity Hierarchies

NFAwtws are strictly more expressive than NFAwtls, with DFAwtws capable of accepting some languages that cannot be recognized by any NFAwtl [8]. However, they still primarily accept semi-linear languages.

The closure and decidability properties of class $\mathcal{L}(\mathsf{NFAwtw})$ are similar to those of class $\mathcal{L}(\mathsf{NFAwtl})$. However, the closure on concatenation and decidability of equivalence are open for $\mathcal{L}(\mathsf{NFAwtw})$.

In [9], two crucial parameters were introduced to classify the expressive capacity and descriptional complexity of NFAwtw.

- *k-cardinality-restriction* limits the maximum number of translucent words in any set $\tau(q)$ to k elements.
- *ℓ-length-restriction* restricts the maximum length of any translucent word in any set $\tau(q)$ to ℓ.

These two parameters induce *infinite strictly ascending two-dimensional hierarchies of language classes* for both DFAwtws and NFAwtws [9]. Notably, a 1-length-restricted NFAwtw is equivalent to an NFAwtl. However, these measures are not independent, as an ℓ-length-restricted NFAwtw enables at most $(s-1) \cdot s^{\ell-1}$-cardinality-restricted automaton, where s is the size of the alphabet. Nevertheless, [2] shows a family of binary languages L_ℓ (for $\ell \geq 1$) demonstrating that the degree of cardinality-restriction can indeed grow exponentially with the degree of length-restriction. The number of translucent words needed to accept L_ℓ grows in proportion to the ℓ-th Fibonacci number.

Furthermore, for repetitive DFAwtws [4], essential decision problems such as emptiness, finiteness, regularity, inclusion, and equivalence are undecidable.

Non-repetitive non-returning NFAwtws are equivalent to finite automata without translucent words and accept only regular languages [3]. However, repetitive nrNFAwtws are more expressive than the corresponding automata with translucent letters and repetitive returning automata with translucent words. Trivially, the fixed membership problem for an nrDFAwtw can be solved in linear space and $O(n^2)$ time, where n is the input length. However, [3] shows that if the input alphabet size is at most two, it can be solved in time $O(n \cdot \log n)$, and if the alphabet size is at least three, in time $O\left(n \cdot (\log n)^2\right)$.

The closure properties of $\mathcal{L}(\mathsf{nrDFAwtw})$ and $\mathcal{L}(\mathsf{nrNFAwtw})$ remain to be studied. The class $\mathcal{L}(\mathsf{nrNFAwtw})$ is closed under union, while $\mathcal{L}(\mathsf{nrDFAwtw})$ is not closed under union and alphabetic morphisms. Although the complement of each language in $\mathcal{L}(\mathsf{nrDFAwtw})$ lies in $\mathcal{L}(\mathsf{nrNFAwtw})$, it is open whether $\mathcal{L}(\mathsf{nrDFAwtw})$ is closed under complementation.

While the standard NFAwtw model retains decidability for key problems like emptiness and finiteness, its repetitive versions lose this property, making fundamental questions undecidable. Emptiness, finiteness, regularity, inclusion, equivalence, and finiteness are undecidable for repetitive DFAwtw and also for nrDFAwtw.

4. Conclusion

Finite automata with translucent words extend those with translucent letters, yet their closure and decidability properties may be identical, though this remains unresolved. It is unknown if the class accepted by DFAwtw is closed under complementation or if NFAwtw is closed under concatenation. Like translucent letter automata, repetitive and non-returning variants lose decidability for key problems such as emptiness and finiteness, making fundamental questions undecidable. Further closure properties for these variants have not been studied.

References

[1] F. MRÁZ, F. OTTO, Non-returning finite automata with translucent letters. In: H. BORDIHN, G. HORVÁTH, G. VASZIL (eds.), *12th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2022)*. EPTCS 367, Open Publishing Association, 2022, 143–159.

[2] F. MRÁZ, F. OTTO, On a measure for the descriptional complexity of finite automata with translucent words. In: A. MALCHER, L. PRIGIONIERO (eds.), *27th International Conference on Descriptional Complexity of Formal Systems (DCFS 2025), Proc.*. LNCS 15759, Springer Nature Switzerland AG, 2025, 180–195.

[3] F. MRÁZ, F. OTTO, *On non-returning finite automata with translucent words*, 2025. Submitted.

[4] F. MRÁZ, F. OTTO, On repetitive finite automata with translucent words. In: N. MOREIRA, L. PRIGIONIERO (eds.), *15th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2025)*. Electronic Proceedings in Theoretical Computer Science 422, Open Publishing Association, 2025, 59–72.

[5] B. NAGY, F. OTTO, CD-systems of stateless deterministic R(1)-automata accept all rational trace languages. In: A.-H. DEDIU, H. FERNAU, C. MARTÍN-VIDE (eds.), *LATA 2010, Proc.*. LNCS 6031, Springer, Berlin, 2010, 463–474.

[6] B. NAGY, F. OTTO, Finite-state acceptors with translucent letters. In: *Proceedings of the 1st International Workshop on AI Methods for Interdisciplinary Research in Language and Biology (BILC 2011)*. SciTePress, 2011, 3–13.

[7] B. NAGY, F. OTTO, Linear automata with translucent letters and linear context-free trace languages. *RAIRO – Theoretical Informatics and Applications* **54** (2020), 3.

[8] B. NAGY, F. OTTO, Finite automata with sets of translucent words. In: J. D. DAY, F. MANEA (eds.), *DLT 2024, Proceedings*. LNCS 14791, Springer, Cham, Switzerland, 2024, 236–251.

[9] B. NAGY, F. OTTO, *A two-dimensional inïňĄnite hierarchy for finite automata with translucent words*, 2024. Submitted.

.

THEORIE-TAG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 9–9.

Descriptional Complexity of Models for Regular Languages

Luca Prigioniero

University of Loughborough
L.Prigioniero@lboro.ac.uk

Finite automata are classical machines used to recognise regular languages. However, a variety of alternative models are also known to characterise this class. In many cases, these alternative devices can represent regular languages much more concisely than classical recognisers.
In this talk, I will present an overview of such models and their capacity to capture regular languages in a compact way. I will focus on selected devices and their properties, discussing recent results as well as open problems in the area.

THEORIE-TAG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 11–14.

From Automata Theory to Classroom Practice: Fostering Algorithmic Thinking from an Early Age Using Dynamic Labyrinths

Inge Schwank

University of Cologne, Germany
{inge.schwank}@uni-koeln.de

Abstract

Grounded in automata theory, Dynamic Labyrinths are computationally universal (Turing complete) yet remain accessible through a deceptively simple arrangement of path and controller elements. Its multi-representational design combines a physical construction set with a multilingual digital app, supporting hands-on, enactive, and virtual-enactive learning. In this way, Dynamic Labyrinths bridge a crucial gap between mathematics education and algorithmic thinking. They are particularly well suited to foster the functional-logical cognitive style — centered on dynamic processes and sequences — which is essential for cultivating a genuine, process-oriented mathematical-computational understanding.

1. Dynamic Labyrinths

Since Seymour Papert's [4] pioneering work, educators have sought effective methods to introduce children to algorithmic thinking. One such approach utilizes Dynamic Labyrinths [1, 2].

- *Theoretical Foundation:* The concept is rooted in mathematical logic and automata theory. Despite their deceptively simple design, Dynamic Labyrinths are computationally universal (Turing complete), meaning they can be constructed to compute any computable function. This makes them a powerful, scalable tool applicable — from kindergarten through university-level education.
- *Practical Implementation:* Mathematical definitions are set aside in favor of hands-on learning. A physical construction set and a complementary digital app provide multiple representation forms:
  - *Enactive:* Building Dynamic Labyrinths with tangible blocks on a pegboard (Figure 1, top right)
  - *Iconic:* Drawing and illustrating Dynamic Labyrinths.
  - *Virtual-Enactive:* Constructing and testing Dynamic Labyrinths digitally via the DL-App (Figure 1, top left and bottom).

The app's interface is currently available in several languages, including Chinese, English, German, Indonesian, Persian, Russian, and Ukrainian.

The core challenge involves the deliberate design of workflows — sequential, organized actions in space and time that involve rule adherence, exploration, and creative problem-solving.

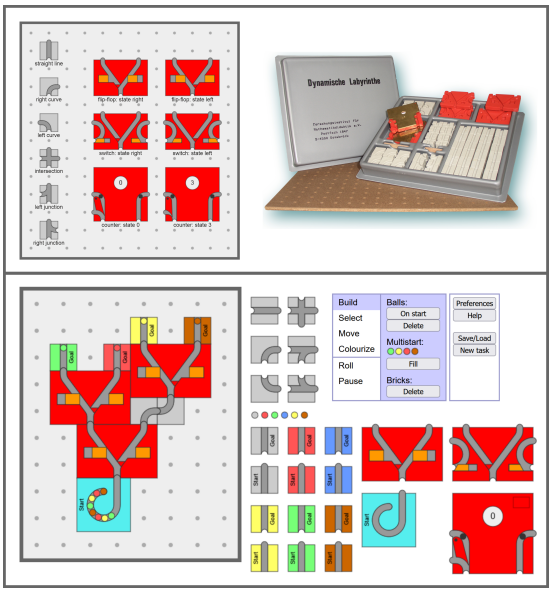


Figure 1: Dynamic Labyrinths Construction Set and DL-App.
Top left: The 6 path elements (gray) and 3 controllers (red). *Top right*: The physical construction set with pegboard. *Bottom*: The digital app for virtual construction.

All bricks function as one-way paths. A Dynamic Labyrinth may have one or more starting points and goals, depending on its purpose. Only one signal propagates from start to goal through the labyrinth at any given time. The core components, shown in Figure 1 (top left), are:

- *Path Elements:* Six basic path types — Straight, Curve (left, right), Junction (left, right), Crossing — form the connection structure.
- *Controller Elements:* Flip-flop, switch, and counter introduce variable internal states, making the signal's path dependent on its history and shaping its future routes.

The incorporation of the Switch and Junction elements allows for the construction of all finite (Mealy-) automata. Combined with the Counter, the system becomes capable of computing any computable function.
Example A: Figure 1 (bottom left) depicts a Dynamic Labyrinth with three flip-flops to sort balls of a specific color sequence (green, yellow, red, brown). Each ball alters the state of certain flip-flops, thereby preparing the correct path for the next ball.
Example B: The Dynamic Labyrinth in Figure 2 determines the parity of the value stored in Counter 1 (C1). A signal travels from the start-brick to the goal-brick, producing a binary result in counter R: a value of 1 for even input, and 0 for odd.

The development of arithmetic understanding is substantially advanced when interwoven with elements of algorithmic reasoning. Such integration fosters mathematical-algorithmic thinking that emphasizes the dynamic character of numbers, highlighting how they emerge, transform, and interact through processes rather than static relations.

Process-Oriented View of Numbers. For example, every natural number can be constructed or deconstructed as a specific multiple of another number plus a remainder. This *constructive-dynamic perspective* fundamentally contributes to a child's numerical construction sense [6]. Dynamic Labyrinths provide an ideal embodied environment for exploring these number construction processes firsthand.

Example C: Figure 2 (right) illustrates the dynamic exploration of numerical structure. The depicted Dynamic Labyrinth is designed to explore and understand the fundamental principle that any number a can be expressed as a multiple of a freely chosen number b plus a specific remainder r (i.e., $a = q \cdot b + r$). For a chosen number (here, $b = 3$), it visually demonstrates the patterns that emerge when numbers are categorized by their remainder, moving beyond simple calculation to reveal their inherent structure.

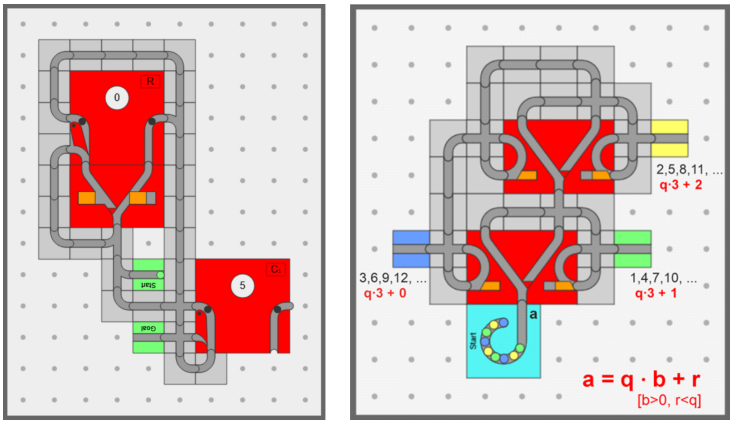


Figure 2: Mathematical Insight with Dynamic Labyrinths
Left: Determining Parity through Process. The Dynamic Labyrinth computes the parity of a value given in counter C1, returning 1 in counter R if the value is even and 0 if it is odd.
Right: Decomposition into Multiple and Remainder. The Dynamic Labyrinth shows in principle the general case of how any number a can be broken down into $a = q \cdot b + r$, where q is the whole number of multiples and r is the remainder.

While resources like [3] provide a solid foundation for teaching (whole) numbers, the connections between early mathematics education and computational thinking remain a largely unexplored research territory. Dynamic Labyrinths, due to their unique combination of simplicity and Turing Completeness, could serve as a golden key to unlock this new way of thinking for children.

2. A Key Cognitive Component in Algorithmic Thinking

While mathematical-algorithmic thinking integrates multiple cognitive processes, a particularly vital foundation lies in distinct reasoning styles. Research on logical thinking, a core component of fluid intelligence, dates back to psychologist Charles Edward Spearman [7], who pioneered figural matrix tasks. Building on this, Schwank (e.g., [5]) identified a critical distinction between two fundamental cognitive approaches to problem-solving:

- *Predicative-Logical Thinking*: Focused on static features, attributes, and invariants.
- *Functional-Logical Thinking*: Focused on dynamic, interwoven processes, sequences of actions, and construction.

Figure 3 shows one of Schwank's matrices, developed to investigate this distinction. The task is designed from a functional-logical perspective, emphasizing dynamic construction sequences. However, a predicative-logical solution based on static features is also possible. Readers are invited to construct a coherent solution using either approach.

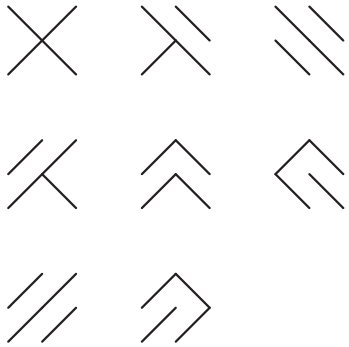


Figure 3: Figural Matrix Task [5] Which figure could reasonably complete the empty space at the bottom right, and why?

The Importance of Functional Thinking. Individuals may exhibit a preference or aptitude for one of these two reasoning styles. However, the ability for functional-logical thinking is especially pivotal, as it greatly facilitates the development of robust arithmetic reasoning and the deep comprehension of processes, transformations, and dynamics. Therefore, actively promoting this mode of thought is essential for fostering a genuine, process-based mathematical-algorithmic understanding. Dynamic Labyrinths are particularly well suited to nurture this capacity by making dynamic processes tangible, visible, and constructible.

References

[1] E. BOERGER, R. GLASCHICK, Logic and Machines: Turing Tradition of the Logic School of Muenster. *The Newsletter of the Formal Aspects of Computing Science (FACS) Specialist Group* **2022-1** (2022) 1, 69–129. ISSN 0950-1231, published by BCS-FACS.

[2] E. COHORS-FRESENBORG, The Benefit of Microworlds in Learning Computer Programming. In: E. BÖRGER (ed.), *Computation Theory and Logic*. Lecture Notes in Computer Science 270, Springer, Berlin, Heidelberg, 1987, 88–100.

[3] B. R. HODGSON, Artefacts and Tasks in the Mathematical Preparation of Teachers of Elementary Arithmetic from a Mathematician's Perspective: A Commentary on Chapter 9. In: M. G. BARTOLINI BUSSI, X. H. SUN (eds.), *Building the Foundation: Whole Numbers in the Primary Grades. The 23rd ICMI Study*. Springer Nature, 2018, 227–250.

[4] S. PAPERT, *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York, 1980.

[5] I. SCHWANK, Challenging Tasks – 13 Years Zingy-Mathematics-Olympiad. *Jurnal Penelitian Pendidikan Matematika Sumba, STKIP Weetebula* **2** (2020) 2, 216–227.

[6] I. SCHWANK, E. SCHWANK, Development of Mathematical Concepts During Early Childhood Across Cultures. In: J. D. WRIGHT (ed.), *The International Encyclopedia of the Social and Behavioral Sciences*, second edition. Elsevier, 2015, 772–784.

[7] C. SPEARMAN, General Intelligence, Objectively Determined and Measured. *American Journal of Psychology* **15** (1904), 201–293.

THEORIE-AG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 15–15.

Enumeration of Word Substructures

Stefan Siemer

Institut für Informatik, Georg-August-Universität Göttingen
stefan.siemer@cs.uni-goettingen.de

This talk provides an overview of enumeration algorithms for various substructures of words:

- Given a string w, another string v is a **subsequence** of w if v can be obtained from w by deleting some of its letters.
- v is an **absent subsequence** of w if it is not a subsequence of w.
- v is a **shortest absent subsequence** of w if it is length-minimal among all absent subsequences of w.
- v is a **minimal absent subsequence** of w if it is absent from w, but every proper subsequence of v occurs in w.
- **Regular patterns with variables** generalize subsequences and allow for the retrieval of structural information from a text.

We will discuss optimal enumeration algorithms with linear-time preprocessing and output-linear delay for subsequences [1] and absent subsequences (both shortest and minimal) [3, 4]. Furthermore, we will explore recent developments in the enumeration of matches of patterns with variables (regular patterns). Finally, we will examine optimal algorithms for the incremental enumeration of these substructures with linear-time preprocessing and constant delay. In this incremental setting, we output only compact edit scripts that describe how each solution differs from the previous one, thereby avoiding the need to output all solutions in full [2].

References

[1] D. ADAMSON, P. FLEISCHMANN, A. HUCH, F. MANEA, P. SARNIGHAUSEN-CAHN, M. WIEDENHÃŰFT, Tight Bounds for the Number of Absent Subsequences. In: *FCT 2025, WrocÅĆaw, Poland, Proceedings*. Lecture Notes in Computer Science, Springer, 2025.

[2] D. ADAMSON, P. GAWRYCHOWSKI, F. MANEA, Enumerating m-Length Walks in Directed Graphs with Constant Delay. In: *LATIN 2024, Puerto Varas, Chile, Proceedings*. Lecture Notes in Computer Science, Springer, 2024.

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

[4] F. MANEA, T. RINGLEB, S. SIEMER, M. WINKLER, Efficiently Finding All Minimal and Shortest Absent Subsequences in a String. *CoRR* **abs/2504.21471** (2025).

This talk is based on joint work with Pawel Gawrychowski, Florin Manea, Tina Ringleb, Paul Sarnighausen-Cahn, Maximilian Winkler

THEORIE-AG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 17–17.

k-Universality of Regular Languages Revisited

Duncan Adamson[(A)] Pamela Fleischmann[(B)] Annika Huch[(B)]
Tore Koß[(C)] Florin Manea[(C)]

[(A)]School of Computer Science, University of St Andrews, UK
duncan.adamson@st-andrews.ac.uk

[(B)]Department of Computer Science, Kiel University, Germany
{fpa,ahu}@informatik.uni-kiel.de

[(C)]Department of Computer Science, University of Göttingen, Germany
{tore.koss,florin.manea}@cs.uni-goettingen.de

Abstract

A subsequence of a word w is a word u such that $u = w[i_1]w[i_2]\cdots w[i_k]$, for some set of indices $1 \leq i_1 < i_2 < \cdots < i_k \leq |w|$. A word w is *k-subsequence universal* over an alphabet Σ if every word over Σ up to length k appears in w as a subsequence. In this paper, we revisit the problem k-ESU of deciding, for a given integer k, whether a regular language, given either as nondeterministic finite automaton or as a regular expression, contains a k-universal word. [Adamson et al., ISAAC 2023] showed that this problem is NP-hard, even in the case when $k = 1$, and an FPT algorithm w.r.t. the size of the input alphabet was given. In this paper, we improve the aforementioned algorithmic result and complete the analysis of this problem w.r.t. other parameters. That is, we propose a more efficient FPT algorithm for k-ESU, with respect to the size of the input alphabet, and propose new FPT algorithms for this problem w.r.t. the number of states of the input automaton and the length of the input regular expression. We also discuss corresponding lower bounds. Our results significantly improve the understanding of this problem.

THEORIE-AG 2025 B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 19–19.

Linear Time Subsequence and Supersequence Regex Matching

Antoine Amarilli[(A)] Florin Manea[(B)] Tina Ringleb[(B)]
Markus L. Schmid[(C)]

[(A)]Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
a3nm@a3nm.net

[(B)]University of Göttingen, Institute for Computer Science and CIDAS, D-37077 Göttingen, Germany
florin.manea@cs.uni-goettingen.de
tina.ringleb@uni-goettingen.de

[(C)]Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany
MLSchmid@MLSchmid.de

It is well-known that checking whether a given string w matches a given regular expression r can be done in quadratic time $O(|w| \cdot |r|)$ and that this cannot be improved to a truly subquadratic running time of $O((|w| \cdot |r|)^{1-\epsilon})$ assuming the strong exponential time hypothesis (SETH). We study a different matching paradigm where we ask instead whether w has a subsequence that matches r, and show that regex matching in this sense can be solved in linear time $O(|w| + |r|)$. Further, the same holds if we ask for a supersequence. We show that the quantitative variants where we want to compute a longest or shortest subsequence or supersequence of w that matches r can be solved in $O(|w| \cdot |r|)$, i. e., asymptotically no worse than classical regex matching; and we show that $O(|w| + |r|)$ is conditionally not possible for these problems. We also investigate these questions with respect to other natural string relations like the infix, prefix, left-extension or extension relation instead of the subsequence and supersequence relation. We further study the complexity of the universal problem where we ask if all subsequences (or supersequences, infixes, prefixes, left-extensions or extensions) of an input string satisfy a given regular expression.

This presentation is based on [1].

References

[1] A. Amarilli, F. Manea, T. Ringleb, M. L. Schmid, Linear Time Subsequence and Supersequence Regex Matching. *CoRR, to appear in the proceedings of MFCS 2025* **abs/2504.16288** (2025). https://doi.org/10.48550/arXiv.2504.16288

THEORIE-AG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 21–24.

2-Word-π-Representable Graphs

Amanita Dietz[(A)] Pamela Fleischmann[(A)] Annika Huch[(A)]
Silas Cato Sacher[(B)]

[(A)]Christian-Albrechts-Universität zu Kiel, Germany
stu215770@mail.uni-kiel.de, {fpa,ahu}@informatik.uni-kiel.de

[(B)]Informatikwissenschaften, Universität Trier, Germany
sacher@uni-trier.de

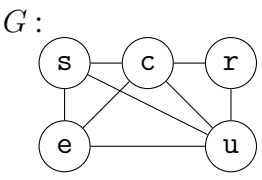
Abstract

This work introduces and investigates 2-word-π-representable graphs where the nodes of the graph correspond to the letters of the two words and there exists an edge between two nodes if the projections of any two letters of both words are equal. The benefit of not only using one word for a representation as introduced by Kitaev and Pyatkin is that every graph is 2-word-π-representable. We present an algorithm that returns two representing words for any graph. Aside, we show that every permutation graph is representable by two 1-uniform words and give constructions how graph operations on 2-word-π-representable graphs can be realised on their representing words which give further insights into the representation of cographs.

1. Introduction

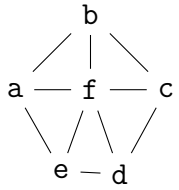
A graph is *word-representable* if there exists a word whose letters are the nodes and the edges in the graph exactly correspond to the pairs of letters that are alternating in the word: two distinct letters are called *alternating* if the removal of all other letters does not yield any two consecutive occurrences of the same letter. A profound introduction to this theory can be found in [9, 7]. For instance, the word `rescues` represents the graph from Figure 1.

Not every graph is word-representable. The smallest non-word representable graph is the wheel graph on 5 nodes [10, 9] (see Figure 2). An asymptotic result about the number of word-representable graphs can be found in [2]. The decision problem whether a graph is word-representable is NP-complete. Thus other approaches to represent graphs by words have been heavily studied, e.g. [1, 3, 5, 6, 8, 11].



$w = \texttt{rescues}$

Figure 1: G represented by w.



$w = \texttt{acbdecef}, v = \texttt{cdaebecf}$

Figure 2: W_5 2-word-π-represented by w and v.

Own Contribution To obtain a more powerful definition of word-representability, we generalised this notion on two words. A graph is *2-word-π-representable* if there exist two words w, v such that the letters of the alphabet of w, v are the nodes of the graph and the projections π on any two letters of w and v are equal iff there is an edge between the corresponding notes. For example, the graph represented by $w = \mathtt{acbdecef}$ and $v = \mathtt{cdaebecf}$ is is the wheel graph on 5 notes (see Figure 2). After the introduction of 2-word-π-representable graphs, we continue their investigation by proving that every graph $G = (V, E)$ is 2-word-π-representable by two words of length $|V| \cdot |\overline{E}|$ where $\overline{E}$ denotes the complement of the edges of G w.r.t. the complete graph. To improve this length, we continue with an investigation of graphs that are representable by two 1-uniform words (every letter occurs exactly once in each word). Those graphs are exactly the permutation graphs. Moreover, we present how basic graph operations like union and join can be realised on 2-word-π-representations of graphs.

Definition 1.1 *Let $w, v \in \Sigma^*$ with $\mathrm{alph}(w) = \mathrm{alph}(v) = A$. Define the graph $G(w, v) = (A, E)$ by $\{\mathtt{a}, \mathtt{b}\} \in E$ if and only if $\pi_{\mathtt{a},\mathtt{b}}(v) = \pi_{\mathtt{a},\mathtt{b}}(w)$ for all $\mathtt{a}, \mathtt{b} \in A$. If there exists $w, v \in \Sigma^*$ such that for a given graph G, we have $G = G(w, v)$ then G is called* 2-word-π-representable.

2. Construction of Words for a Given Graph

In this section, we show that every graph is 2-word-π-representable.

Lemma 2.1 *For all $w, v, u \in \Sigma^*$ with $\mathrm{alph}(w) = \mathrm{alph}(v) \supseteq \mathrm{alph}(u)$ we have $G(w, v) = G(wu, vu) = G(uw, uv)$.*

Note that, Theorem 2.1 is holding for pre- and appending the same word. To increase readability we will only state results for appending in the following.

Lemma 2.2 *Let $w, v \in \Sigma^*$ with $\mathrm{alph}(w) = \mathrm{alph}(v)$ and $\{\mathtt{a}, \mathtt{b}\} \in E(G(w, v))$ for some $\mathtt{a}, \mathtt{b} \in \mathrm{alph}(w)$. Then there exist 1-uniform words $u_w, u_v \in \Sigma^*$, such that $E(G(wu_w, vu_v)) = E(G(w, v)) \setminus \{\{\mathtt{a}, \mathtt{b}\}\}$.*

Proof. W.l.o.G. $|\mathrm{alph}(w)| > 2$. Let $\mathtt{a}, \mathtt{b} \in \mathrm{alph}(w)$ and $x \in (\mathrm{alph}(w) \setminus \{\mathtt{a}, \mathtt{b}\})^*$ with $|x|_{\mathtt{c}} = 1$ for all $\mathtt{c} \in \mathrm{alph}(w) \setminus \{\mathtt{a}, \mathtt{b}\}$. Let $u_w = x\mathtt{ab}$ and $u_v = x\mathtt{ba}$. These words are 1-uniform by construction. Let $\mathtt{c} \in \mathrm{alph}(w) \setminus \{\mathtt{a}, \mathtt{b}\}$ and $\mathtt{d} \in \mathrm{alph}(w) \setminus \{\mathtt{c}\}$. Then we obtain $\pi_{\mathtt{c},\mathtt{d}}(wu_w) = \pi_{\mathtt{c},\mathtt{d}}(vu_v)$ iff $\pi_{\mathtt{c},\mathtt{d}}(w) = \pi_{\mathtt{c},\mathtt{d}}(v)$ and $\pi_{\mathtt{c},\mathtt{d}}(u_w) = \pi_{\mathtt{c},\mathtt{d}}(u_v)$. Since the latter one holds by construction, we have $\{\mathtt{c}, \mathtt{d}\} \in E(G(vu_v, wu_w))$ iff $\{\mathtt{c}, \mathtt{d}\} \in E(G(v, w))$. Also by construction, we get that $\pi_{\mathtt{a},\mathtt{b}}(wu_w) \neq \pi_{\mathtt{a},\mathtt{b}}(vu_v)$. □

Theorem 2.3 *For every graph $G = (V, E)$ there exist $w, v \in \Sigma^*$ with $\mathrm{alph}(w) = \mathrm{alph}(v) = V$, such that $G(w, v) = G$.*

Proof. Clearly, the complete graph is 2-word-π-representable. For every other graph, the words w, v are constructed iteratively by removing one edge after another with Theorem 2.2, starting with $w = v = \varepsilon$. After removing all edges $\{\mathtt{a}, \mathtt{b}\} \in \overline{E}$, only the edges in E are left. This takes $|\overline{E}|$ iterations and since in each iteration $|V|$ letters are added to the words w, v, they are of length $|w| = |v| = |\overline{E}| \cdot |V|$. □

$w = \varepsilon$	$w = \mathtt{bdac}$	$w = \mathtt{bdac} \cdot \mathtt{adbc}$	$w = \mathtt{bdacadbc} \cdot \mathtt{acbd}$
$v = \varepsilon$	$v = \mathtt{bdca}$	$v = \mathtt{bdca} \cdot \mathtt{adcb}$	$v = \mathtt{bdcaadcb} \cdot \mathtt{acdb}$

Figure 3: Construction of w, v for G.

Consider the graph $G = (V, E)$ with vertices $V = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$ and edges $E = \{\{\mathtt{a}, \mathtt{b}\}, \{\mathtt{a}, \mathtt{d}\}, \{\mathtt{c}, \mathtt{d}\}\}$. Figure 3 shows the corresponding graphs for the iterations. The construction from the proof of Theorem 2.3 yields an algorithm to construct words that 2-word-π-represent a given graph.

Theorem 2.4 *A permutation graph G given by the permutation $\sigma : [n] \to [n]$ is 2-word-π-represented by the word $1 \cdots n$ and the reverse of the one-line notation of σ interpreted as a word. The set of graphs that are 2-word-π-representable by 1-uniform words is the class of permutation graphs.*

Remark 2.5 *It is folklore that permutations graphs are closed under graph complement [4]. For two 1-uniform words w and v such that* $\mathrm{alph}(w) = \mathrm{alph}(v)$, $\overline{G(w, v)} = G(w, v^R) = G(w^R, v)$.

3. k-Uniformly 2-Word-π-Representable Graphs

The full characterisation (Theorem 2.3) of 1-uniform 2-word-π-representability motivates the following definition.

Definition 3.1 *Define $\mathcal{G}_k := \{G \mid \exists w, v \in V(G)^* : w$ and v are k-uniform and $G = G(w, v)\}$ for each $k \in \mathbb{N}$.*

Lemma 3.1
- *For each $k \in \mathbb{N}$, $\mathcal{G}_k$ is hereditary.*
- *For each $k \in \mathbb{N}$, $\mathcal{G}_k \subseteq \mathcal{G}_{k+1}$. (This can be proven by Theorem 2.1.)*
- *$\bigcup_{k=1}^{\infty} \mathcal{G}_k$ is the class of all graphs. (This is an immediate consequence of Theorem 2.3.)*
- *$\mathcal{G}_1 \subsetneq \mathcal{G}_2$.*

Conjecture 3.2 *For each $k \in \mathbb{N}$, the inclusion $\mathcal{G}_k \subseteq \mathcal{G}_{k+1}$ is proper.*

4. Operations on Graphs

In the context of graph theory, it is reasonable to analyse the standard operations on graphs and how to realise them on 2-word-π-representable graphs. For the rest of this section let $w, w', v, v' \in \Sigma^*$ with $\mathrm{alph}(w) \cap \mathrm{alph}(w') = \emptyset$, such that $G(w, v)$ and $G(w', v')$ are 2-word-π-representable.

Theorem 4.1 *If $|w|_{\mathtt{a}} = |v|_{\mathtt{a}}$, $|w'|_{\mathtt{b}} = |v'|_{\mathtt{b}}$ for all $\mathtt{a} \in \mathrm{alph}(w), \mathtt{b} \in \mathrm{alph}(w')$ then the join $G(w, v) + G(w', v')$ is given by $G(ww', vv')$.*

Theorem 4.2 *We have* $G(w,v) \cup G(w',v') = G(ww',v'v)$.

Remark 4.3 *Cographs can be* 2*-word-*π*-represented by 1-uniform words according to Theorem 2.4. Theorem 4.1, Theorem 4.2 and Theorem 2.5 give further insight into how to construct a* 2*-word-*π*-representation for cographs based on their inductive definition.*

5. Open Problems

It is possible to prove with an information-theoretic argument that for every $k \in \mathbb{N}$, the class $\mathcal{G}_k$ is not the class of all graphs. However we did not find an example of a graph G such that $G \notin \mathcal{G}_2$ and we only proved $\mathcal{G}_k \subsetneq \mathcal{G}_{k+1}$ for $k = 1$. We conjecture that this statement holds for every k. Another problem that we suggest is to determine the smallest k for a given graph G such that $G \in \mathcal{G}_k$. Moreover, the problem of given a graph G and a word w, determine a second word v, such that $G(w,v) = G$ is in our opinion an interesting problem that should be pursued.

References

[1] G.-S. CHEON, J. KIM, M. KIM, S. KITAEV, A. PYATKIN, On k-11-representable graphs. *J. Comb.* **10** (2019) 3.

[2] A. COLLINS, S. KITAEV, V. LOZIN, New results on word-representable graphs. *Discret. Appl. Math.* **216** (2017), 136–141.

[3] Z. FENG, H. FERNAU, P. FLEISCHMANN, K. MANN, S. C. SACHER, Generalized Word-Representable Graphs. 2024.
`https://arxiv.org/abs/2411.03274`

[4] M. C. GOLUMBIC, Chapter 7 - Permutation graphs. In: *Algorithmic Graph Theory and Perfect Graphs*. Annals of Discrete Mathematics 57, Elsevier, 2004, 157–170.

[5] M. JONES, S. KITAEV, A. PYATKIN, J. REMMEL, Representing Graphs via Pattern Avoiding Words. *Electron. J. Comb.* **22** (2015) 2, 2.

[6] B. KENKIRETH, A. MALHOTRA, On Word-Representable and Multi-Word-Representable Graphs. In: *DLT*. LNCS 13911, 2023, 156–167.

[7] S. KITAEV, A Comprehensive Introduction to the Theory of Word-Representable Graphs. In: É. CHARLIER, J. LEROY, M. RIGO (eds.), *Developments in Language Theory - 21st International Conference, DLT*. LNCS 10396, Springer, 2017, 36–67.

[8] S. KITAEV, Existence of u-representation of graphs. *J. Graph Theory* **85** (2017) 3, 661Ű668.

[9] S. KITAEV, V. V. LOZIN, *Words and Graphs*. Monographs in Theoretical Computer Science. An EATCS Series, Springer, 2015.

[10] S. KITAEV, A. V. PYATKIN, On Representable Graphs. *Journal of Automata, Languages and Combinatorics* **13** (2008) 1, 45–54.

[11] M. PETKOVŠEK, Letter graphs and well-quasi-order by induced subgraphs. *Discrete Mathematics* **244** (2002) 1, 375–388.

THEORIE-AG 2025 B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 25–27.

Subsequence Matching and Analysis Problems for Automata with Translucent Letters

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

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

[(B)]University of Göttingen, Germany
{bela.klein,tore.koss,timo.specht}@uni-goettingen.de
florin.manea@informatik.uni-goettingen.de

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

Abstract

In this paper, we settle a series of algorithmic problems related to the (sets of) subsequences occurring in the strings of a given formal language, represented by a deterministic finite automaton with translucent letters accepting it, which were left open in [Fazekas et al., ISAAC 2024]. Firstly, we show that one can decide in polynomial time whether all elements of a given language have a given string as subsequence. We continue by considering the problems of deciding for a given language $L \subset \Sigma^*$ and a given positive integer $k > 0$ (respectively, for all integers $k > 0$) whether there exists a k-universal string in L (i.e., a string which has all possible length-k strings over Σ as subsequences). For both problems we show that they are NP-hard, and in the case of the second problem we give an NP-upper bound and thus achieve NP-completeness. For the other problem, i.e. for the question whether there exists a k-universal word in L given a language $L \subset \Sigma^*$ and a positive integer $k > 0$, we present a PSPACE algorithm.

1. Introduction

A string v is a subsequence of a string w, denoted $v \leq w$, if there exist (possibly empty) strings $x_1, \ldots, x_{\ell+1}$ and $v_1, \ldots, v_\ell$ such that $v = v_1 \cdots v_\ell$ and $w = x_1 v_1 \cdots x_\ell v_\ell x_{\ell+1}$. That is, v can be obtained from w by removing some of its letters.

Following [2, 1, 3], the focus of this paper is the study of the subsequences of strings of a formal language, the main idea behind this series of works being to extend the fundamental problems related to matching subsequences in a string and to the analysis of the sets of subsequences of a single string to the case of sets of strings. Firstly, [1] and its predecessor [2] approached algorithmic matching and analysis problems related to the universality of regular languages. More precisely, a string over Σ is called k-universal if its set of subsequences includes all strings of length k over Σ, and the respective papers presented algorithmic results and hardness results for problems asking whether a given regular language contains a k-universal

string (for some given k, or, respectively, for all $k > 0$), and if all the words of a given regular language are k-universal (for a given $k > 0$). As a direct predecessor of this paper, [3] proposed a more systematic approach to such problems and defined a series of five algorithmic matching and analysis problems (listed below) related to matching subsequences in the words of regular, context-free, and context-sensitive languages and the universality of such languages, and showed results related to their decidability and complexity. The reader interested in the motivation behind the study of these subsequence-centered matching and analysis problems for formal languages, and their connections to other areas, as well as the study of subsequences in general, is referred to [3, 4] and the references therein.

The approached problems and an overview of our results: This work is a direct continuation of [3], where the following five problems were defined and, as mentioned above, thoroughly considered for regular, context-free and context-sensitive languages.

Problem 1 ($\exists$-Subsequence) *Given language L by a machine (grammar) M accepting (resp., generating) it and a string w, is there a string $v \in L$ such that $w \leq v$?*

Problem 2 ($\forall$-Subsequence) *Given language L by a machine (grammar) M accepting (resp., generating) it and a string w, do all strings $v \in L$ satisfy $w \leq v$?*

Problem 3 ($\exists$-k-universal) *Given language L by a machine (grammar) M accepting (resp., generating) it and integer k, is there a k-universal string in L?*

Problem 4 ($\forall$-k-universal) *Given language L by a machine (grammar) M accepting (resp., generating) it and integer k, are all strings of L k-universal?*

Problem 5 (∞-universal) *Given language L by a machine (grammar) M accepting (resp., generating) it, are there k-universal strings in L, for all integers $k > 0$?*

To give some intuition on our terminology, Problems 1 and 3 can be seen as *matching problems* (asking to find a string which contains a certain subsequence or set of subsequences), while the other three problems are *analysis problems* (asking to decide properties concerning multiple strings of the language).

In [3], the authors conclude that the complexity of the approached problems is, to a certain extent, similar when the input languages are given by finite automata and context-free grammars: Problems 1, 2, 4, and 5 can be solved in polynomial time, while Problem 3 is NP-hard and FPT with respect to the size of the input alphabet as parameter in the case of CFL, but NP-complete in the regular case. Moreover, all problems become undecidable for context-sensitive languages. As such, a well-motivated, natural question is to identify classes of languages (ideally, strictly included in the class of context-sensitive languages, and also generalizing the class of regular languages) which exhibit a different behaviour. To this end, they suggested considering the class of languages accepted by deterministic finite automata with translucent letters (a novel, non-classical model of automata, see, e.g., [6, 5] and the references therein). As an initial result, in [3], it was shown that Problem 1 is NP-complete in this setting; the other problems were left open. Here, we make significant progress in the investigation of these problems, when the input language is given as a deterministic finite automaton with translucent letters. Firstly,

we give a polynomial-time algorithm for Problem 2. Secondly, we show the NP-hardness for Problems 3 and 5, by giving a reduction from the Hamiltonian cycle problem. Thirdly, we present a PSPACE algorithm for Problem 3 and in the last section of this paper we focus on the proof that Problem 5 is contained in NP. In particular, the P-membership of Problem 2 and the NP-membership of Problem 5 are obtained by rather involved, novel combinatorial insights and arguments.

References

[1] D. ADAMSON, P. FLEISCHMANN, A. HUCH, T. KOSS, F. MANEA, k-Universality of Regular Languages Revisited. In: V. CHAU, C. DÜRR, M. LI, P. LU (eds.), *Proc. 6th International Joint Conference on Theoretical Computer Science âĂŞ 19th Frontier of Algorithmic Wisdom (IJTCS FAW 2025)*. Lecture Notes in Computer Science 15828, Springer Nature Singapore, Singapore, 2025, 16–32.

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

[3] S. Z. FAZEKAS, T. KOSS, F. MANEA, R. MERCAŞ, T. SPECHT, Subsequence Matching and Analysis Problems for Formal Languages. In: *Proc. 35th International Symposium on Algorithms and Computation, (ISAAC 2024)*. LIPIcs 322, 2024, 28:1–28:23.

[4] M. KOSCHE, T. KOSS, F. MANEA, S. SIEMER, Combinatorial Algorithms for Subsequence Matching: A Survey. In: *Proc. 12th International Workshop on Non-Classical Models of Automata and Applications, (NCMA 2022)*. EPTCS 367, 2022, 11–27.

[5] V. MITRANA, A. PĂUN, M. PĂUN, J. SÁNCHEZ-COUSO, Jump complexity of finite automata with translucent letters. *Theoretical Computer Science* **992** (2024), 114450.

[6] F. OTTO, A Survey on Automata with Translucent Letters. In: *Proc. 27th International Conference on Implementation and Application of Automata, (CIAA 2023)*. 14151, 2023, 21–50.

THEORIE-AG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 29–33.

Varianten von Geffert-Normalformen für Einfüge-Lösch-Systeme

Henning Fernau

Informatikwissenschaften, Universität Trier
fernau@uni-trier.de

Zusammenfassung

Geffert-Normalformen haben sich als ein wichtiges Werkzeug erwiesen für den Nachweis dafür, dass ein Grammatikmechanismus Turing-mächtig ist und doch mit beschränkten Ressourcen auskommt. Für Einfüge-Lösch-Systeme wurden in der Literatur sogenannte Spezielle GNF eingeführt, die nicht mehr auf eine kleine Zahl von Nichtterminalen fokussieren, sondern mehr auf strukturelle Eigenschaften. Wir diskutieren in dieser Arbeit, wie man weitere Strukturen erzwingen kann, die dann einfachere Beweise oder auch stärkere Ergebnisse für (regulierte) Einfüge-Lösch-Systeme gestatten, als dies mit früheren Methoden möglich war.

1. Einleitung 1: Geffert-Normalformen

Die heute so genannten Geffert-Normalformen wurden von Viliam Geffert in seiner Dissertation [11] eingeführt und später in [12] veröffentlicht. Sie sind in erster Linie dazu entworfen worden, um alle rekursiv aufzählbaren Sprachen mit möglichst wenigen Nichtterminalen beschreiben zu können. Ein scheinbarer Nebeneffekt ist, dass die Regeln eine gewisse Struktur der Satzformen mit sich bringen. Diese ist jedoch bei Beweisführungen sehr hilfreich.

Später wurden in [14] spezielle Geffert-Normalformen (SGNF) eingeführt. Diese fokussieren nun völlig auf den strukturellen Aspekt und opfern die Zahl der Nichtterminale völlig. Das ist insbesondere sinnvoll, wenn es um Beschreibungskomplexitätsbetrachtungen bei Einfüge-Lösch-Systemen geht, wie wir noch sehen werden und wie in [2] genauer dargelegt wurde. Wir werden im Folgenden den Nutzen von zwei Varianten beschreiben, die wir nun einführen möchten. Wir beginnen mit der *separierenden* Variante (space-separating, ss):

Satz 1.1 *[5] Jede RE-Sprache $L \subseteq \Sigma^*$ lässt sich beschreiben durch eine Typ-0 Grammatik $G = (N, \Sigma, P, S)$ in ssSGNF, d.h.:*

- $N = N' \cup N''$, $N' \cap N'' = \emptyset$, $N'' = \{A, B, C, D, E, F\}$, $S, S' \in N'_0$, $N' = N'_0 \cup N'_1$.
- *Die kontextfreien Regeln in P haben die Form $X \to Yb$, $X \to b'Y$ oder $S' \to \lambda$ mit*
  - $X \in N'_0$, $Y \in N'_1$, $b \in \{F\}$, $b' \in \{E\}$ *oder*
  - $X \in N'_1$, $Y \in N'_0$, $b \in \{B, D\}$, $b' \in \{A, C\}$ *oder*
  - $X, Y \in N'_0$, $X \neq Y$, $b \in \Sigma$.

- *Ferner sind in* P*:* $AB \to \lambda$*,* $CD \to \lambda$ *und* $EF \to \lambda$*.*

Wir würden also gerne Ableitungen der Bauart $S \Rightarrow EZ \Rightarrow EAZ'$ oder $S \Rightarrow ZF \Rightarrow Z'BF$ sehen. Die Nichtterminale können überdies dafür sorgen, dass zunächst auf der rechten Seite der Satzform nur Terminalzeichen produziert werden. Eine wichtige Eigenschaft ist, dass keine ableitbare Satzform ein Teilwort der Form RR enthält mit $R \in N''$.

Eine zweite Variante greift auf eine weitere Geffert-Normalform zurück.

Satz 1.2 *[8] Jede* RE*-Sprache* $L \subseteq \Sigma^*$ *lässt sich beschreiben durch eine Typ-0 Grammatik* $G = (N, \Sigma, P, S)$ *in* ABC*-SGNF, d.h.:*

- $N = \overbrace{N'_0 \cup N'_1 \cup \{S'\}}^{N'} \cup N''$ *mit* $N'' = \{A, B, C\}$*,* N'_0 *enthält das Nichtterminal* S*;*
- *die einzige nicht-kontextfreie Regel in* P *ist* $ABC \to \lambda$*;*
- *die kontextfreien Regeln in* P *sind (i)* $S' \to \lambda$ *oder (ii)* $X \to Y_1Y_2$ *mit* $X \in N'_0$*,* $Y_1Y_2 \in (\{A, B\}(N'_1 \cup \{S'\})) \cup ((N'_1 \cup \{S'\})(T \cup \{B, C\}))$ *oder (iii)* $Y \to X_1X_2$ *mit* $Y \in N'_1$*,* $X_1X_2 \in (\{A, B\}(N'_0 \cup \{S'\})) \cup ((N'_0 \cup \{S'\})(T \cup \{B, C\}))$*.*

2. Einleitung 2: Einfüge-Lösch-Systeme

Einfüge-Lösch-Systeme (ELS, englisch: ID systems) wurden ursprünglich als formales Modell im Bereich *DNA Computing* eingeführt [13]. Es handelt sich hierbei im Grunde genommen um Phrasenstrukturgrammatiken mit zwei (eingeschränkten) Grundoperationen, dem Einfügen bzw. Löschen von Zeichenketten in der augenblicklichen Satzform, jeweils unter möglicher Betrachtung des Kontexts der Einfüge- bzw. Löschstelle. Genauer entspricht eine $(\alpha, \beta, \gamma)_I$ notierte Einfügeregel der Ersetzungsregel $\alpha\gamma \to \alpha\beta\gamma$, während eine $(\alpha, \beta, \gamma)_D$ notierte Löschregel der Ersetzungsregel $\alpha\beta\gamma \to \alpha\gamma$ entspricht. Die Beschreibungskomplexität von solchen System lässt sich daher beschreiben durch ein Tupel der Form $(n, i', i''; m, j', j'')$ mit folgender Bedeutung für die Regeln besagten Systems:

- n ist die maximale Länge einer eingefügten Zeichenkette,
- i' ist die maximale Länge irgendeines linken Kontexts einer Einfügeregel,
- i'' ist die maximale Länge irgendeines rechten Kontexts einer Einfügeregel,
- m, j', j'' haben analoge Bedeutungen für Löschregeln.

Sodann betrachtet man Sprachfamilien wie $\mathsf{ID}(n, i', i''; m, j', j'')$, die alle Sprachen enthalten, die durch ELS beschrieben werden können, deren Beschreibungskomplexität (komponentenweise) durch $(n, i', i''; m, j', j'')$ beschränkt ist. Beispielsweise ist bekannt, dass $\mathsf{ID}(1, 1, 1; 1, 1, 1) = \mathsf{RE}$ gilt, siehe [15]. Gleichzeitig ist bekannt, dass mancherlei Parameterbeschränkungen zu schwach sind, um ganz RE zu beschreiben.

Schon früh wurden Regulationsmechanismen für ELS betrachtet und die Trade-offs zwischen den genannten und weiteren Parametern wie der Zahl der Komponenten in graphgesteuerten Systemen untersucht. Wir fokussieren uns nun auf besonders einfache derartige Steuerungsmechanismen, beruhend auf einfachsten Kontrollgraphen wie Pfade [3, 4, 6], Sterne [7, 9] und Kreise [1, 8]. In der angegebenen Literatur finden sich auch die genauen Definitionen.

3. Ein vereinfachter Beweis für ein wichtiges Resultat

Satz 3.1 $\mathit{ID}(1,1,1;1,1,1) = \mathit{RE}$.

Takahara und Yokomori [15] führen ihren Beweis über die bekannte Penttonen-Normalform. Der Beweis füllt fast den gesamten Zeitschriftenartikel. Auch wenn wir im Folgenden nur die Idee einer alternativen Simulation skizzieren, dürfte ihre Korrektheit leichter ersichtlich sein.
Beweis. Wir gehen davon aus, dass die RE-Sprache $L \subseteq \Sigma^*$ als ssSGNF Grammatik vorliegt (s.o.). Das simulierende ELS habe Sonderzeichen $\#_\ell, \#_r$ (sowie Regelmarker wie p, p', p'') und starte mit $\#_\ell S \#_r$. Eine kontextfreie Regel $p : Y \to \alpha_1\alpha_2$ mit $Y \notin \{\alpha_1, \alpha_2\}$ wird simuliert durch:

$$\boxed{(X,p,Y)_I,\ (Y,p',Z)_I,\ (p,Y,p')_D,\ (p,\alpha_1,p')_I,\ (\alpha_1,\alpha_2,p')_I,\ (X,p,\alpha_1)_D,\ (\alpha_2,p',Z)_D.}$$

für Kontexte $X \in \{\#_\ell, A, C\}$ und $Z \in \{\#_r, B, D\} \cup \Sigma$. Die löschende Regel $AB \to \lambda$ wird simuliert durch: $(E,A,B)_D$ und $(E,B,F)_D$; analog behandele $CD \to \lambda$. Für $EF \to \lambda$ haben wir $(X,E,F)_D$ und $(X,F,Z)_D$ mit $X \in \{\#_\ell, A, C\}$ und $Z \in \{\#_r, B, D\} \cup \Sigma$. Löschende kontextfreie Regeln sind trivial; sie bewirken auch das Löschen von $\#_\ell, \#_r$. □

4. Konstruktionsbeispiele für Regulationen

Als Erstes zeigen wir eine Konstruktion aus [9], die eine besondere Verwendung von ssSGNF illustriert; es geht um Steuerung über Sterngraphen. Bei den Simulationen wird für die kontextfreien (linear-artigen) Regeln meist zwischen $p : X \to bY$, $q : X \to Yb$ und $h : S' \to \lambda$ unterschieden.

Satz 4.1 $\mathit{GCID}_S(3;2,1,0;1,1,0) = \mathit{RE}$.

Komponente C_1	**Komponente** C_2	**Komponente** C_3
$p1.1 : (1,(\lambda,p,\lambda)_I,2)$ $p1.2 : (1,(p,bY,\lambda)_I,2)$	$p2.1 : (2,(p,X,\lambda)_D,1)$ $p2.2 : (3,(\lambda,p,\lambda)_D,1)$	
$q1.1 : (1,(\lambda,q,\lambda)_I,2)$ $q1.2 : (1,(q,Yb,\lambda)_I,2)$	$q2.1 : (2,(q,X,\lambda)_D,1)$ $q2.2 : (2,(\lambda,q,\lambda)_D,1)$	
$h1.1 : (1,(\lambda,hh',\lambda)_I,2)$ $h1.2 : (1,(h,h',\lambda)_D,2)$	$h2.1 : (3,(h',S',\lambda)_D,1)$ $h2.2 : (3,(\lambda,h,\lambda)_D,1)$	
$f1.1 : (1,(A,B,\lambda)_D,3)$		$f3.1 : (3,(\lambda,A,\lambda)_D,1)$
$g1.1 : (1,(C,D,\lambda)_D,3)$		$g3.1 : (3,(\lambda,C,\lambda)_D,1)$
$e1.1 : (1,(E,F,\lambda)_D,3)$		$g3.1 : (3,(\lambda,E,\lambda)_D,1)$

Es folgen zwei Beispiele von Simulationen aus [8]. Die Kreisgraphensteuerung kann man auch als zeitvariiende Systeme interpretieren.

Satz 4.2 $\mathit{GCID}_C(3;2,1,0;\underline{1,0,1}) = \mathit{GCID}_C(3;2,0,1;1,1,0) = \mathit{RE}$.
$\mathit{GCID}_C(3;2,1,0;1,1,0) = \overline{\mathit{GCID}}_C(3;2,0,1;\underline{1,0,1}) = \mathit{RE}$.

Dabei werden die f- / g-Regeln (immer zusammen mit der e-Regel) in den unterstrichenen Fällen wie folgt simuliert:

C_1	C_2	C_3	C_1	C_2	C_3
$(\lambda,f,\lambda)_I$	$(\lambda,F,f)_D$	$(\lambda,B,f)_D$	$(\lambda,g,\lambda)_I$	$(\lambda,F,g)_D$	$(\lambda,D,g)_D$
$(\lambda,A,f)_D$	$(\lambda,E,f)_D$	$(\lambda,f,\lambda)_D$	$(\lambda,C,g)_D$	$(\lambda,E,g)_D$	$(\lambda,g,\lambda)_D$

Für die kontextfreien Regeln nehmen wir:

Einfügeparameter $(2,1,0)$

C_1	C_2	C_3
$(\lambda,p,\lambda)_I$	$(\lambda,X,p)_D$	$(\$,\#,\lambda)_I$
$(p,bY,\lambda)_I$	$(\lambda,p,\lambda)_D$	$(\lambda,\#,\lambda)_D$
$(\lambda,q,\lambda)_I$	$(\lambda,X,q)_D$	$(\$,\#,\lambda)_I$
$(q,Yb,\lambda)_I$	$(\lambda,q,\lambda)_D$	$(\lambda,\#,\lambda)_D$
$(\lambda,h,\lambda)_I$	$(\lambda,S',h)_D$	$(\lambda,h,\lambda)_D$

Einfügeparameter $(2,0,1)$

C_1	C_2	C_3
$(\lambda,p,\lambda)_I$	$(\lambda,X,p)_D$	$(\lambda,\#,\$)_I$
$(\lambda,bY,p)_I$	$(\lambda,p,\lambda)_D$	$(\lambda,\#,\lambda)_D$
$(\lambda,q,\lambda)_I$	$(\lambda,X,q)_D$	$(\lambda,\#,\$)_I$
$(\lambda,Yb,q)_I$	$(\lambda,q,\lambda)_D$	$(\lambda,\#,\lambda)_D$
$(\lambda,h,\lambda)_I$	$(\lambda,S',h)_D$	$(\lambda,h,\lambda)_D$

Wir starten mit dem Axiom $\$S$ und terminieren schließlich mit einem weiteren Marker t:

C_1	C_2	C_3
$(\$,t,\lambda)_I$	$(\lambda,\$,t)_D$	$(\lambda,t,\lambda)_D$

Satz 4.3 $\mathsf{GCID}_C(5;1,1,0;1,0,1) = \mathsf{GCID}_C(5;1,0,1;1,1,0) = \mathsf{RE}$.

Hier werden die Regeln einer Grammatik in ABC-SGNF wie folgt simuliert.

C_1	C_2	C_3	C_4	C_5	simuliert
$(\lambda,p,\lambda)_I$	$(\lambda,X,p)_D$	$(p,Y,\lambda)_I$	$(p,b,\lambda)_I$	$(\lambda,p,\lambda)_D$	$p : X \to bY$
$(\lambda,q,\lambda)_I$	$(\lambda,X,q)_D$	$(q,b,\lambda)_I$	$(q,Y,\lambda)_I$	$(\lambda,q,\lambda)_D$	$q : X \to Yb$
$(\lambda,h,\lambda)_I$	$(\lambda,S',h)_D$	$(h,h,\lambda)_I$	$(\lambda,h,h)_D$	$(\lambda,h,\lambda)_D$	$h : S' \to \lambda$
$(\lambda,r,\lambda)_I$	$(\lambda,C,r)_D$	$(\lambda,B,r)_D$	$(\lambda,A,r)_D$	$(\lambda,r,\lambda)_D$	$r : ABC \to \lambda$

5. Abschließende Bemerkungen

Wir haben in diesem Artikel dargestellt, wie verschiedene Varianten von speziellen Geffert-Normalformen geeignet sind, stärkere Beschreibungskomplexitätsresultate (zur Berechnungsvollständigkeit) für (regulierte) ELS zu erzielen. Es ist dabei manchmal auch hilfreich, mit den Codierungen von Binärwörtern zu „spielen“, so wie das letztlich auch Geffert schon gemacht hat. Die separierende Eigenschaft von ssSGNF wurde so erreicht. Diese Vorgehensweise ist auch vorteilhaft bei der Erzielung von Berechnungsvollständigkeitsresultaten für klassischere regulierte Ersetzungssysteme. Hier möchten wir nur auf [10] als eine unserer aktuellen Arbeiten zu dem Thema verweisen.

Die Meisten der erwähnten Ergebnisse sind in Zusammenarbeit mit Indhumathi Raman und Lakshmanan Kuppusamy entstanden. Teilweise wurden diese Forschungen dankenswerterweise durch den DAAD bzw. DST gefördert.

Literatur

[1] A. ALHAZOV, R. FREUND, S. IVANOV, S. VERLAN, Regulated Insertion-Deletion Systems. *Journal of Automata, Languages and Combinatorics* **27** (2022) 1-3, 15–45.

[2] A. ALHAZOV, S. IVANOV, S. VERLAN, A 15-Year Retrospective on Insertion-Deletion Systems: Progress, Evolution, and Future Directions. *Computer Science Journal of Moldova* **32** (2024) 3(96), 332–371.

[3] H. FERNAU, L. KUPPUSAMY, I. RAMAN, On the computational completeness of graph-controlled insertion-deletion systems with binary sizes. *Theoretical Computer Science* **682** (2017), 100–121. Special Issue on Languages and Combinatorics in Theory and Nature.

[4] H. FERNAU, L. KUPPUSAMY, I. RAMAN, On the Generative Power of Graph-Controlled Insertion-Deletion Systems with Small Sizes. *Journal of Automata, Languages and Combinatorics* **22** (2017), 61–92.

[5] H. FERNAU, L. KUPPUSAMY, I. RAMAN, Computational completeness of simple semi-conditional insertion-deletion systems of degree $(2, 1)$. *Natural Computing* **18** (2019) 3, 563–577.

[6] H. FERNAU, L. KUPPUSAMY, I. RAMAN, On path-controlled insertion-deletion systems. *Acta Informatica* **56** (2019) 1, 35–59.

[7] H. FERNAU, L. KUPPUSAMY, I. RAMAN, When Stars Control a Grammar's Work. In: Z. GAZDAG, S. IVÁN, G. KOVÁSZNAI (eds.), *Proceedings of the 16th International Conference on Automata and Formal Languages, AFL*. EPTCS 386, Open Publishing Association, 2023, 96–111.

[8] H. FERNAU, L. KUPPUSAMY, I. RAMAN, On Time-Varying Insertion-Deletion Systems. In: E. FORMENTI, L. MANZONI (eds.), *Unconventional Computation and Natural Computation, UCNC*. LNCS 15xyz, Springer, 2025, xx–xx.

[9] H. FERNAU, L. KUPPUSAMY, I. RAMAN, Succinct Star-Controlled Insertion-Deletion Systems Using Space Separating Normal Forms. In: E. FORMENTI, J. DURAND-LOSE (eds.), *Machines, Computations, and Universality, 10th International Conference, MCU 2024*. LNCS 15270, Springer, 2025, 17–34.

[10] H. FERNAU, L. KUPPUSAMY, I. RAMAN, G. VASZIL, Matrix Forbidding Grammars. In: A. MALCHER, L. PRIGIONIERO (eds.), *Descriptional Complexity of Formal Systems - 26th IFIP WG 1.02 International Conference, DCFS*. LNCS 15759, Springer, 2025, 94–107.

[11] V. GEFFERT, *Problémy zložitosti generatívnych systémov (in Slovak)*. Ph.D. thesis, Katedra teoretickej kybernetiky, Matematicko-fyzikálnej fakulty UK, Bratislava, Czechoslovakia, 1987.

[12] V. GEFFERT, Normal forms for phrase-structure grammars. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications* **25** (1991), 473–498.

[13] L. KARI, *On insertions and deletions in formal languages*. Ph.D. thesis, University of Turku, Finland, 1991.

[14] GH. PĂUN, *Membrane Computing. An Introduction*. Springer, 2002.

[15] A. TAKAHARA, T. YOKOMORI, On the computational power of insertion-deletion systems. *Natural Computing* **2** (2003) 4, 321–336.

THEORIE-AG 2025 B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 35–36.

Revisiting Weighted Information Extraction: A Simpler and Faster Algorithm for Ranked Enumeration

Paweł Gawrychowski[(A)] Florin Manea[(B)] Markus L. Schmid[(C)]

[(A)]University of Wrocław, Wrocław, Poland
gawry@cs.uni.wroc.pl

[(B)]Computer Science Department and CIDAS, Universität Göttingen, Göttingen, Germany
florin.manea@cs.uni-goettingen.de

[(C)]Humboldt-Universität zu Berlin, Berlin, Germany
mlschmid@mlschmid.de

The information extraction from textual data, where the query is represented by a finite transducer and the task is to enumerate all results without repetition, and its extension to the weighted case, where each output element has a weight and the output elements are to be enumerated sorted by their weights, are important and well studied problems in database theory. On the one hand, the first framework already covers the well-known case of regular document spanners, while the latter setting covers several practically relevant tasks that cannot be described in the unweighted setting.

It is known that in the unweighted case this problem can be solved with linear time preprocessing $O(|D|)$ and output-linear delay $O(|s|)$ in data complexity, where D is the input data and s is the current output element. For the weighted case, Bourhis, Grez, Jachiet, and Riveros [1] recently designed an algorithm with linear time preprocessing, but the delay of $O(|s| \cdot \log|D|)$ depends on the size of the data.

We first show how to leverage the existing results on enumerating shortest paths to obtain a simple alternative algorithm with linear preprocessing and a delay of $O(|s_i| + \min\{\log i, \log|D|\})$ for the i^{th} output element s_i (in data complexity); thus, substantially improving the previous algorithm. Next, we develop a technically involved rounding technique that allows us to devise an algorithm with linear time preprocessing and output-linear delay $O(|s|)$ with high probability. To this end, we combine tools from algebra, high-dimensional geometry, and linear programming.

This presentation is based on the paper [2], presented at PODS 2025.

References

[1] P. Bourhis, A. Grez, L. Jachiet, C. Riveros, Ranked Enumeration of MSO Logic on Words. In: K. Yi, Z. Wei (eds.), *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus*. LIPIcs 186, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 20:1–20:19.
https://doi.org/10.4230/LIPIcs.ICDT.2021.20

[2] P. GAWRYCHOWSKI, F. MANEA, M. L. SCHMID, Revisiting Weighted Information Extraction: A Simpler and Faster Algorithm for Ranked Enumeration. *Proc. ACM Manag. Data* **2** (2024) 5, 222:1–222:19.
https://doi.org/10.1145/3695840

THEORIE-AG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 37–39.

On Language Families with a Decidable Pumping-Problem

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

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

[(B)]School of Electrical Engineering and Computer Science
University of Kassel, 34121 Kassel, Germany
christian.rauch@uni-kassel.de

Pumping lemmata are fundamental tools for proving that certain languages do not belong to a specific formal language family, such as regular or context-free languages. Mostly, they provide necessary conditions that all languages within a language family must satisfy, making them essential for formal language analysis and classification. Recently, the PUMPING-PROBLEM was initiated in [7]. 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. Although all context-free languages satisfy the Bar-Hillel pumping lemma [4, page 154, Theorem 4.1], the corresponding PUMPING-PROBLEM was shown to be undecidable in [8], which also holds true for linear context-free languages and their respective pumping lemma [9]. This is in contrast to the decidability of the PUMPING-PROBLEM for regular languages regardless whether a regular [11], linear context-free, or the Bar-Hillel lemma is used to check for the value p. Thus, the question arises, for which language families and pumping lemmata the corresponding PUMPING-PROBLEM becomes decidable. We identify two language families that obey this property, namely the k-rated linear languages and the well-matched visibly pushdown languages (wm-VPL):

- The k-rated linear languages were introduced in [3] as a generalization of even linear languages [2]. A linear grammar is even linear if every production is of the form $A \to u$ or $A \to vBw$ with $|v| = |w|$. These languages lie strictly between regular and general linear context-free languages and admit an algebraic characterization *via* congruences, similar to regular languages [2]. In k-rated linear grammars, the ratio $|v|/|w|$ is fixed to a non-negative rational k for all productions involving nonterminals. A pumping lemma that fits to k-rated linear languages is given in [10] and reads as follows:

  Lemma 1 *Let L be a k-rated linear language with $k = g/h$ over Σ.*

  1. *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 $z = uvwxy$, $0 < |u|, |v| \leq p\frac{g}{g+h}$, $0 < |x|, |y| \leq p\frac{h}{g+h}$, $\frac{|u|}{|y|} = \frac{|v|}{|x|} = \frac{g}{h} = k$, and $uv^twx^ty \in L$ for all $t \geq 0$.*

This is a summary of a paper presented at the 29th International Conference on Implementation and Application of Automata (CIAA) held in Palermo, Italy, September 22–25, 2025.

2. *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 $z = uvwxy$, $0 < |v| \leq p\frac{g}{g+h}$, $0 < |x| \leq p\frac{h}{g+h}$, $0 < |w| \leq p$, $\frac{|u|}{|y|} = \frac{|v|}{|x|} = \frac{g}{h} = k$, and $uv^twx^ty \in L$ for all $t \geq 0$.*

For $k = 1$, we recover the even linear languages. While a general algebraic characterization of k-rated linear languages exists [3], it is non-constructive.

- The well-matched visibly pushdown languages form a subfamily of visibly pushdown languages (VPL), where push and pop operations are balanced. VPLs were formalized in [1] as those languages accepted by visibly pushdown automata. A visibly pushdown automata is a deterministic or nondeterministic pushdown automata, where the input alphabet Σ is partitioned into $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$, where the symbols symbols from Σ_c cause a push, symbols from Σ_r trigger a pop, and symbols from Σ_i leave the pushdown unchanged. They extend regular languages while preserving many of their closure properties and support nested structures, such as those in recursive programs or XML documents. Since well-matched VPL are a special case of VPL, which in turn are context-free languages, these language families must satisfy the Bar-Hillel pumping lemma, which reads as follows:

  Lemma 2 *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 all $t \geq 0$—it is then said that v and x can be (simultaneously)* pumped *in z.*

  Well-matched VPLs and variants have recently been characterized algebraically using Ext-algebras [5, 6].

For both k-rated linear languages and well-matched visibly pushdown languages, we prove that the PUMPING-PROBLEM—with respect to the above given specific pumping lemma variants—is decidable. In case of k-rated linear languages we obtain the result:

Theorem 1 *Let G be a k-rated linear grammar and p a natural number. Then it is decidable whether for the language $L(G)$ the statement of Lemma 1.1 holds w.r.t. the value p. The decidability holds true also in case Lemma 1.2 is considered instead.*

For the well-matched VPL languages, we find the following situation, where the abbreviation DVPDA refers to a *deterministic visibly pushdown automaton* that accepts well-matched words only.

Theorem 2 *Let A be a DVPDA and p a natural number. It is decidable whether for the language $L = L(A)$ the statement of Lemma 2 holds for the value p.*

Our approach to prove these results relies on the algebraic characterization of each family: for k-rated linear languages, we build on the congruence-based characterization of even linear languages [2], enabling a constructive automata-theoretic view. For well-matched VPLs, we use the Ext-algebra framework of [5, 6], which builds on forest algebra techniques.

References

[1] R. ALUR, P. MADHUSUDAN, Visibly pushdown languages. In: *Proceedings of the* 36*th Annual ACM Symposium on Theory of Computing*. ACM, Chicago, IL, USA, 2004, 202–211.

[2] V. AMAR, G. PUTZOLU, On a Family of Linear Grammars. *Information and Control* **7** (1964) 3, 283–291.

[3] V. AMAR, G. PUTZOLU, Generalizations of Regular Events. *Information and Control* **8** (1965) 1, 56–63.

[4] Y. BAR-HILLEL, M. PERLES, E. SHAMIR, On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung* **14** (1961), 143–177.

[5] S. GÖLLER, N. GROSSHANS, The AC^0-Complexity Of Visibly Pushdown Languages. arXiv:2302.13116v3, 2023.

[6] S. GÖLLER, N. GROSSHANS, The AC^0-Complexity Of Visibly Pushdown Languages. In: O. BEYERSDORFF, M. M. KANTÉ, O. KUPFERMAN, D. LOKSHTANOV (eds.), *Proceedings of the* 41*st International Symposium on Theoretical Aspects of Compter Science*. Leibniz International Proceedings in Informatics 289, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, Clermont-Ferrand, France, 2024, 38:1–38:18.

[7] H. GRUBER, M. HOLZER, C. RAUCH, The Pumping Lemma for Regular Languages is Hard. In: B. NAGY (ed.), *Proceedings of the* 27*th International Conference on Implementation and Application of Automata*. Number 14151 in LNCS, Springer, Famagusta, Cyprus, 2023, 128–140.

[8] H. GRUBER, M. HOLZER, C. RAUCH, The Pumping Lemma for Context-Free Languages is Undecidable. In: J. D. DAY, F. MANEA (eds.), *Proceedings of the* 28*th International Conference on Developments in Language Theory*. Number 14791 in LNCS, Springer, Göttingen, Germany, 2024, 141–155.

[9] J. E. HOPCROFT, J. D. ULLMAN, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[10] G. HORVÁTH, B. NAGY, Pumping lemmas for linear and nonlinear context-free languages. *Acta Universitatis Sapientiae - Informatica* **2** (2010) 2, 194–209.

[11] D. C. KOZEN, *Automata and Computability*. Undergraduate Texts in Computer Science, Springer, 1997.

THEORIE-AG 2025 B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 41–45.

NoPE: The Counting Power of Transformers with No Positional Encodings (Extended Abstract)

Chris Köcher[(A)] Alexander Kozachinskiy[(B)]
Anthony Widjaja Lin[(A,C)] Marco Sälzer[(C)] Georg Zetzsche[(A)]

[(A)]Max Planck Institute for Software Systems, Kaiserslautern, Germany
{ckoecher,awlin,georg}@mpi-sws.org

[(B)]Centro Nacional de Inteligencia Artificial, Santiago, Chile
alexander.kozachinskyi@cenia.cl

[(C)]Rheinland-Pfälzische Technische Universität, Kaiserslautern, Germany
marco.saelzer@rptu.de

Transformers [26] have emerged in recent years as a powerful model with a plethora of successful applications including (among others) natural language processing, computer vision, and speech recognition. Despite the success of transformers, the question of what transformers can express is still not well-understood and has in recent years featured in a rich body of research works (e.g. [25, 9, 21, 11]).

Formal language theory has proven to be extremely useful in understanding such expressiveness issues (e.g. see [25]). More precisely, a transformer T is said to express a formal language L (i.e. a set of strings over an alphabet Σ), when T can classify those input strings that are in L and those input strings that are not in L. One class of formal languages that have recently featured in the study of expressiveness of neural networks — in particular, Recurrent Neural Networks (RNN) and Transformers — are the so-called *counting languages*, e.g., see [2, 9, 10, 6, 28, 3, 7, 19, 27, 25]. Intuitively, counting languages (a.k.a. counter languages) require counting the numbers of occurrences of certain characters in the input string and, perhaps, additionally compare them. Of special interests are *permutation-invariant* (counting) languages, i.e., languages that are closed under shuffling the positions of the letters in the string (e.g. $\mathtt{aba} \in L$ implies $\mathtt{baa} \in L$). An example is the language MAJ over the alphabet $\{\mathtt{a}, \mathtt{b}\}$ consisting of strings with more a's than b's (e.g. $\mathtt{aab} \in$ MAJ but $\mathtt{abb} \notin$ MAJ). Another example is the language PARITY consisting of all strings over the alphabet $\{\mathtt{a}, \mathtt{b}\}$ with an even number of occurrences of a. A simple application[1] of counting is sentiment analysis, where the number of positive words should exceed the number of negative words in a text.

Which counting languages can transformers express? Answering this question depends on two crucial parameters: (1) the type of attention mechanism (2) the type of *Positional Encodings (PEs)*. Most theoretical research results (see the survey [25]) focus on *hard-attention mechanisms*, which replace softmax by picking the leftmost position with the maximum attention score (i.e. *unique hard attention*) or averaging those (not necessarily leftmost) positions (i.e *average hard attention*, a.k.a., saturated attention). In the sequel, we will write UHAT (resp.

[1] https://medium.com/data-science/sentiment-analysis-with-text-mining-13dd2b33de27

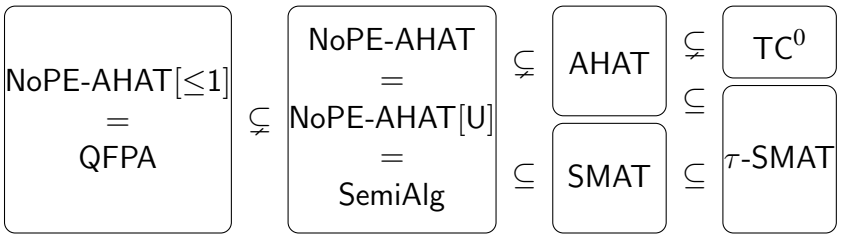


Figure 1: Visualization of our results. The inclusion AHAT $\subseteq$ τ-SMAT was shown in [29].

AHAT) for Unique (resp. Average) Hard Attention Transformers. AHATs have been argued to be a realistic approximation of how transformers function in practice [17]. In this paper, we primarily focus on AHATs, though we will also discuss implications on soft attention transformers. We now proceed to PEs. Attention is an operation that aggregates an input sequence via a weighted sum, which treats the elements in the sequence as a *bag* (i.e. permutation invariant). PEs — which essentially annotate elements in the sequence by some positional information like i and $\sin(2\pi \cdot \frac{i}{n})$ (for positive integers i and n) — are often used to recover the ordering of the elements in the sequence. PEs are, however, known to substantially increase the expressive power of transformers in theory since essentially all computable PEs of the form $p : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^d$ (or $p : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}^d$) are permitted[2]. This has led some researchers to use *positional masking* (a.k.a. causal attention) [28, 24] — which essentially filters elements in the sequence (relative to the current position) before applying attention — instead of PEs.

Several recent results shed some lights on the expressivity of transformers on counting languages. Firstly, UHATs (even with PEs) are known to be strictly contained in the (nonuniform) circuit complexity class AC^0 [11, 2], which characterizes the "inability of counting", e.g., the most basic counting language PARITY is not in AC^0 [1]. In fact, NoPE-UHATs with positional masking can express only star-free regular languages [28]. What about AHATs? We know that AHAT languages (with PEs) are subsumed in the circuit complexity class $\mathsf{TC}^0 \supseteq \mathsf{AC}^0$ [11, 18], which essentially enriches AC^0 circuits with "counting" and "arithmetics". Whether AHAT languages (with PEs) are strictly contained in TC^0 was posed an open problem [2]. Furthermore, AHAT (with PEs) can express all counting languages corresponding to the permutation closures of all regular languages [2]. The question of precisely which counting languages AHATs can express remains open. Furthermore, since some of these constructions employed non-trivial PEs (e.g. trigonometry functions), the question arises as well on the role of PEs in recognizing these languages.

Contributions. Our main contribution[3] is to show that NoPE-AHATs are extremely expressive, in contrast to the case of UHATs [28]: they can express nonnegative integer solution sets to multivariate polynomial equations (i.e. *Diophantine equations*), which play an important role in mathematics (particularly, number theory, and algebraic geometry) and theoretical computer science (particularly, computability theory [16]). A simple example of such a permutation-

[2]Some results even refrain from any computability assumption since commonly used PEs (e.g. trigonometry functions) are not rational, e.g., [2]

[3]For a full version of our contribution including all proofs see [14].

invariant language is

$$\textsc{Sqrt} = \{w \in \{\mathtt{a},\mathtt{b}\}^* \mid |w|_{\mathtt{a}} < |w|/\sqrt{2}\}, \tag{1}$$

containing words whose proportion of the positions with letter a is at most $\sqrt{2}$. More precisely, we provide a *precise characterization* of languages expressible in NoPE-AHAT: finite unions of sets of non-negative integer solutions to multivariate polynomial inequalities, which we call *semi-algebraic sets* (henceforth, written SemiAlg). To the best of our knowledge, this is the first result showing that a sequential model based on neural networks (including RNN and transformers) could represent all Diophantine equations. Our main result and its consequences (more below) are summarized in Figure 1.

Key consequences of our main result: 1. Counting power of NoPE-AHATs in relation to other established models. 2. Undecidability of reasoning about transformers 3. Counting power of soft attention transformers

Firstly, NoPE-AHAT can express counting languages (e.g. SQRT) that cannot be expressed by established models in the literature of transformers (e.g. UHAT with PEs), circuit complexity (AC^0), and formal languages and concurrency theory (e.g. simplified counter machines, Petri nets, and models of higher-order recursion). In particular, simplified counter machines — frequently employed in the investigation of the counting power of RNN and Transformers [19, 27] — cannot recognize permutation-invariant languages beyond *linear integer arithmetics* (a.k.a. semilinear sets [20], meaning those definable in existential Presburger arithmetic). Interestingly, it follows from our characterization that PEs are indispensable for capturing $\textsc{Parity} \notin \mathsf{SemiAlg}$ using AHAT.

Secondly, we can apply our result to show undecidability of formally *verifying* (a.k.a. *interpreting* or *checking robustness*) of transformers, which has recently received attention (e.g. see [22]). For less formal approaches to verification, cf. [12, 23, 4, 8]. [More generally, formal verification of neural networks (including feed-forward neural networks and RNN) is an established research field (cf. [23, 13, 15]) with an annual solver competition (cf. [5]).] Especially, our result implies undecidability of verifying a very simple transformer model with no PEs, left as an open question in [22].

For these aforementioned results, we provide a detailed *parameteric analysis* in terms of the number of layers. In particular, with one layer, NoPE-AHAT expresses only linear Diophantine equations (more precisely, *quantifier-free Presburger Arithmetic (QFPA)*, and so cannot define SQRT. To achieve powerful counting ability (e.g. resulting in undecidability), we show sufficiency of NoPE-AHATs with only two layers.

Finally, it was recently shown [29] that soft attention transformers with either PEs or temperature scaling (written τ-SMAT) can simulate AHAT languages. Our construction of NoPE-AHATs from SemiAlg in fact satisfies a stronger condition: only uniform layers are needed (U) showing that NoPE-AHATs can be simulated even by SMAT without temperature scaling. As a result, τ-SMAT can also express counting languages corresponding to solutions to Diophantine equations.

Separation from TC^0. To complement our above results, we use an *information-theoretic argument* to exhibit a permutation-invariant language that is not expressible by AHAT even in the presence of PEs. In particular, we show that this implies that AHAT-definable languages are a *strict* subset of TC^0, answering an open problem from [2].

References

[1] M. AJTAI, Σ^1_1-Formulae on finite structures. *Annals of Pure and Applied Logic* **24** (1983) 1, 1–48.

[2] P. BARCELÓ, A. KOZACHINSKIY, A. W. LIN, V. V. PODOLSKII, Logical Languages Accepted by Transformer Encoders with Hard Attention. In: *ICLR 2024*. OpenReview.net, 2024.

[3] S. BHATTAMISHRA, K. AHUJA, N. GOYAL, On the Ability and Limitations of Transformers to Recognize Formal Languages. In: *EMNLP 2020*. Association for Computational Linguistics, 2020, 7096–7116.

[4] G. BONAERT, D. I. DIMITROV, M. BAADER, M. VECHEV, Fast and precise certification of transformers. In: *PLDI 2021*. PLDI 2021, Association for Computing Machinery, 2021, 466–481.

[5] C. BRIX, S. BAK, T. T. JOHNSON, H. WU, The Fifth International Verification of Neural Networks Competition (VNN-COMP 2024): Summary and Results. *CoRR* **abs/2412.19985** (2024).

[6] D. CHIANG, P. CHOLAK, Overcoming a Theoretical Limitation of Self-Attention. In: *ACL 2022*. Association for Computational Linguistics, 2022, 7654–7664.

[7] G. DELÉTANG, A. RUOSS, J. GRAU-MOYA, T. GENEWEIN, L. K. WENLIANG, E. CATT, C. CUNDY, M. HUTTER, S. LEGG, J. VENESS, P. A. ORTEGA, Neural Networks and the Chomsky Hierarchy. In: *ICLR 2023*. OpenReview.net, 2023.

[8] X. DONG, A. T. LUU, R. JI, H. LIU, Towards Robustness Against Natural Language Word Substitutions. In: *ICLR 2021*. OpenReview.net, 2021.

[9] M. HAHN, Theoretical Limitations of Self-Attention in Neural Sequence Models. *TACL* **8** (2020), 156–171.

[10] M. HAHN, M. ROFIN, Why are Sensitive Functions Hard for Transformers? In: *ACL 2024*. Association for Computational Linguistics, 2024, 14973–15008.

[11] Y. HAO, D. ANGLUIN, R. FRANK, Formal Language Recognition by Hard Attention Transformers: Perspectives from Circuit Complexity. *TACL* **10** (2022), 800–810.

[12] Y. HSIEH, M. CHENG, D. JUAN, W. WEI, W. HSU, C. HSIEH, On the Robustness of Self-Attentive Models. In: *ACL 2019*. Association for Computational Linguistics, 2019, 1520–1529.

[13] X. HUANG, W. RUAN, W. HUANG, G. JIN, Y. DONG, C. WU, S. BENSALEM, R. MU, Y. QI, X. ZHAO, K. CAI, Y. ZHANG, S. WU, P. XU, D. WU, A. FREITAS, M. A. MUSTAFA, A Survey of Safety and Trustworthiness of Large Language Models through the Lens of Verification and Validation. *CoRR* **abs/2305.11391** (2023).

[14] C. KÖCHER, A. KOZACHINSKIY, A. W. LIN, M. SÄLZER, G. ZETZSCHE, NoPE: The Counting Power of Transformers with No Positional Encodings (2025).

[15] J. MARQUES-SILVA, A. IGNATIEV, Delivering Trustworthy AI through Formal XAI. In: *AAAI 2022*. AAAI Press, 2022, 12342–12350.

[16] Y. V. MATIYASEVICH, *Hilbert's Tenth Problem*. MIT Press, Cambridge, Massachusetts, 1993.

[17] W. MERRILL, V. RAMANUJAN, Y. GOLDBERG, R. SCHWARTZ, N. A. SMITH, Effects of Parameter Norm Growth During Transformer Training: Inductive Bias from Gradient Descent. In: *EMNLP 2021*. Association for Computational Linguistics, 2021, 1766–1781.

[18] W. MERRILL, A. SABHARWAL, N. A. SMITH, Saturated Transformers are Constant-Depth Threshold Circuits. *TACL* **10** (2022), 843–856.

[19] W. MERRILL, G. WEISS, Y. GOLDBERG, R. SCHWARTZ, N. A. SMITH, E. YAHAV, A Formal Hierarchy of RNN Architectures. In: *ACL 2020*. Association for Computational Linguistics, 2020, 443–459.

[20] R. PARIKH, On Context-Free Languages. *JACM* **13** (1966) 4, 570–581.

[21] J. PÉREZ, P. BARCELÓ, J. MARINKOVIC, Attention is Turing-Complete. *JMLR* **22** (2021), 75:1–75:35.

[22] M. SÄLZER, E. ALSMANN, M. LANGE, Transformer Encoder Satisfiability: Complexity and Impact on Formal Reasoning. In: *ICLR 2025*. OpenReview.net, 2025.

[23] Z. SHI, H. ZHANG, K. CHANG, M. HUANG, C. HSIEH, Robustness Verification for Transformers. In: *ICLR 2020*. OpenReview.net, 2020.

[24] L. STROBL, D. ANGLUIN, D. CHIANG, J. RAWSKI, A. SABHARWAL, Transformers as Transducers. *TACL* **13** (2025), 200–219.

[25] L. STROBL, W. MERRILL, G. WEISS, D. CHIANG, D. ANGLUIN, What Formal Languages Can Transformers Express? A Survey. *TACL* **12** (2024), 543–561.

[26] A. VASWANI, N. SHAZEER, N. PARMAR, J. USZKOREIT, L. JONES, A. N. GOMEZ, L. KAISER, I. POLOSUKHIN, Attention is All you Need. In: *NeurIPS 2017*. 2017, 5998–6008.

[27] G. WEISS, Y. GOLDBERG, E. YAHAV, On the Practical Computational Power of Finite Precision RNNs for Language Recognition. In: *ACL 2018*. Association for Computational Linguistics, 2018, 740–745.

[28] A. YANG, D. CHIANG, D. ANGLUIN, Masked Hard-Attention Transformers Recognize Exactly the Star-Free Languages. In: *NeurIPS 2024*. 37, Curran Associates, Inc., 2024, 10202–10235.

[29] A. YANG, L. STROBL, D. CHIANG, D. ANGLUIN, Simulating Hard Attention Using Soft Attention. *CoRR* **abs/2412.09925** (2024).

THEORIE-TAG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 47–50.

Idefix-abgeschlossene Sprachen und ihre Verwendung in kontextualen Grammatiken

Marvin Ködding[(A)] Bianca Truthe[(B)]

[(A)]Institut für Mathematik und Informatik, Pädagogische Hochschule Heidelberg
Im Neuenheimer Feld 561, 69120 Heidelberg, Germany
koedding@ph-heidelberg.de

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

Zusammenfassung

In dieser Arbeit setzen wir die Untersuchung der Ausdrucksstärke kontextueller Grammatiken mit Auswahlsprachen aus Unterklassen der regulären Sprachen fort. Im Fokus stehen infix-, präfix- und suffix-abgeschlossene Sprachen (im Folgenden als *idefix-abgeschlossene Sprachen* bezeichnet), die wir mit weiteren subregulären Sprachfamilien vergleichen. AnschlieSSend untersuchen wir die für externe und interne kontextuelle Grammatiken entstehenden Hierarchien mit den Sprachfamilien, die sich aus diesen neuen Auswahlsprachtypen ergeben. Auf diese Weise erweitern wir bestehende Hierarchien um bislang nicht berücksichtigte Sprachfamilien. Zudem wird ein offenes Problem im Zusammenhang mit internen kontextuellen Grammatiken und suffix-abgeschlossenen Auswahlsprachen gelöst.

1. Einleitung

Kontextuelle Grammatiken, erstmals von Solomon Marcus [5] eingeführt, bieten einen formalen Rahmen zur Modellierung der Erzeugung natürlicher Sprachen. Dabei entstehen neue Wörter durch das Anfügen geordneter Wortpaare (u, v) – sogenannter Kontexte – an bestehende Wörter. Die Anwendung eines Kontextes kann *extern* erfolgen, indem (u, v) an den Anfang bzw. das Ende angefügt wird, oder *intern*, indem es in das Wortinnere eingefügt wird. Der Einsatz eines Kontextes ist an eine *Auswahlsprache* gebunden: Ein Kontext darf nur auf ein Wort x angewendet werden, wenn x zu seiner Auswahlsprache gehört. Beschränkt man die Auswahlsprache auf eine vorgegebene Sprachfamilie F, so erhält man kontextuelle Grammatiken mit Auswahl in F.

Erste Untersuchungen zu externen kontextuellen Grammatiken mit regulären Auswahlsprachen stammen von Jürgen Dassow [1] und wurden von Dassow, Manea und Truthe [2, 3] sowohl auf externe als auch interne Varianten ausgedehnt. Diese Arbeiten analysierten den Einfluss verschiedener subregulärer Einschränkungen auf die Auswahlsprache. In der vorliegenden Arbeit erweitern wir diese Hierarchie um sogenannte *idefix-abgeschlossene* subreguläre Sprachfamilien, d. h. präfix-, suffix- oder infix-abgeschlossene Sprachen, und untersuchen die Erzeugungskraft externer und interner kontextueller Grammatiken mit Auswahlsprachen aus diesen Familien. Dabei lösen wir auch ein offenes Problem zu internen kontextuellen Grammatiken mit suffix-abgeschlossenen Auswahlsprachen, das bereits in [7] aufgeworfen wurde.

2. Definitionen

Wir verwenden die üblichen Begriffe aus der Theorie der Automaten und formalen Sprachen (vgl. [6]). Ein *Alphabet* V ist eine endliche Menge von Symbolen; V^* (bzw. V^+) bezeichnet die Menge aller (nichtleeren) Wörter über V, λ das leere Wort. Die Familie der regulären Sprachen wird mit *REG* bezeichnet, jede Unterfamilie davon als *subreguläre Sprachfamilie*.

Für eine Sprache L über V bezeichnen $\mathit{Inf}(L)$, $\mathit{Pre}(L)$ und $\mathit{Suf}(L)$ ihre Infix-, Präfix- bzw. Suffix-Abschlüsse. Die Familien *INF* (infix-abgeschlossen), *PRE* (präfix-abgeschlossen) und *SUF* (suffix-abgeschlossen) fassen wir unter dem Begriff *idefix-abgeschlossene Sprachen* zusammen. Neben diesen betrachten wir weitere subreguläre Sprachfamilien: kombinational (*COMB*), definitiv (*DEF*), symmetrisch definitiv (*SYDEF*), nilpotent (*NIL*), kommutativ (*COMM*), zirkulär (*CIRC*), nicht-zählend bzw. kleenesternfrei ($\mathit{NC} = \mathit{SF}$), vereinigungsfrei (*UF*), endlich (*FIN*), monoidal (*MON*), potenzseparierend (*PS*), geordnet (*ORD*), Sternsprachen (*STAR*) sowie die links-, rechts- und zweiseitigen Kometsprachen (*LCOM*, *RCOM*, *2COM*) (vgl. [4]).

Ein *kontextuelle Grammatik mit Auswahl in* $\mathcal{F}$ ist ein Tripel $G = (V, \mathcal{S}, A)$ mit einem Alphabet V, einer endlichen Menge von Auswahlpaaren (S, C) bestehend aus einer Auswahlsprache $S \in \mathcal{F}$ und einer endlichen Menge von Kontexten $C \subset V^* \times V^*$, sowie einer endlichen Menge von Axiomen $A \subset V^*$. In der *externen* Ableitung wird ein Kontext $(u, v) \in C$ um ein Wort x gelegt, wenn $x \in S$ gilt, in der *internen* Ableitung entsprechend um ein Teilwort x_2 von x. Die von G im externen bzw. internen Modus erzeugten Sprachen bilden die Familien $\mathcal{EC}(\mathcal{F})$ bzw. $\mathcal{IC}(\mathcal{F})$.

3. Ergebnisse

Satz 3.1 *Es gelten die Teilmengenbeziehungen aus den Abbildungen 1, 2 und 3. 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.*

Eine offene Frage aus [7] kann nun auch beantwortet werden.

Satz 3.2 *Die Sprachfamilien $\mathcal{IC}(\mathit{ORD})$ und $\mathcal{IC}(\mathit{NC})$ sind unvergleichbar zu der Sprachfamilie $\mathcal{IC}(\mathit{SUF})$.*

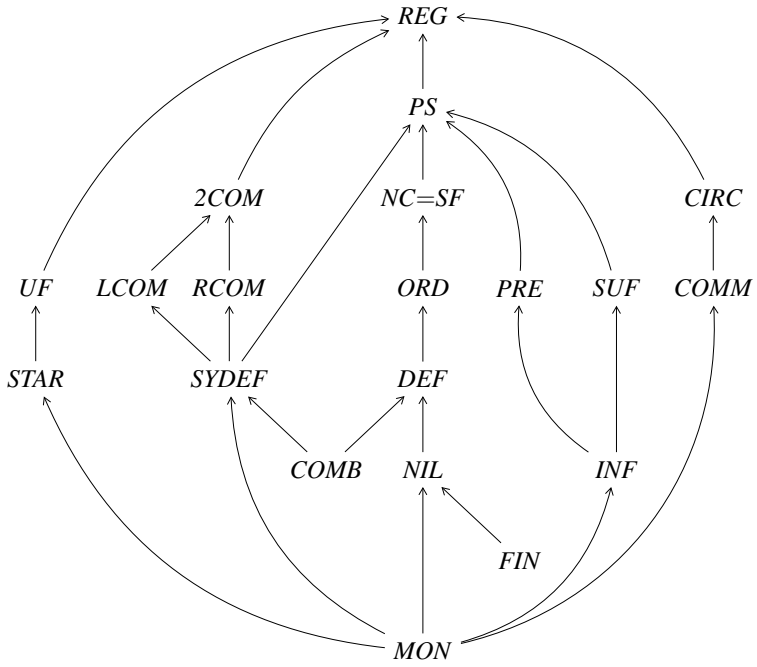


Abbildung 1: Hierarchie von subregulären Sprachfamilien

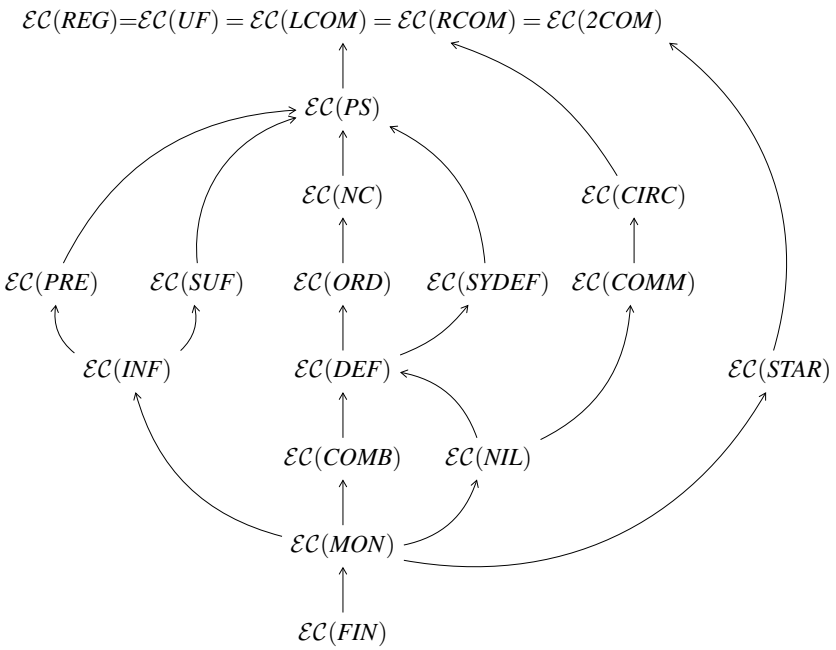


Abbildung 2: Hierarchie von Sprachfamilien als Auswahl für externe kontextuale Grammatiken

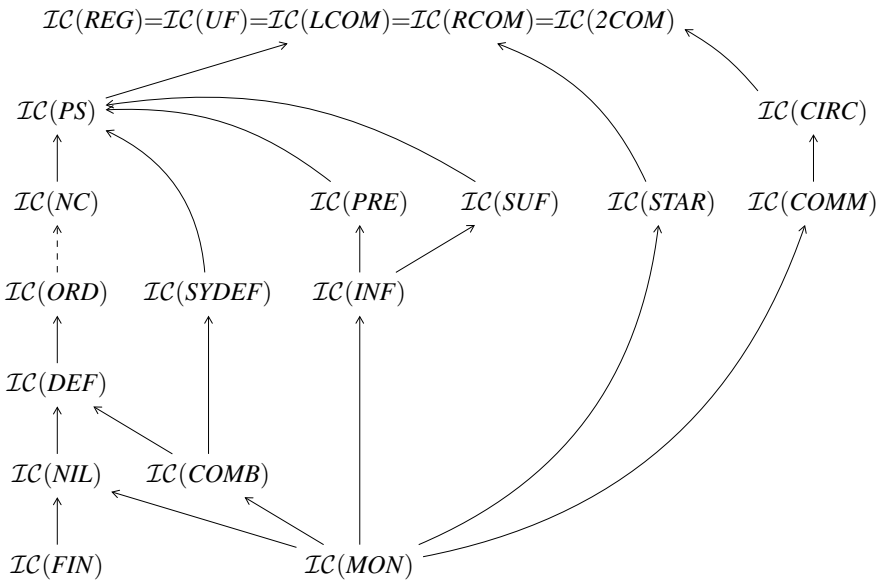


Abbildung 3: Hierarchie von Sprachfamilien als Auswahl für interne kontextuale Grammatiken

Literatur

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

[2] J. DASSOW, F. MANEA, B. TRUTHE, On external contextual grammars with subregular selection languages. *Theoretical Computer Science* **449** (2012), 64–73.

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

[4] M. KÖDDING, B. TRUTHE, Various Types of Comet Languages and Their Application in Contextual Grammars. *Journal of Automata, Languages, and Combinatorics* (submitted).

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

[6] G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

[7] B. TRUTHE, Hierarchies of Language Families of Contextual Grammars. In: R. FREUND, F. MRÁZ, D. PRŮŠA (eds.), *Nineth Workshop on Non-Classical Models of Automata and Applications (NCMA), Prague, Czech Republic, August 17–18, 2017, Proceedings*. books@ocg.at 329, Österreichische Computer Gesellschaft, 2017, 13–28.

THEORIE-AG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 51–52.

SMTQuery: A Novel Tool for Analysing SMT-LIB String Benchmarks

Mitja Kulczynski[(A)] Kevin Lotz[(A)] Florin Manea[(B)]
Danny Bøgsted Poulsen[(C)] Paul Sarnighausen-Cahn[(B)]

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

[(B)]Department of Computer Science, University of Göttingen, Göttingen, Germany
{florin.manea,paul.cahn}@cs.uni-goettingen.de

[(C)]Department of Computer Science, Aalborg University, Aalborg, Denmark
dannybpoulsen@cs.aau.dk

Abstract

Constraint satisfaction problems involving strings have been a subject of theoretical study for decades, but the recent years have seen an increased interest in the development of practical solving methods. This interest in solving string constraints led to the development of various techniques and solvers, often accompanied by specific benchmark sets. As a result, there is now a substantial corpus of publicly available, yet largely unclassified, such benchmarks. In this context, we present SMTQuery, a framework for maintaining and analyzing benchmarks for SMT string problems. SMTQuery enables the execution of user-defined queries to extract domain-specific information from these benchmarks, facilitating a deeper analysis of the underlying problems. We demonstrate its utility by analyzing over 100,000 benchmarks and training an algorithm selection model to match benchmarks with suitable solvers.

1. Overview

We introduce smtquery, a framework designed for storing, analyzing, and querying extensive collections of SMT-LIB benchmarks. SMTQuery stores structural properties of formulas and outcomes from user-defined solvers. Users submit *queries* to filter benchmarks based on specific predicates, which can examine structural properties of formulas or solver outcomes. The system compiles these benchmarks into user-chosen formats. The system supports combining multiple predicates using Boolean logic for versatile information retrieval. All queries are formulated in $qlang$, a query language inspired by SQL but tailored to address the unique requirements of the system, developed specifically for smtquery.

1.1. Maintaining Benchmarks.

SMTQuery processes collections of SMT-LIB benchmarks by registering each instance in a database. If configured, it executes a predefined set of solvers on each instance, storing

the results alongside the benchmark in the database. Simultaneously, SMTQuery constructs an internal data structure capturing the logical structure of the SMT formula and additional derived information. Thus, the storage layer comprises two primary components: a database recording all processed benchmarks along with metrics of the executed solvers, and an internal representation of each benchmark's logical formula, enhanced with metadata. The latter is maintained in the form of an *abstract syntax tree* (*AST*) that can be dynamically augmented with additional data. Each node in the AST is annotated with an $inteldict$, a key-value store containing arbitrary metadata about the node. Initially, when a new instance is presented to the system, its corresponding $inteldict$ is empty. Whenever the answer to a query depends on a node's structural properties, SMTQuery first checks if this information exists in the $inteldict$. If present, the result is retrieved directly; if not, SMTQuery computes the property and stores it in the $inteldict$. This process is explained in more detail below.

1.2. Querying Information.

SMTQuery provides an interface for querying stored information. Each query is structured into two components:

1. a boolean combination of *predicates* $\mathcal{P}$, which filters stored instances, and
2. an *extractor* e, which specifies the desired output format.

SMTQuery selects the set of instances T' that fulfill $\mathcal{P}$. The query is then answered by evaluating the extractor e on the resulting set T'. In its simplest form, extractors return the names of the instances or the instances themselves in SMT-LIB format. More complex extractors can generate plots from the data. SMTQuery provides a rich set of predicates and extractors out of the box, addressing a wide range of use cases. The architecture is flexible enough to be easily extended with more complex operations.

1.3. Use Case

As one of the main examples supporting SMTQuery, we have developed a machine-learning-based *algorithm selector* (e.g., [1]) that predicts the most reliable and efficient solver for an unseen problem based on its structural features. While similar approaches have been proposed for general SMT-solving (e.g., [2]), our focus is on applying this method specifically to the domain of string solving.

References

[1] J. R. RICE, The Algorithm Selection Problem. *Adv. Comput.* **15** (1976), 65–118. https://doi.org/10.1016/S0065-2458(08)60520-3

[2] J. SCOTT, A. NIEMETZ, M. PREINER, S. NEJATI, V. GANESH, Publisher Correction: Algorithm selection for SMT. *Int. J. Softw. Tools Technol. Transf.* **25** (2023) 5, 799–800. https://doi.org/10.1007/s10009-023-00714-1

THEORIE-AG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 53–55.

Disjointness, Inclusion, and Regularity of ω-Rational Trace Languages

Dietrich Kuske

TU Ilmenau
dietrich.kuske@tu-ilmenau.de

Abstract

This paper studies the disjointness, the inclusion, and the regularity problem for nonterminating behaviors of concurrent systems modelled using Mazurkiewicz traces. The main finding is a trichotomy that, for every independence alphabet, determines the complexity of the problems (almost) completely. Noteworthy, for all the problems, the classes are the same. While these classes already appeared in the study of terminating behaviors, the trichotomy obtained here is more pronounced as the complexities vary from decidable via low levels of the arithmetical hierarchy to low levels of the analytical hierarchy (that do not feature in the result for terminating behaviors).

1. Introduction

Free partially commutative monoids were first studied by Cartier and Foata in combinatorics [3]. Later, Mazurkiewicz introduced them as trace monoids into computer science as a means for the description of the behavior of certain concurrent systems [15]. More recently, traces have also been used to model multi-buffer systems [9] and multi-pushdown systems [10, 12]. Graph monoids are a more general notion [13, 14] and are used in valence systems [17, 6] to model systems with multiple partially blind counters and pushdowns. Starting with [7, 8, 4, 5], interest has extended to infinite traces as they allow to model nonterminating behaviors of concurrent systems.

The idea behind traces is that concurrent processes synchronize via shared actions. Then sequences of basic actions, i.e., words, form interleavings of the executions of these processes. Words that are interleavings of one and the same concurrent execution are considered equivalent. Formally, the concurrency in a system is given by an independence alphabet that describes which pairs of actions are executed by disjoints sets of sequential processes and can therefore commute in interleaving executions. As a consequence, questions like "Do two concurrent systems allow the same concurrent executions?" translate into "Does every interleaving execution of one system have an equivalent interleaving execution of the other system?" or "Are the closures of the sets of interleaving executions under the equivalence of words equal?" In other words, one is interested in the closure of a language under this equivalence.

For regular sets of finite words, questions of this type (in particular, disjointness, inclusion, and regularity of closures) have been studied in detail. Here, e.g., non-inclusion asks for the existence of some finite object (a word) with some decidable property (to belong to the first regular

The results of this paper are presented at FCT'25, cf. [11].

set, but not to the closure of the second). Hence inclusion belongs to Π^0_1, the first universal level of the arithmetical hierarchy. Differently, regularity asks for the existence of a finite object (a finite automaton) such that all finite objects (words) satisfy some decidable property (to belong to the closure if, and only if, they are accepted by the automaton). Hence, this problem belongs to Σ^0_2, the second existential level of the arithmetical hierarchy. These easy observations do not say that inclusion or regularity cannot be expressed by simpler means, e.g., are decidable or belong Π^0_1. Using two properties of independence alphabets (transitivity and diagonality), borders between decidability and Π^0_1-hardness have been identified for disjointness [1] (diagonal vs. non-diagonal) and for inclusion [2] and regularity [16] (transitive vs. non-transitive). While disjointness and inclusion belong to Π^0_1, the precise complexity of regularity remained open. Here, this question is settled: regularity is Σ^0_2-complete for every non-transitive independence alphabet. Computationally, this says that neither the regularity nor the non-regularity of the closure of some regular set is semi-decidable.

The main part of this paper deals with nonterminating behaviours, i.e., ω-words and Büchi-automata. To express disjointness of closures in this setting, it does not suffice to talk about finite objects, but one needs to make statements about all ω-words. As a consequence, disjointness, inclusion, and regularity now belong to Π^1_1 and Π^1_2, i.e., the first two universal levels of the analytical hierarchy. The question, again, is whether simpler means suffice to characterise, e.g., those Büchi-automata that accept some ω-language with regular closure. Also in this case, transitivity and diagonality characterise the situation completely: (1) for transitive independence alphabets, disjointness and inclusion are decidable while regularity belongs to Π^0_1, (2) for non-transitive diagonal independence alphabets, disjointness belongs to Π^0_1, inclusion is complete for Π^0_1, and regularity is complete for Σ^0_2, and (3) for non-diagonal independence alphabets, disjointness is complete for Π^1_1 and inclusion and regularity both are hard for Π^1_1. Thus, diagonal independence alphabets allow to express disjointness etc. refering to finite objects only while this is not possible for any non-diagonal independence alphabet. Furthermore, transitive and diagonal independence alphabets behave much the same in the finite and the infinite setting (it is still open whether regularity for the transitive case and disjointness for the diagonal case are decidable).

References

[1] I. AALBERSBERG, H. HOOGEBOOM, Characterizations of the decidability of some problems for regular trace languages. *Mathematical Systems Theory* **22** (1989), 1–19.

[2] I. AALBERSBERG, E. WELZL, Trace languages defined by regular string languages. *RAIRO, Inf. ThÃl'or. Appl.* **20** (1986), 103–119.

[3] P. CARTIER, D. FOATA, *Problèmes combinatoires de commutation et réarrangements*. Lecture Notes in Mathematics vol. 85, Springer, Berlin - Heidelberg - New York, 1969.

[4] V. DIEKERT, A. MUSCHOLL, Deterministic Asynchronous Automata for Infinite Traces. In: *STACS'93*. Lecture Notes in Comp. Science vol. 665, Springer, 1993, 617–628.

[5] W. EBINGER, A. MUSCHOLL, Logical definability on infinite traces. In: *ICALP'93*. Lecture Notes in Comp. Science vol. 700, Springer, 1993, 335–346.

[6] M. GANARDI, R. MAJUMDAR, G. ZETZSCHE, The Complexity of Bidirected Reachability in Valence Systems. In: *LICS'22*. ACM, 2022, 26:1–26:15.

[7] P. GASTIN, Infinite Traces. In: *Semantics of Systems of Concurrent Processes*. Lecture Notes in Comp. Science vol. 469, Springer, 1990, 277–308.

[8] P. GASTIN, Recognizable and Rational Languages of Finite and Infinite Traces. In: *STACS'91*. Lecture Notes in Comp. Science vol. 480, Springer, 1991, 89–104.

[9] M. HUTAGALUNG, N. HUNDESHAGEN, D. KUSKE, M. LANGE, É. LOZES, Multi-buffer simulations: decidability and complexity. *Information and Computation* **262** (2018) 2, 280–310.

[10] C. KÖCHER, D. KUSKE, Backwards-Reachability for Cooperating Multi-Pushdown Systems. *Journal of Computer and System Sciences* **148** (2025). Article no. 103601.

[11] D. KUSKE, *Disjointness, inclusion, and regularity of ω-rational trace languages*, 2025. Accepted for FCT'25.

[12] D. KUSKE, The theory of reachability in trace-pushdown systems. In: *CiE'25*. Lecture Notes in Comp. Science. vol. 15764, Springer, 2025, 283–298.

[13] M. LOHREY, B. STEINBERG, The submonoid and rational subset membership problems for graph groups. In: *LATA'07.*. Report 35/07, Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona, 2007, 367–378.

[14] M. LOHREY, G. ZETZSCHE, The Complexity of Knapsack in Graph Groups. In: *STACS'17*. LIPIcs 66, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 52:1–52:14.

[15] A. MAZURKIEWICZ, *Concurrent program schemes and their interpretation*. Technical report, DAIMI Report PB-78, Aarhus University, 1977.

[16] J. SAKAROVITCH, The "last" decision problem for rational trace languages. In: *LATIN'92*. Lecture Notes in Comp. Science vol. 583, Springer, 1992, 460–473.

[17] G. ZETZSCHE, Silent Transitions in Automata with Storage. In: *ICALP 2013*. Lecture Notes in Comp. Science vol. 7966, Springer, 2013, 434–445.

THEORIE-AG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 57–59.

Context-Free Subphrase Grammars

Björn Lötters

Technische Hochschule Mittelhessen, Wiesenstr. 14, 35390 Gießen
bjoern.loetters@mni.thm.de

Abstract

Context-Free Grammars are widely used for syntactic specification, but are also notorious for their lack of modularity. We introduce and study a generalization of *Context-Free Subphrase Grammars*, which extend the usual phrase structure with a subphrase order and provide a domain-theoretic foundation for the modular description of syntax.

1. Introduction

Context-Free Subphrase Grammars (CFSGs) were first introduced in [6] as a grammar formalism that matches ChomskyâĂŹs Context-Free Grammars (CFGs) [1] in expressive power, but addresses their well-known lack of modularity [9, 5, 4]. This is achieved by adding a layer of indirection through the generalization of non-terminal *symbols* to non-terminal *expressions*. These expressions are then interpreted via an additional rule scheme besides production rules, namely *subphrase rules*. They introduce a notion of *subsumption*, as it is also known from type theory and logic (cf. [7]), and give rise to a domain-theoretic interpretation, where non-terminal expressions denote elements of a syntactic Scott domain [8].

However, in earlier formulations a single CFSG could induce multiple such domains in parallel, necessitating an additional well-formedness criterion. In our current work, we generalize CFSGs further by incorporating all such domains into a single universal domain. This yields a grammar formalism that does not only support the modular definition of syntax, but also reflects this modularity semantically: by defining a family of languages rather than a single one.

2. Formalization

Let $G = (\mathbb{T}, \mathbb{V}, \mathbb{P}, \mathbb{S})$ be a Context-Free Subphrase Grammar (CFSG), where $\mathbb{T}$ is the finite set of terminal symbols and $\mathbb{V}$ is the finite set of variable symbols. We define $\mathbb{N} = \mathbb{V} \cup \{\top, \bot\}$ to be the set of non-terminal symbols. The set of non-terminal expressions $\mathbb{E}$ is then given by the smallest set satisfying the conditions (1) $\mathbb{N} \subseteq \mathbb{E}$, (2) if $E \in \mathbb{E}$, then $\downarrow E \in \mathbb{E}$ and (3) if $E_1, E_2 \in \mathbb{E}$, then $E_1 \sqcup E_2 \in \mathbb{E}$ as well as $E_1 \sqcap E_2 \in \mathbb{E}$. Finally, the rules of G are given as the finite sets $\mathbb{P} \subseteq \mathbb{N} \times (\mathbb{T} \cup \mathbb{E})^*$, the production rules, and $\mathbb{S} \subseteq \mathbb{N} \times \mathbb{N}$, the subphrase rules.

While production rules are used to derive strings from non-terminal symbols, subphrase rules operate differently: they give rise to a notion of subsumption, which enables rewrites at the level of non-terminal expressions. For this purpose, we define the *subphrase relation* $\mathbb{S}^*$ as the reflexive and transitive closure of the relation $\mathbb{S} \cup \{(X, \top), (\bot, X) \mid X \in \mathbb{N}\}$ and henceforth

identify any two non-terminal symbols $X, Y \in \mathbb{N}$ iff $(X,Y) \in \mathbb{S}^* \wedge (Y,X) \in \mathbb{S}^*$. Given that $\downarrow S = \{ Y \mid \exists X \in S : (Y,X) \in \mathbb{S}^* \}$ denotes the lower set of a set $S \subseteq \mathbb{N}$, we can now define the *subphrase lattice* as the superalgebraic lower set lattice $\downarrow \mathbb{S}^* = \{ \downarrow S \mid S \subseteq \mathbb{N} \wedge S \neq \varnothing \}$ (cf. [2]), where $\downarrow \{ \top \}$ and $\downarrow \{ \bot \}$ constitute the greatest and the least element, respectively. This subphrase lattice provides us with an appropriate semantics for non-terminal expressions, where the denotation $[\![\cdot]\!] : \mathbb{E} \to \downarrow \mathbb{S}^*$ is inductively defined over $\mathbb{E}$ by: (1) If $X \in \mathbb{N}$, then $[\![X]\!] = \downarrow \{ X \}$, (2) $[\![\downarrow E]\!] = ([\![E]\!] \setminus \mathit{max}([\![E]\!])) \cup [\![\bot]\!]$, (3) $[\![E_1 \sqcup E_2]\!] = [\![E_1]\!] \cup [\![E_2]\!]$ and (4) $[\![E_1 \sqcap E_2]\!] = [\![E_1]\!] \cap [\![E_2]\!]$.

Proposition 2.1 *The language $\mathbb{E}$ is sound and complete with respect to $\downarrow \mathbb{S}^*$.*

With these definitions at hand, we can finally turn our attention to the derivation relation, which splits up in two essential parts. The first of these parts is the usual *application of production rules* $\mathbb{R} \subseteq (\mathbb{T} \cup \mathbb{E})^* \times (\mathbb{T} \cup \mathbb{E})^*$, which is defined as $(\alpha X \beta, \alpha \omega \beta) \in \mathbb{R} \iff (X, \omega) \in \mathbb{P}$. The second of these parts is the *subsumption of non-terminal expressions* $\mathbb{Q} \subseteq (\mathbb{T} \cup \mathbb{E})^* \times (\mathbb{T} \cup \mathbb{E})^*$, which is defined as $(\alpha E_1 \beta, \alpha E_2 \beta) \in \mathbb{Q} \iff [\![E_2]\!] \subseteq [\![E_1]\!]$. The derivation relation $\mathbb{D}^*$ is then simply the reflexive and transitive closure of the union $\mathbb{R} \cup \mathbb{Q}$.

The derivation relation now gives rise to the *family of languages* $(\mathcal{L}^G_E)_{E \in \mathbb{E}}$ generated by G, where $\mathcal{L}^G_E = \{ w \in \mathbb{T}^* \mid (E, w) \in \mathbb{D}^* \}$. In contrast to CFGs, a dedicated start non-terminal symbol is not required by our formalism. This is a reflection of its modularity, as a CFSG can be better understood as an agglomeration of several grammars. Nevertheless, CFSGs are sound and complete with respect to CFGs, which implies the following theorem.

Theorem 2.2 *CFSGs characterize precisely the class of Context-Free Languages (CFLs).*

On top of that, the consideration of all the languages described opens up a new perspective on the order induced by the subphrase lattice. It turns out that the image $\mathit{im}(\mathcal{L}^G)$ of the function $\mathcal{L}^G : \mathbb{E} \to \mathcal{P}(\mathbb{T}^*)$ associated with this language family forms itself a bounded lattice. One of the most important consequences that lead to this fact is the following lemma.

Lemma 2.3 ($\mathcal{L}^G$ is Scott-Continuous) $\mathcal{L}^G_E = \bigcup_{X \in [\![E]\!]} \mathcal{L}^G_X$

That is to say, not only can we represent any non-terminal expression as a finite union of principal ideals, but its semantics is also completely determined by the semantics of the principal ideals, which is only possible since CFLs are closed under union (cf. [3]).

3. Example

In order to demonstrate the modularity of our formalism, we consider the grammar in (1). Here, we write $X \sqsubseteq Y$ for the subphrase rule $(X,Y) \in \mathbb{S}$ and $X \to \omega$ for the production rule $(X, \omega) \in \mathbb{P}$. As we can observe at the rule $T \to T$ `"*"` $\downarrow T$, production rules only depend on the non-terminal symbol on their respective left-hand side in this example. This is possible due to the indirection introduced by the greatest subphrase expression $\downarrow T$, which is equivalent to the non-terminal symbol F in (1). That is to say, an expression such as $\downarrow T$ constitutes an *extension point*, whose semantics depends on the exact subphrase relation in the respective context. As a consequence, if we merge the grammars from (1) and (2) the semantics of $\downarrow T$ change accordingly and without further corrections, leaving it as equivalent to P instead of F.

$$
\begin{array}{l}
E \to E \ \texttt{"+"} \downarrow E \\
T \sqsubseteq E \\
T \to T \ \texttt{"*"} \downarrow T \\
F \sqsubseteq T \\
F \to \texttt{"x"}
\end{array}
\qquad (1) \qquad
\begin{array}{l}
P \sqsubseteq T \\
F \sqsubseteq P \\
P \to \downarrow P \ \texttt{"\^{}"} \ P
\end{array}
\qquad (2)
$$

References

[1] N. CHOMSKY, On certain formal properties of grammars. *Information and Control* **2** (1959) 2, 137–167.

[2] M. ERNÃL', M. GEHRKE, A. PULTR, Complete Congruences on Topologies and Down-set Lattices. *Applied Categorical Structures* **15** (2007), 163–184.

[3] J. E. HOPCROFT, R. MOTWANI, J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006.

[4] A. JOHNSTONE, E. SCOTT, M. VAN DEN BRAND, Modular grammar specification. *Science of Computer Programming* **87** (2014), 23–43.
`https://www.sciencedirect.com/science/article/pii/S0167642313002414`

[5] P. KLINT, R. LÄMMEL, C. VERHOEF, Toward an engineering discipline for grammarware. *ACM Trans. Softw. Eng. Methodol.* **14** (2005) 3, 331âĂŞ380.
`https://doi.org/10.1145/1072997.1073000`

[6] B. LÖTTERS, Context-Free Subphrase Grammars. In: J. HEMANN, S. CHANG (eds.), *Trends in Functional Programming*. Springer Nature Switzerland, Cham, 2025, 112–133.

[7] B. C. PIERCE, *Types and Programming Languages*. MIT Press, 2002.

[8] D. SCOTT, Data Types as Lattices. *SIAM Journal on Computing* **5** (1976) 3, 522–587.

[9] S. WINTNER, Modular Context-Free Grammars. *Grammars* **5** (2002) 1, 41–63.
`https://doi.org/10.1023/A:1014216630973`

THEORIE-TAG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 61–64.

The Pumping Lemma for 1-turn Weakly Accepting Blind Counter Automata is Undecidable

Julia Meusel Klaus Reinhardt

Martin-Luther-Universität Halle-Wittenberg, Institut für Informatik
Von-Seckendorff-Platz 1, 06120 Halle (Saale), Germany
{Klaus.Reinhardt,Julia.Meusel}@informatik.uni-halle.de

Abstract

We show that the PUMPING-PROBLEM is undecidable already for the special case that the pumping number p equals 1 and that the language is given by a counter automaton, which can increment and decrement the counter and accepts in an end state if the counter is positive. The languages accepted by such an automaton are a proper subset of 1-BLIND. This result still holds if the automaton is 1-turn. In that case the recognized languages are a proper subset of both 1-BLIND and LIN.

1. Introduction

Gruber, Holzer and Rauch introduced the PUMPING-PROBLEM in [2], where the objective is to determine whether a language L satisfies a fixed pumping lemma with respect to a given value p. In the same paper, they showed that the PUMPING-PROBLEM for regular languages is hard but decidable and in [3] that it is undecidable for context-free and linear languages. This raises the question of language classes in between. The proof in [3] immediately extends to blind respectively weak counter automata, which accept only if the counter is zero. This is sufficient to recognize an invalid computation of a Turing machine by using the nondeterminism to guess not only the configuration with the incorrect successor configuration, but also the position within that configuration where the error occurs. The counter is only used to identify the same position in the following configuration.

This raises the question about even weaker language classes and leads us to the following:

Definition 1.1 *A weakly accepting blind counter automaton M is a 5-tuple $M = (Z, \Sigma, \delta, q_0, E)$ consisting of a set of states Z, an alphabet Σ, a transition function $\delta : Z \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}_e(Z \times \{-1, 0, 1\})$, a start state q_0 and a set of end states E. A configuration $K = (p, x_i \dots x_n, c) \in Z \times \Sigma^* \times \mathbb{Z}$ consists of the current state, the rest of the word and the current content counter. The successor configuration of K when reading $u \in \Sigma \cup \{\epsilon\}$ is $K' = (q, x_{i+|u|} \dots x_n, c + d)$ for $(q, d) \in \delta(p, u)$. We denote this with the relation $\vdash$. The automaton then accepts the language $L(M) = \{x \in \Sigma^* \mid \exists q \in E : (q_0, x, 0) \vdash^* (q, \epsilon, m)\}$ with $m > 0$.*
We call M 1-turn *if it can only switch between incrementing and decrementing once.*

Obviously, any language accepted by such an automaton is in 1-BLIND. Furthermore they constitute the trio generated by the language $\{w \in \{a,b\}^* \mid |w|_a > |w|_b\}$ (where a can be read as increment operation and b as decrement operation). A trio is the closure under homomorphism, inverse homomorphism and intersection with regular languages[1]. It is a proper subset of 1-BLIND, which is the trio generated by $\{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$. If we furthermore restrict the automaton to 1-turn, its language is also in LIN. This corresponds to the trio generated by the language $\{a^n b^m \mid n > m\} \cup \{b^m a^n \mid n > m\}$.

To show that the PUMPING-PROBLEM for such automata is undecidable, we reduce the halting problem of a 2-counter Minsky machine [5].

Definition 1.2 *A* 2-counter Minsky machine $M = (Z, \delta, q_0, E)$ *is a 4-tuple consisting of a set of states* Z*, a transition function* $\delta : Z \times \{0,1\} \times \{0,1\} \to Z \times \{-1,0,1\} \times \{-1,0,1\}$*, a start state* q_0 *and a set of end states* E*. A configuration* $K = (p, c_1, c_2) \in Z \times \mathbb{Z} \times \mathbb{Z}$ *consists of the current state and the current contents of the counters. The successor configuration of* K *is* $K' = (q, c_1 + d_1, c_2 + d_2)$ *for* $\delta(p, \text{sgn}(c_1), \text{sgn}(c_2)) = (q, d_1, d_2)$*. We denote this with the relation* $\vdash$*. The 2-counter Minsky machine then halts if there is a sequence of configurations* $K_0 = (q_0, 0, 0), K_1 \dots K_e = (q, 0, 0)$ *with* $\forall i \in \{1 \dots e\} K_{i-1} \vdash K_i$*.*

2. Result

We consider a variant of the pumping lemma for context-free languages as stated in [4]. Then, deciding the PUMPING-PROBLEM means determining, for a given language L and integer p, whether there exists a decomposition $z = uvwxy$ for all $z \in L$ with $|z| \geq p$, such that $|vx| \geq 1$, $|vwx| \leq p$ and $uv^i wx^i y \in L$ for all $i \geq 0$.

We consider the special case of $p = 1$. Since $|vwx| \leq p = 1$, this means either $v = \epsilon$ or $x = \epsilon$. Thus, the decomposition of z into five parts is obsolete. For this reason, the pumping lemma for the special case of $p = 1$ just formulates as a decomposition $z = uvw$ into three parts. We then show, that this special case of the PUMPING-PROBLEM is undecidable if the language is given by a 1-turn weakly accepting blind counter automaton.

Theorem 2.1 *Given a language* $L = L(C) \subseteq \Sigma^*$ *by a 1-turn weakly accepting blind counter automaton* C *(and* $p = 1$*), it is undecidable whether, for all* $z \in L$ *with* $|z| \geq 1$*, there exist* $u, v, w \in \Sigma^*$ *such that* $z = uvw$ *and* $|v| = 1$ *such that* $uv^i w \in L$ *for all* $i \geq 0$*.*

Proof. We reduce the undecidable [5] halting problem of a 2-counter Minsky machine M to the PUMPING-PROBLEM. Let $\Sigma = \{a, b\}$. A configuration of M is encoded as $a^{i+1} b^{j+1} a b^{k+1}$ where i is the number of the state, j is the content of the first counter and k is the content of the second counter. To adhere to the construction of [3], let $\mathsf{VAL}(M)$ be the set of all words of the form $w_1 w_2 w_3 \dots w_e$ where w_i is the encoding of the i-th configuration in a valid halting configuration sequence. Then $\mathsf{INVAL}(M)$ is $\Sigma^* \setminus \mathsf{VAL}(M)$. If M does not halt, this means $\mathsf{INVAL}(M) = \Sigma^*$.

We can construct a weakly accepting blind counter automaton that accepts $\mathsf{INVAL})(M)$ as follows: First, we use the nondeterminism to guess the reason for the configuration sequence

being invalid. One or more of the following apply:

1. The format is incorrect, i.e. the encoding is not in $(a^+b^+ab^+)^*$.
2. The first configuration is not the start configuration.
3. The last configuration is not an end configuration.
4. There are configurations K_i and K_{i+1} such that $K_i \nvdash K_{i+1}$ for one of the following reasons:
   (a) The state of K_{i+1} is wrong.
   (b) The content of one of the counters of K_{i+1} is too large.
   (c) The content of one of the counters of K_{i+1} is too small.

Cases 1, 2, 3 and 4a can be checked using only the finite states of the automaton. To check case 4b, we increment the counter for all bs denoting the content of the affected counter in the encoding of K_i and then increment respectively decrement it according to the state transition. At this point, the content of the machine's counter is the value that the affected counter should have in K_{i+1}. Finally, we decrement the counter for all bs that denote the content of the affected counter in the encoding of K_{i+1}. Case 4c is handled analogously by exchanging increment and decrement.

From now on we follow the construction of [3], making some simplifications. We consider the language

$$L_M = h(\Sigma^*)ch(\{a\}^*) \cup h(\mathsf{INVAL}(M))c\Sigma^+ \cup \Sigma^*c \cup c^*$$

where $h : \Sigma^* \to \Sigma^*$ is the homomorphism defined by $h(a) = d^2$ for all $d \in \Sigma$. This language can be recognized by a weakly accepting blind counter automaton C because all of its parts excluding $h(\mathsf{INVAL}(M))$ are regular and a weakly accepting blind counter automaton can be constructed for $h(\mathsf{INVAL}(M))$ as described above.

We will show that M does not halt if and only if for all z in the language L_M described by this automaton C with $|z| \geq 1$ there exist $u, v, w \in \Sigma^*$ such that $z = uvw$, $|v| = 1$ and $uv^iw \in L_M$ for all $i \geq 0$. We consider two cases:

1. If $\mathsf{VAL}(M) = \emptyset$, then $\mathsf{INVAL}(M) = \Sigma^*$. This implies that

   $$L_M = h(\Sigma^*)c\Sigma^+ \cup \Sigma^*c \cup c^*.$$

   Any word ucw from the part $h(\Sigma^*)c\Sigma^+$ can be pumped by any single letter in w. Even if $|w| = 1$ and $i = 0$ we get a word in Σ^*c. Conversely, any word $wc \in \Sigma^*c$ can be pumped by a single letter in w for $|w| > 0$. If $w = \epsilon$, c can be pumped instead.

2. If $u \in \mathsf{VAL}(M) \neq \emptyset$, consider the word $z = h(u)caa \in L_M$ as a counterexample. Pumping a single letter in $h(u)$ can can result in an odd length before c, which leads out of L_M. Pumping c also leads out of L_M. Pumping a single letter a after c can make the length after c odd, thus leading out of $\{aa\}^*$ and consequently out of L_M again.

□

Remark 2.2 *Since the halting problem for a Turing machine reduces to the halting problem of a 2-counter Minsky machine [5], the* PUMPING-PROBLEM *for 1-turn weakly accepting blind counter automata is complete for the set of complements of the recursive enumerable languages* co-RE.

Remark 2.3 *We can also fix any other constant $p > 0$ as a special case and obtain the same undecidability by using $h(a) = d^{p+1}$ for all $d \in \Sigma$ and the counterexample $z = h(u)ca^{p+1} \in L_M$. It does not even make a difference here whether the version of the pumping lemma allows pumping one (like for regular languages), two (like for context-free languages), or even more parts of the word z simultaneously.*

3. Open Problem

A stronger restriction of 1-turn could be considered, in which the increment operations have to occur before the decrement operations. This would restrict the counter to positive contents. This corresponds to the trio generated by $\{a^n b^m \mid n > m\}$. It is open whether the PUMPING-PROBLEM is still undecidable in this case.

References

[1] J. BERSTEL, *Transductions and Context-Free Languages*. Vieweg+teubner Verlag, 1979.

[2] H. GRUBER, M. HOLZER, C. RAUCH, The Pumping Lemma for Regular Languages is Hard. In: B. NAGY (ed.), *Implementation and Application of Automata - 27th International Conference, CIAA 2023, Famagusta, North Cyprus, September 19-22, 2023, Proceedings*. Lecture Notes in Computer Science 14151, Springer, 2023, 128–140.
https://doi.org/10.1007/978-3-031-40247-0_9

[3] H. GRUBER, M. HOLZER, C. RAUCH, The Pumping Lemma for Context-Free Languages is Undecidable. In: J. D. DAY, F. MANEA (eds.), *Developments in Language Theory - 28th International Conference, DLT 2024, Göttingen, Germany, August 12-16, 2024, Proceedings*. Lecture Notes in Computer Science 14791, Springer, 2024, 141–155.
https://doi.org/10.1007/978-3-031-66159-4_11

[4] J. E. HOPCROFT, J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

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

THEORIE-AG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 65–68.

Finite Automata with Translucent Words: Many Variants and Open Problems

Friedrich Otto

Fachbereich Elektrotechnik/Informatik, Universität Kassel
D-34109 Kassel
f.otto@uni-kassel.de

Abstract

Here we present various ways in which the finite automaton with translucent words can be modified and extended. In particular, we concentrate on the requirements for the sets of translucent words and the definition of the induced computation relation.

1. Introduction

The *finite automaton with translucent words* was introduced in [6] as a generalization of the *finite automaton with translucent letters* [5] (see also [7]). A finite automaton with translucent words, an *NFAwtw* for short, has an associated finite set of *translucent words* for each state, and in each step, it reads the first letter that is only preceded by a product of words that are translucent for the current state. If and when the tape only contains a word that is a product of words that are translucent for the current state, then the automaton halts, and it accepts if this state is final. In [6] (and also in [2]) it is required that, for each state q, the associated set of translucent words $\tau(q)$ satisfies the following two technical restrictions:

1. $\tau(q)$ is a finite prefix code and
2. no word in the set $\tau(q)$ begins with a letter that the automaton can read in state q.

Due to these restrictions, the single-step computation relation induced on the set of configurations of an NFAwtw can be computed by simply reading the tape letter-by-letter from left to right.

In this note, we consider various options for extending or modifying the NFAwtw, where we concentrate on the following parameters:

(a) The form of the finite sets of translucent words: Should they be required to be prefix codes, codes, or should we admit just any finite sets?

(b) The relationship between the translucent words and the letters that are processed: Should no word from a set of translucent words begin with a letter that the automaton considered can process in the corresponding state, or should that be allowed?

(c) The definition of the single-step computation relation: Should always the letter be read that is preceded by the longest prefix that can be written as a product of translucent words for the current state?

(d) Determinism versus nondeterminism: The transition relation of an NFAwtw is, in general, nondeterministic, but we may also choose to admit nondeterminism in the choice of the prefix ignored as in [3].

2. New Variants

Here we modify the definition of the finite automaton with translucent words from [6] by incorporating the final states into the transition function.

Definition 2.1 *A* finite automaton with translucent words*, or an* NFAwtw*, is defined by a 6-tuple* $A = (Q, \Sigma, \lhd, \tau, I, \delta)$*, where* Q *is a finite set of states,* Σ *is a finite input alphabet,* $\lhd$ *is a special letter that serves as an end-of-tape marker,* $I \subseteq Q$ *is a set of initial states,* $\tau : Q \to \mathcal{P}_{\text{fin}}(\Sigma^*)$ *is a* translucency mapping*, and*

$$\delta : Q \times (\Sigma \cup \{\lhd\}) \to (\mathcal{P}(Q) \cup \{\mathsf{Accept}, \mathsf{Reject}\})$$

is a transition function. Here we require that, for each state $q \in Q$ *and each letter* $a \in \Sigma$*,* $\delta(q,a) \subseteq Q$ *and* $\delta(q, \lhd) \in \{\mathsf{Accept}, \mathsf{Reject}\}$.

For each state $q \in Q$*, let* $\Sigma_q^{(A)} = \{\, a \in \Sigma \mid \delta(q,a) \neq \emptyset \,\}$*, that is,* $\Sigma_q^{(A)}$ *contains those letters that* A *can read in state* q*.*

(a) *The automaton* A *is called* prefix-restricted *if* $\tau(q)$ *is a finite prefix code for each state* $q \in Q$*.*

(b) *The automaton* A *is called* code-restricted *if* $\tau(q)$ *is a finite code for each state* $q \in Q$*, but not necessarily a prefix code.*

(c) *The automaton* A *is called* unrestricted *if* $\tau(q)$ *is just a finite set for each state* $q \in Q$*, but not necessarily a code.*

(d) *The automaton* A *is called* dependent *if, for each state* $q \in Q$*, no word in the set* $\tau(q)$ *begins with a letter from the set* $\Sigma_q^{(A)}$ *– otherwise, the automaton* A *is called* independent.

An NFAwtw as defined in [6] is required to be prefix-restricted and dependent. In the general case, there are various options of how to define the single-step computation relation for an NFAwtw.

Definition 2.2 *Let* $Z \subseteq \Sigma^*$ *be a finite language. For a word* $w \in \Sigma^*$*, we define its locally and globally maximal prefixes with respect to* Z *as follows:*

- *A word* $u \in \Sigma^*$ *is a* locally maximal prefix *of* w *with respect to* Z*, if* $w = uv$ *for some word* $v \in \Sigma^*$ *such that* $v \notin Z \cdot \Sigma^*$*.*
- *A word* $u \in \Sigma^*$ *is a* globally maximal prefix *of* w *with respect to* Z*, if* $w = uv$ *for some word* $v \in \Sigma^*$ *such that no longer prefix of* w *belongs to the set* Z^**.*

For specifying the single-step computation relation of an NFAwtw that is just code-restricted or even unrestricted, we may use the notion of globally maximal prefix or the notion of locally maximal prefix.

Definition 2.3 *Let $A = (Q, \Sigma, \lhd, \tau, I, \delta)$ be an NFAwtw.*

(a) *The automaton A is said to work in* global mode *if the single-step computation relation $\vdash_A$ is defined as follows, where $q \in Q$ and $w \in \Sigma^*$:*

$$qw \cdot \lhd \vdash_A \begin{cases} q'uv \cdot \lhd, \text{ if } w = uav,\ u \in (\tau(q))^* \text{ is the globally maximal prefix of } w \\ \qquad\qquad \text{wrt. } \tau(q), a \in \Sigma_q^{(A)},\ v \in \Sigma^*, \text{ and } q' \in \delta(q, a), \\ \mathsf{Accept}, \text{ if } w \in (\tau(q))^* \text{ and } \delta(q, \lhd) = \mathsf{Accept}, \\ \mathsf{Reject}, \text{ otherwise}. \end{cases}$$

(b) *The automaton A is said to work in* local mode *if the single-step computation relation $\vdash_A$ is defined as follows, where $q \in Q$ and $w \in \Sigma^*$:*

$$qw \cdot \lhd \vdash_A \begin{cases} q'uv \cdot \lhd, \text{ if } w = uav,\ u \in (\tau(q))^* \text{ is a locally maximal prefix of } w \\ \qquad\qquad \text{wrt. } \tau(q), a \in \Sigma_q^{(A)},\ v \in \Sigma^*, \text{ and } q' \in \delta(q, a), \\ \mathsf{Accept}, \text{ if } w \in (\tau(q))^* \text{ and } \delta(q, \lhd) = \mathsf{Accept}, \\ \mathsf{Reject}, \text{ otherwise}. \end{cases}$$

(c) *The automaton A is said to work in* non-maximal mode *if the single-step computation relation $\vdash_A$ is defined as follows, where $q \in Q$ and $w \in \Sigma^*$:*

$$qw \cdot \lhd \vdash_A \begin{cases} q'uv \cdot \lhd, \text{ if } w = uav,\ u \in (\tau(q))^*, a \in \Sigma_q^{(A)},\ v \in \Sigma^*, \text{ and } q' \in \delta(q, a), \\ \mathsf{Accept}, \text{ if } w \in (\tau(q))^* \text{ and } \delta(q, \lhd) = \mathsf{Accept}, \\ \mathsf{Reject}, \text{ otherwise}. \end{cases}$$

Lemma 2 in [6] shows that, for a prefix-restricted NFAwtw $A = (Q, \Sigma, \lhd, \tau, I, \delta)$ that is dependent, $puav \cdot \lhd \vdash_A quv \cdot \lhd$ implies $puavw \cdot \lhd \vdash_A quvw \cdot \lhd$ and $paw \cdot \lhd \vdash_A qw \cdot \lhd$ for each word $w \in \Sigma^*$, where $p \in Q$, $u \in (\tau(p))^*$, $a \in \Sigma_p^{(A)}$, and $v \in \Sigma^*$. However, if we consider an NFAwtw that is prefix-restricted and independent, or that is only code-restricted and dependent, then, in general, this implication does not hold anymore.

If $A = (Q, \Sigma, \lhd, \tau, I, \delta)$ is a prefix-restricted NFAwtw that is dependent, then $L(A)$ is letter-equivalent to a regular subset of $L(A)$, and therewith, it is semi-linear. However, the proof as described in [4] *does not work* if the sets $\tau(q)$ are allowed to contain words that begin with a letter that A can read in state q. On the other hand, we have the following positive result.

Theorem 2.4 *If A is an independent and unrestricted DFAwtw that works in global mode. then the complement of the language $L(A)$ is accepted by an automaton of the same type.*

Recall from [6] that the complement of a language accepted by a prefix-restricted, dependent DFAwtw is accepted by a prefix-restricted, dependent NFAwtw, but it is still open whether the class of languages accepted by prefix-restricted, dependent DFAwtws is closed under complementation.

3. Problems

By relaxing the definition of the sets of translucent words to be either just codes or to be independent we loose some of the major technical properties of the single-step computation relation of an NFAwtw. Concerning the DFAwtw and the NFAwtw that are *code-restricted* and/or *independent*, the following problems are still unsolved.

Research Problem 3.1 (a) *How do the language classes accepted by these DFAwtws or NFAwtws relate to the classes* $\mathcal{L}(\mathsf{DFAwtw})$ *and* $\mathcal{L}(\mathsf{NFAwtw})$ *of [6]?*

(b) *Is the language accepted by such an NFAwtw always semi-linear?*

(c) *What is the complexity of the membership problem for a DFAwtw of this form?*

(d) *Study the closure and non-closure properties of the language classes defined by these DFAwtws and NFAwtws!*

(e) *Study the* repetitive *and the* non-returning *variants of these DFAwtws and NFAwtws (see, e.g., [1] for the corresponding definitions)!*

References

[1] F. MRÁZ, F. OTTO, Repetitive finite automata with translucent letters. In: F. MANEA, G. PIGHIZZINI (eds.), *14th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2024), Proc.*. EPTCS 407, 2024, 150–167.

[2] F. MRÁZ, F. OTTO, On a measure for the descriptional complexity of finite automata with translucent words. In: A. MALCHER, L. PRIGIONIERO (eds.), *DCFS 2025, Proc.*. Lecture Notes in Computer Science 15759, Springer, Cham, Switzerland, 2025, 180–195.

[3] B. NAGY, State-deterministic finite automata with translucent letters and finite automata with nondeterministically translucent letters. In: Z. GAZDAG, I. SZABOLCS, G. KOVÁSZNAI (eds.), *16th International Conference on Automata and Formal Languages (AFL 2023)*. EPTCS 386, 2023, 170–184.

[4] B. NAGY, F. OTTO, A two-dimensional infinite hierarchy for finite automata with translucent words. *JALC*. Extended version of [6], accepted for publication.

[5] B. NAGY, F. OTTO, Finite-state acceptors with translucent letters. In: G. BEL-ENGUIX, V. DAHL, A. O. DE LA PUENTE (eds.), *BILC 2011: AI Methods for Interdisciplinary Research in Language and Biology, Proc.*. SciTePress, Portugal, 2011, 3–13.

[6] B. NAGY, F. OTTO, Finite automata with sets of translucent words. In: J. D. DAY, F. MANEA (eds.), *DLT 2024, Proc.*. Lecture Notes in Computer Science 14791, Springer, Cham, Switzerland, 2024, 236–251.

[7] F. OTTO, A survey on automata with translucent letters. In: B. NAGY (ed.), *CIAA 2023, Proc.*. Lecture Notes in Computer Science 14151, Springer, Cham, Switzerland, 2023, 21–50.

THEORIE-AG 2025
B. Truthe (Hrsg.): 35. Theorietag GI-Fachgruppe AFS, Schotten, 30.9. – 2.10.2025
Justus-Liebig-Universität Gießen, IFIG Research Report 2501, S. 69–69.

Constraint Automata on Infinite Data Trees: Decision Procedures for Constraint CTL

Karin Quaas

Universität Leipzig
quaas@informatik.uni-leipzig.de

We introduce the class of *tree constraint automata*. A tree constraint automaton is equipped with a finite set of variables ranging over the integers. Along a transition, the values of these variables can be restricted to satisfy certain *constraints*, which are Boolean formulas over the variables using the usual order over the integers and equality to constants. For example, the constraint $x_1 < x_2'$ expresses that the value of x_1 at the current position is strictly smaller than the value of x_2 at the next position in a run (we use the prime in x_2' to refer to the value of x_2 at the next position), and the constraint $x_2 = 5$ expresses that the value of x_2 at the current position must be equal to 5. A run of a tree constraint automaton is an infinite tree, and such a run is accepting if in each branch of the tree an accepting state is visited infinitely often. Deciding whether a given tree constraint automaton has an accepting run is a nontrivial problem: even though the constraints can only restrict the values of local variables, such local constraints can be propagated to formalize constraints on variables whose distance in a run is a priori unbounded. We prove that this problem can be decided in EXPTIME (compare with the polynomial time upper bound of the same problem for tree automata without constraints). As an application, we use the classical automata-based approach for temporal logics and prove the precise computational complexity of the satisfiability problem for the branching-time logics CTL and CTL* with constraints in the integers (only decidability was known so far). More detailed, we prove that the satisfiability problem for CTL with constraints is EXPTIME-complete, and the satisfiability problem for CTL* with constraints is 2EXPTIME-complete.

This is joint work with Stéphane Demri, published in Logical Methods in Computer Science 21(2) (2025).

Institut für Informatik

Justus-Liebig-Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany

Recent Reports

(Further reports are available at `www.informatik.uni-giessen.de`.)

M. Holzer *A Short Comment on Controlled Context-Free Grammar Derivations*, Report 2402, July 2024.

H. Gruber, M. Holzer, C. Rauch *On Pumping Preserving Homomorphisms and the Complexity of the Pumping Problem*, Report 2401, June 2024.

M. Holzer, *Comments on Monoids Induced by NFAs*, Report 2301, March 2023.

S. Beier, M. Holzer, *Semi-Linear Lattices and Right One-Way Jumping Finite Automata*, Report 1901, April 2019.

M. Kutrib, T. Worsch, *Self-Verifying Cellular Automata*, Report 1803, April 2018.

S. Beier, M. Holzer, *Properties of Right One-Way Jumping Finite Automata*, Report 1802, March 2018.

B. Truthe, *Hierarchy of Subregular Language Families*, Report 1801, February 2018.

M. Holzer, M. Hospodár, *On the Magic Number Problem of the Cut Operation*, Report 1703, October 2017.

M. Holzer, S. Jakobi, *A Note on the Computational Complexity of Some Problems for Self-Verifying Finite Automata*, Report 1702, April 2017.

S. Beier, M. Holzer, M. Kutrib, *On the Descriptional Complexity of Operations on Semilinear Sets*, Report 1701, April 2017.

M. Holzer, S. Jakobi, M. Wendlandt, *On the Computational Complexity of Partial Word Automata Problems*, Report 1404, May 2014.

H. Gruber, M. Holzer, *Regular Expressions From Deterministic Finite Automata, Revisited*, Report 1403, May 2014.

M. Kutrib, A. Malcher, M. Wendlandt, *Deterministic Set Automata*, Report 1402, April 2014.

M. Holzer, S. Jakobi, *Minimal and Hyper-Minimal Biautomata*, Report 1401, March 2014.

J. Kari, M. Kutrib, A. Malcher (Eds.), *19th International Workshop on Cellular Automata and Discrete Complex Systems AUTOMATA 2013 Exploratory Papers*, Report 1302, September 2013.

M. Holzer, S. Jakobi, *Minimization, Characterizations, and Nondeterminism for Biautomata*, Report 1301, April 2013.

A. Malcher, K. Meckel, C. Mereghetti, B. Palano, *Descriptional Complexity of Pushdown Store Languages*, Report 1203, May 2012.

M. Holzer, S. Jakobi, *On the Complexity of Rolling Block and Alice Mazes*, Report 1202, March 2012.

M. Holzer, S. Jakobi, *Grid Graphs with Diagonal Edges and the Complexity of Xmas Mazes*, Report 1201, January 2012.

H. Gruber, S. Gulan, *Simplifying Regular Expressions: A Quantitative Perspective*, Report 0904, August 2009.

M. Kutrib, A. Malcher, *Cellular Automata with Sparse Communication*, Report 0903, May 2009.

M. Holzer, A. Maletti, *An $n \log n$ Algorithm for Hyper-Minimizing States in a (Minimized) Deterministic Automaton*, Report 0902, April 2009.

H. Gruber, M. Holzer, *Tight Bounds on the Descriptional Complexity of Regular Expressions*, Report 0901, February 2009.

M. Holzer, M. Kutrib, and A. Malcher (Eds.), *18. Theorietag Automaten und Formale Sprachen*, Report 0801, September 2008.

M. Holzer, M. Kutrib, *Flip-Pushdown Automata: Nondeterminism is Better than Determinism*, Report 0301, February 2003

M. Holzer, M. Kutrib, *Flip-Pushdown Automata: $k+1$ Pushdown Reversals are Better Than k*, Report 0206, November 2002

M. Holzer, M. Kutrib, *Nondeterministic Descriptional Complexity of Regular Languages*, Report 0205, September 2002

H. Bordihn, M. Holzer, M. Kutrib, *Economy of Description for Basic Constructions on Rational Transductions*, Report 0204, July 2002

M. Kutrib, J.-T. Löwe, *String Transformation for n-dimensional Image Compression*, Report 0203, May 2002

A. Klein, M. Kutrib, *Grammars with Scattered Nonterminals*, Report 0202, February 2002

A. Klein, M. Kutrib, *Self-Assembling Finite Automata*, Report 0201, January 2002